Ovaj diplomski rad obranjen je dana _____
pred nastavničkim povjerenstvom u sastavu:

1. _____ , predsjednik
2. _____ , član
3. _____ , član

Povjerenstvo je rad ocijenilo ocjenom _____ .

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sveučilište u Zagrebu
PMF-Matematički odjel

Igor Krnić

TEORIJA SLOŽENOSTI

Diplomski rad

Zagreb, rujan 2006.

Sadržaj

1 Uvod **1**
1.1 Osnovni pojmovi 1
1.2 Konačni automat 2
1.3 Turingov stroj 3
1.3.1 Deterministički Turingov stroj 4
1.3.2 Nedeterministički Turingov stroj 6
1.3.3 Univerzalni Turingov stroj 9
1.3.4 Zaključak 10
1.4 Rekurzivni i rekurzivno prebrojivi jezici 10
1.5 Grafovi 15

2 DTIME i DSPACE **18**
2.1 Uvod 18
2.2 Osnovne klase složenosti 18
2.3 Primjeri problema u PTIME 20
2.4 Odnosi među klasama složenosti 23

3 NTIME i NSPACE **39**
3.1 Uvod 39
3.2 Nedeterminističke klase složenosti 39
3.3 Primjeri problema u klasi NP 42
3.4 Opći rezultati o odnosima klasa složenosti 48

4 NP-potpunost **53**
4.1 Uvod 53
4.2 Polinomna reducibilnost i NP-potpunost 54
4.3 Cook-Levinov Teorem 56
4.4 Zaključak 65

Literatura **67**

1 Uvod

1.1 Osnovni pojmovi

Da bismo krenuli dalje, potrebno je formalizirati neke pojmove, što je i cilj ovog poglavlja. Krenimo od možda najosnovnijeg pojma, tj. znakova i nizova znakova.

Definicija 1.1

- *Proizvoljan konačan skup zvat ćemo* ***abeceda*** *i označavat ćemo ga sa* Σ.
- *Elemente abecede zvat ćemo* ***znakovi***.
- *Konačan niz znakova iz neke abecede* Σ *zvat ćemo* ***riječ***.
- ***Duljina*** *riječi jednaka je broju znakova od kojih se ta riječ sastoji.*
- *Sa* Σ^n *označavat ćemo skup svih riječi duljine* n.
- *Sa* Σ^* *označavat ćemo skup svih riječi nad abecedom* Σ.
- *Proizvoljan skup riječi iz neke abecede* Σ *zvat ćemo* ***jezik***.

Smatramo da svaki jezik (neovisno o abecedi) sadrži praznu riječ. Praznu riječ označavamo sa $*$. Za proizvoljnu riječ $w \in \Sigma^*$, sa $|w|$ označavat ćemo duljinu riječi w. Napomenimo da je prazna riječ, riječ duljine 0.
Od uređaja na skupu Σ^* koristit ćemo najčešće leksikografski uređaj, koji je induciran proizvoljnim uređajem na Σ.

Često složenost nekog algoritma, procedure ili postupka promatramo u odnosu na duljinu ulaznih podataka, odnosno složenost neke procedure se u ovisnosti o duljini ulaza, može prikazati kao funkcija sa skupa $\mathbf{N}$ u skup $\mathbf{R}$. Takve funkcije nazivamo funkcije složenosti. Naravno, u slučaju da imamo dvije različite procedure za rješavanje istog problema, postavlja se pitanje usporedbe njihovih funkcija složenosti. U tu svrhu uvedimo sljedeću definiciju.

Definicija 1.2 *Neka su* $f, g \colon \mathbf{N} \to \mathbf{R}$ *dvije realne funkcije nad skupom prirodnih brojeva. Tada pišemo:*

a) $f = O(g)$ *, ako postoji konstanta* $c \in \mathbf{R}, c \geq 0$ *takva da za sve dovoljno velike* $n \in \mathbf{N}$ *vrijedi* $|f(n)| \leq c|g(n)|$.

b) $f = o(g)$ *, ako postoji limes* $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)}$ *i jednak je* 0.

c) $f = \Omega(g)$ *, ako je* $g = O(f)$*.*

d) $f = \Theta(g)$ *, ako vrijedi i* $f = O(g)$ *i* $g = O(f)$*.*

Naravno, postavlja se pitanje što je to algoritam ili procedura, točnije, koja je njihova stroga matematička definicija. Nakon toga, prirodno se nameće pitanje računala pa zatim pitanje reprezentativnog računala...
Rješenje je nađeno uvođenjem idealiziranih, apstraktnih računala koje zovemo računalni modeli ili matematički strojevi. Svaki od tih modela promatra intuitivni pojam računala iz nekoliko aspekata zanemarujući možda pritom druge aspekte. Samim time jasno je da će postojati razni modeli. Ipak, za ovaj je rad ključan samo jedan od njih, Turingov stroj. No, prije Turingovog stroja spomenut ćemo najjednostavniji računalni model, konačni automat.

1.2 Konačni automat

Konačni automat[1] je jako jednostavan računalni model. Dobro modelira jednostavno računalo s iznimno malom količinom memorije.
Rad mu se temelji na promjeni stanja ovisno o ulaznom podatku, te emitiranju izlaznog podatka.
Točnije, formalna definicija je sljedeća:

Definicija 1.3 ***Konačni automat*** *je uređena petorka* $(\Sigma, \Sigma', \Gamma, \delta, q_0)$ *gdje je:*

- Σ*, odnosno* Σ'*, abeceda koju zovemo ulazna, odnosno izlazna abeceda*
- Γ *konačan skup kojeg nazivamo skup unutarnjih stanja, a njegove elemente unutarnja stanja (ili samo stanja)*
- $\delta: \Gamma \times \Sigma \to \Gamma \times \Sigma'$ *funkcija koja ovisno o trenutnom unutarnjem stanju automata te ulaznom znaku, automatu pridružuje: novo stanje u koje automat prelazi, te znak kojeg automat daje kao izlaz*
- $q_0 \in \Gamma$*, unutarnje stanje kojega nazivamo početno stanje (automat se nalazi u stanju* q_0 *na početku rada).*

Napomena 1.1 *Postoje razne definicije konačnog automata, međutim, sve su one ekvivalentne.*

Sada krenimo na najbitniji računalni model, bar iz aspekta ovog rada, Turingov stroj.

[1] engleski: finite automaton

1.3 Turingov stroj

Povijesno gledano, Turingov stroj je prvi pravi računalni model. Uveo ga je engleski matematičar Alan Turing 1936. godine, dakle prije izuma modernog, programabilnog računala. Opisno, Turingov stroj je konačni automat opremljen neograničenom količinom memorije.

Centralni dio svakog Turingovog stroja je kontrolna jedinica koja je u biti, konačni automat. Njen je zadatak rukovođenje ostalim dijelovima stroja. Jedan od tih dijelova je memorija.
Memorija Turingovog stroja dana je u obliku jedne ili k traka ($k > 1$), koje su s obje strane neograničene. Svaka traka podijeljena je na ćelije, njih prebrojivo mnogo. Ćelije su indeksirane skupom cijelih brojeva, $\mathbf{Z}$[2]. Od ćelija se posebno ističe ona s indeksom 0, koju zovemo početna ili nulta ćelija. Sadržaj ćelija (jedan znak), potpuno je određen elementima zadane abecede Σ, koju zovemo abeceda stroja. Spomenimo posebno jedan znak, takozvani prazan znak (oznaka $*$[3]). Uvedimo sada i oznaku Σ_0 koja će nam označavati skup $\Sigma \backslash \{*\}$.

Kako Turingov stroj ima memoriju, mora imati i mehanizam za djelovanje nad njom. Taj mehanizam čine glave Turingovog stroja, spojene na kontrolnu jedinicu. Njihov je zadatak čitanje, odnosno pisanje podataka.
Na početku rada stroja sve su glave pozicionirane na nultoj ćeliji. Također, na početku rada Turingovog stroja, sve ćelije, osim onih s ulaznim podacima, sadrže prazan znak, $*$.
Pod ulaznim (izlaznim) podacima smatramo uređenu k-torku riječi iz Σ^*, koja se nalazi na trakama prije (nakon) početka (završetka) rada stroja. Najčešće ćemo imati jedan ulazni (izlazni) podatak koji će biti zapisan na prvoj (zadnjoj) traci. Zato ćemo i govoriti da je ulazni (izlazni) podatak stroja riječ $w \in \Sigma^*$, imajući na umu da pri tome mislimo na k-torku $(w, *, \ldots, *)$ $((*, \ldots, *, w))$.

Istaknimo još dva bitna stanja stroja, START i STOP. START je stanje u kojem se stroj nalazi prilikom započinjanja s radom, a STOP stanje u kojem se stroj nalazi kada stane, odnosno završi s radom.

Rad Turingovog stroja, opisno, odvija se na sljedeći način:
Pročitaj znakove koji se trenutno nalaze u ćelijama iznad kojih su pozicionirane glave. Ovisno o pročitanim znakovima i unutarnjem stanju stroja kontrolna jedinica radi:

[2] Primijetimo da kako je $\mathbf{Z}$ ekvipotentan sa $\mathbf{N}$, ćelije možemo indeksirati i s $\mathbf{N}$. To je ekvivalentna varijanta Turingovog stroja kada su trake neograničene samo s jedne strane.

[3] Oznaka nam je ista kao i za praznu riječ.

- Šalje nove znakove, koje glave trebaju upisati u ćelije iznad kojih se nalaze.

- Šalje naredbu svakoj od glava, da se pomakne jedno mjesto u lijevo, ili desno ili da ostane na mjestu.

- Prelazi u novo stanje (koje može biti isto kao i staro).

Postoje dva različita oblika Turingovog stroja: deterministički i nedeterministički. Iako ćemo vidjeti da su jednako "snažni", oni predstavljaju dva bitno različita načina razmišljanja i djelovanja. Ta razlika, dovest će, kasnije u ovom radu, do zanimljivih rezultata, i otvoriti pitanja koja su i danas neodgovorena. Pitanja koja spadaju među najvažnija u teorijskom računarstvu.
Krenimo prvo s definicijom determinističkog Turingovog stroja.

1.3.1 Deterministički Turingov stroj

Definicija 1.4 ***Deterministički Turingov stroj*** *(oznaka DTS) T je uređena četvorka $T = (k, \Sigma, \Gamma, \delta)$ pri čemu je:*

- $k \geq 1$ *prirodan broj koji označava broj traka od T*
- Σ *abeceda koju zovemo abeceda stroja*
- Γ *konačan skup kojeg zovemo skup unutarnjih stanja takav da je* START, STOP $\in \Gamma$ *(elemente od Γ zovemo unutarnja stanja)*
- $\delta: \Gamma \times \Sigma^k \rightarrow \Gamma \times \Sigma^k \times \{-1, 0, 1\}^k$ *funkcija (zvat ćemo je funkcija prijelaza) koja, ovisno o trenutnom stanju stroja i pročitanoj k-torci znakova s traka, stroju pridružuje: novo stanje u koje stroj prelazi, k-torku znakova koje treba upisati na trake te pomak svake od glava (jedno mjesto u lijevo, ili desno ili je ostavlja na miru).*

Definicija 1.5 *Kažemo da DTS $T = (k, \Sigma, \Gamma, \delta)$,* ***prihvaća***[4] ***riječ*** *$w \in \Sigma_0^*$, ako za ulaz w, T stane u konačno mnogo koraka, i u nultoj ćeliji prve trake, zapisan je simbol* 1.
Kažemo da T ***ne prihvaća riječ*** *$w \in \Sigma_0^*$, ako za ulaz w, T stane u konačno mnogo koraka i u nultoj ćeliji prve trake, zapisan je simbol* 0.

Napomena 1.2 *Primijetimo da osim dva moguća slučaja iz prethodne definicije, postoji i mogućnost da se T za ulaz w ne zaustavi, odnosno da radi beskonačno. To je nepoželjno svojstvo, jer tada ne možemo odlučiti prihvaća li stroj danu riječ ili ne.*

[4] engleski: accepts

Definicija 1.6 *Kažemo da DTS $T = (k, \Sigma, \Gamma, \delta)$* ***prepoznaje***[5] ***jezik*** *$\alpha \subseteq \Sigma_0^*$, ako T prihvaća svaku riječ $w \in \alpha$.*
Kažemo da je jezik $\alpha \subseteq \Sigma_0^*$ ***Turing prepoznatljiv***[6]*, ako postoji DTS koji ga prepoznaje.*

Napomena 1.3 *Prethodna definicija garantira zaustavljanje stroja T samo za riječi koje pripadaju jeziku α. Za sve ostale riječi ne znamo kako će T reagirati.*

U definiciji 1.4 vidimo da smo dozvolili da Turingov stroj ima proizvoljan konačan broj traka. Prirodno se nameće pitanje jesu li strojevi s različitim brojem traka jednako snažni. Pokazano je da jesu, te vrijedi sljedeći teorem.

Teorem 1.1 *Za svaki k-tračni DTS $S = (k, \Sigma, \Gamma, \delta)$ postoji 1-tračni DTS $T = (1, \Sigma, \Gamma_T, \delta_T)$ koji zamjenjuje S u sljedećem smislu:*
Za svaku riječ $x \in \Sigma_0^*$, stroj S s ulazom x, staje u konačno mnogo koraka ako i samo ako T s ulazom x staje u konačno mnogo koraka, i nakon što oba stroja stanu, isti je rezultat zapisan na zadnjoj traci od S i prvoj od T. Nadalje, ako S radi N koraka tada T radi $O(N^2)$ koraka.

Napomena 1.4 *Prethodni teorem*[7] *dozvoljava nam da koristimo oba stroja ravnopravno, s time da obratimo pozornost na kvadratičan rast broja koraka. Dalje kroz rad, pogotovo u dokazima teorema, koristit ćemo i višetračni i jednotračni Turingov stroj ovisno o tome koji će nam biti jednostavnije konstruirati.*

Sljedeća dva pojma biti će nam jako bitna u nastavku rada.

Definicija 1.7 ***Vremenska složenost***[8] *DTS T je funkcija $time_T \colon \mathbf{N} \to \mathbf{N}$, takva da je $time_T(n)$ jednak maksimalnom broju koraka koje stroj T radi za sve ulazne podatke duljine n.*
Prostorna složenost[9] *DTS T je funkcija $space_T \colon \mathbf{N} \to \mathbf{N}$, takva da je $space_T(n)$ jednak maksimalnom broju ćelija (uzimajući u obzir sve trake) po kojima T piše, za sve ulazne podatke duljine n.*

Napomena 1.5 *Pretpostavljamo da je $time_T(n) > n$, jer stroj mora barem pročitati ulazni podatak, te još jedan znak koji označava kraj ulaza. Također,*

[5] engleski: recognizes
[6] engleski: Turing-recognizable
[7] Za dokaz vidi [3, str. 14].
[8] engleski: time complexity
[9] engleski: space complexity

primijetimo da se može dogoditi da je $time_T(n) = \infty$*, ako Turingov stroj za neki ulaz duljine* n *ne stane.*
Slično, pretpostavit ćemo da Turingov stroj mora imati neki izlaz, pa je zato, $space_t(n) \geq 1$*.*

Sada, kada smo izložili nabitnije činjenice vezane uz deterministički Turingov stroj, možemo preći na nedeterministički i na odnos među njima.

1.3.2 Nedeterministički Turingov stroj

Krenimo odmah s definicijom.

Definicija 1.8 ***Nedeterministički Turingov stroj***
(oznaka NTS) T *je uređena četvorka* $T = (k, \Sigma, \Gamma, \Phi)$ *pri čemu je:*

- $k \geq 1$ *prirodan broj koji označava broj traka od* T
- Σ *abeceda koju zovemo abeceda stroja*
- Γ *konačan skup kojeg zovemo skup unutarnjih stanja takav da je* START, STOP $\in \Gamma$
- $\Phi \subseteq (\Gamma \times \Sigma^k) \times (\Gamma \times \Sigma^k \times \{-1, 0, 1\}^k)$ *proizvoljna relacija.*

Vidimo da su u definiciji 1.4 i definiciji 1.8, uglavnom svi pojmovi identični. Jedina razlika leži u tome što je relacija Φ u biti poopćenje funkcije prijelaza δ. Samim time, jasno je i da je deterministički Turingov stroj, poseban slučaj nedeterminističkog, kada Φ ima i funkcijska svojstva.

Nedeterminizam se najbolje očituje u tome što stroj, ovisno o istom ulazu i istom unutarnjem stanju, može istovremeno postupati na bitno različite načine. Najbolje je mogućnost postupanja na različite načine zamišljati kao grananje, a cijeli proces rada NTS, kao stablo. Često ćemo u tekstu korisiti upravo ovaj oblik izražavanja.

Definirajmo sada pojmove analogne onima iznesenima kod DTS.

Definicija 1.9 *Kažemo da NTS* $T = (k, \Sigma, \Gamma, \Phi)$*,* ***prihvaća riječ*** $w \in \Sigma_0^*$*, ako za ulaz* w*, bar jedna grana od* T *stane u konačno mnogo koraka, i u nultoj ćeliji prve trake, zapisan je simbol* 1*.*
Kažemo da T ***ne prihvaća riječ*** w*, ako za ulaz* w*, svaka grana od* T *stane u konačno mnogo koraka, i u nultoj ćeliji prve trake, zapisan je simbol* 0*.*

Napomena 1.6

- *DTS Turingov stroj možemo shvatiti kao NTS koji ima samo jednu granu. U prethodnoj definicji odmah vidimo snagu nedeterminizma, a to je istovremeno izvođenje više grana. Opisno, NTS istovremeno izvodi rad kao nekoliko DTS.*
- *Ako se niti jedna grana NTS ne zaustavi, tada stroj radi beskonačno. Prethodna definicija nam ne garantira zaustavljanje.*

Definicija 1.10 *Kažemo da NTS* $T = (k, \Sigma, \Gamma, \Phi)$ ***prepoznaje jezik*** $\alpha \subseteq \Sigma_0^*$*, ako* T *prihvaća svaku riječ* $w \in \alpha$*.*

Kao i kod pitanja odnosa između jednotračnog i višetračnog stroja, postavlja se pitanje odnosa između deterministočkog i nedeterminističkog stroja. Možda i postoje argumenti koji bi nas naveli da pomislimo suprotno, ali činjenica je da razlike u "snazi" nema. Ipak, postoji nešto u čemu je nedeterminizam neusporedivo bolji, a to je štednja vremena. Opet, sljedeći teorem iznosimo bez dokaza[10], ali kao iznimno važan rezultat.

Teorem 1.2 *Za svaki NTS* $S = (k, \Sigma, \Gamma, \Phi)$ *postoji ekvivalentan DTS* $T = (l, \Sigma, \Gamma_T, \delta)$ *koji zamjenjuje* S *u sljedećem smislu:*
Neka je $x \in \Sigma_0^*$ *proizvoljna riječ. Za ulaz* x*, postoji grana od* S *koja staje u konačno mnogo koraka ako i samo ako* T *s ulazom* x *staje u konačno mnogo koraka, i nakon što oba stroja stanu, isti je rezultat napisan na odgovarajućim trakama. Nadalje, ako* S *radi* N *koraka tada* T *radi* $O(2^N)$ *koraka.*

Definirajmo vremensku, odnosnu prostornu složenost NTS.

Definicija 1.11 ***Vremenska složenost*** *NTS* T *je funkcija* $time_T \colon \mathbf{N} \to \mathbf{N}$*, takva da je* $time_T(n)$ *jednak maksimalnom broju koraka (uzimajući u obzir sve grane) koje* T *radi za sve ulazne podatke duljine* n*.*
Prostorna složenost *NTS* T *je funkcija* $space_T \colon \mathbf{N} \to \mathbf{N}$*, takva da je* $space_T(n)$ *jednak maksimalnom broju ćelija (uzimajući u obzir sve trake i sve grane) po kojima* T *piše, za sve ulazne podatke duljine* n*.*

Napomena 1.7 *Opet, uzimamo da je*[11] $time_T(n) > n$ *i* $space_T(n) \geq 1$*.*
Primijetimo da su i kod DTS i NTS, pojam vremenske, odnosno prostorne složenosti u biti jednako definirani. U oba slučaja zanima nas broj koraka u najdužoj grani izračunavanja. Međutim, DTS uvijek ima samo jednu granu pa to nismo posebno naglasili.

[10] Za dokaz vidi [5, str. 138].
[11] Vidi napomenu 1.5.

Za kraj definirajmo dva pojma koja ćemo često koristiti u narednim poglavljima.

Definicija 1.12 *Neka je $T = (k, \Sigma, \Gamma, \Phi)$, k-tračni NTS.* ***Postupak izračunavanja*** *stroja T je konačan niz uređenih petorki*
$(q, z, q^{'}, z', p) \in \Gamma \times \Sigma^k \times \Gamma \times \Sigma^k \times \{-1, 0, 1\}^k$,
takvih da vrijedi:

- $(q, z, q^{'}, z', p) \in \Phi$ *za bilo koju petorku u nizu*
- *ako su* $(q_i, z_i, q_i^{'}, z_i{}', p_i), (q_{i+1}, z_{i+1}, q_{i+1}^{'}, z_{i+1}{}', p_{i+1})$, *dvije susjedne petorke u nizu tada je* $q_i^{'} = q_{i+1}$.

Napomena 1.8 *Analogna definicija vrijedi i za DTS, s time da za zapis funkcije prijelaza koristimo uređene petorke.*
Primijetimo da je time, postupak izračunavanja NTS istog oblika kao i onaj DTS. Samim time, možemo konstruirati DTS koji simulira postupak izračunavanja nekog NTS.

Napomena 1.9 *Neka je T, NTS ili DTS, vremenske složenosti $time_T$. Tada za ulaz duljine n, T radi najviše $time_T(n)$ koraka. Time njegov postupak izračunavanja, za riječi duljine n, sadrži najviše $time_T(n)$ petorki. Kako svaku petorku možemo kodirati s konstantnim brojem znakova, slijedi da cijeli postupak izračunavanja možemo kodirati sa $O(time_T(n))$ znakova.*

Definicija 1.13 ***Konfiguracija*** *k- tračnog Turingovog stroja (DTS ili NTS) je uređena trojka (q, p, h), pri čemu je:*

- $q \in \Gamma$, *neko stanje stroja*
- $p \in \mathbf{N^k}$, *pri čemu vrijedi $-f(n) \leq p_i \leq f(n)$, za svaki $i \leq k$, gdje je $f(n)$ neka gornja ograda za broj korištenih ćelija u postupku izračunavanja, za riječi duljine n. Uređena k-torka p označava apsolutne pozicije glava Turingovog stroja.*
- $h \in (\Sigma^{2f(n)+1})^k$. *To su simboli koji se trenutno nalaze na svim trakama. Dovoljno je promatrati $2f(n)+1$ pozicija zbog načina na koji smo uzeli $f(n)$.*

Napomena 1.10 *Primijetimo da je konfiguraciju moguće zapisati sa*

$$1 + k + k(2(f(n) + 1),$$

odnosno $O(f(n))$ znakova.
Također, različitih konfiguracija ima najviše

$$|\Gamma|(2f(n) + 1)^k |\Sigma|^{2kf(n)+k} = 2^{O(f(n))}.$$

1.3.3 Univerzalni Turingov stroj

Postoji jedna značajna razlika između Turingovog stroja i klasičng računala: svaki novi problem zahtijeva konstrukciju novog Turingovog stroja koji bi ga rješavao, dok je kod računala dovoljno napisati novi program. Pokazat ćemo da i Turingov stroj može raditi na sličnom principu, tj. mijenjanjem samo programa.

Definicija 1.14 *Neka su* $T = (k+1, \Sigma, \Gamma_T, \delta_T)$ *i* $S = (k, \Sigma, \Gamma_S, \delta_S)$*, dva deterministička Turingova stroja* ($k \geq 1$)*, te neka je* $p \in \Sigma_0^*$*.*
Kažemo da T ***simulira*** S *s progamom* p*, ako za proizvoljne riječi* $x_1, \ldots, x_k \in \Sigma_0^*$*, stroj* T*, uz ulaz* $(x_1, \ldots, x_k, p)$*, stane u konačno mnogo koraka ako i samo ako stroj* S*, uz ulaz* $(x_1, \ldots, x_k)$*, stane u konačno mnogo koraka, i u trenutku kada se oba stroja zaustave, prvih* k *traka od* T *i* S *imaju isti sadržaj.*

Često ćemo u dokazima koristiti sljedeću činjenicu.

Teorem 1.3 *Neka su* $T = (k+1, \Sigma, \Gamma_T, \delta_T)$ *i* $S = (k, \Sigma, \Gamma_S, \delta_S)$*, dva Turingova stroja* ($k \geq 1$)*, te neka je* $p \in \Sigma_0^*$*.*
Pretpostavimo da T *simulira* S *s programom* p*. Tada vrijedi:*

$$time_T(n) \leq |p| \cdot time_S(n).$$

Definicija 1.15 *Kažemo da je* $k+1$*-tračni Turingov stroj* T *s abecedom* Σ ***univerzalan****, ako za svaki* k*-tračni Turingov stroj* S *nad abecedom* Σ*, postoji riječ (program)* $p \in \Sigma_0^*$*, s kojom* T *simulira* S*.*

Naravno, postavlja se pitanje egzistencije univerzalnog Turingovog stroja. Sljedeći teorem ćemo opet iznijeti bez dokaza[12].

Teorem 1.4 *Za svaki prirodan broj* $k \geq 1$ *i svaku abecedu* Σ *postoji* $(k+1)$*- tračni univerzalni Turingov stroj.*

Univerzalni Turingovi strojevi su interesantni jer više sliče modernim, programabilnim računalima. Oni će također igrati veliku ulogu u dokazima u nastavku ovog rada.

[12] Za dokaz vidi [3, str. 10].

1.3.4 Zaključak

Postoji mnogo računalnih modela ekvivalentnih Turingovom stroju. Može se pokazati da svi ti modeli mogu simulirati jedni druge uz najviše polinoman rast broja koraka[13], dakle dosta efikasno. Također, pokazalo se da je Turingov stroj dovoljno općenit model. Naravno, tu prvenstveno mislim na Church-Turingovu tezu. Turing je kroz svoj rad došao do pretpostavke da je sve, za što postoji algoritamsko (tu mislimo na intuitivni pojam algoritma) rješenje, moguće izračunati prikladnim algoritmom na Turingovom stroju (odnosno efektivnom konstrukcijom Turingovog stroja, čiju funkciju prijelaza možemo shvatiti kao niz instrukcija). Kako su algoritmi za Turingove strojeve teški za pratiti, a i nečitki, više ćemo, pogotovo u primjerima, korisiti intuitivni pojam algoritma. Pritom ćemo nastojati da koraci u algoritmu budu dovoljno jasni i elementarni, tako da sama konstrukcija odgovarajućeg Turingovog stroja koji bi to izvodio, bude jednostavna. Isto tako, govorit ćemo da neki algoritam zahtijeva određeno vrijeme rada, misleći pritom da možemo konstruirati Turingov stroj koji obavlja isti posao u više manje istom vremenu (naravno, u odnosu na duljinu ulaza).
S druge strane, Church je došao do istog zaljučka kao i Turing, koristeći pritom pojam rekurzivnih funkcija[14].
Nadam se da je ovih par kratkih činjenica dovoljno opravdanje za odabir Turingovog stroja za daljnji razvoj problematike.

1.4 Rekurzivni i rekurzivno prebrojivi jezici

U prethodnoj točki o Turingovim strojevima vidjeli smo da nema razlike (osim vremenske) između višetračnog i jednotračnog Turingovog stroja. Zato u sljedećim definicijama, u kojima nam broj koraka rada nije bitan, ne pravimo razliku među strojevima na temelju broja traka.

Definicija 1.16 *Kažemo da je funkcija* $f: \Sigma_0^* \to \Sigma_0^*$ ***rekurzivna*** *ili* ***izračunljiva*** [15] *ako postoji DTS takav da za svaki ulaz* $x \in \Sigma_0^*$ *stane nakon konačno mnogo koraka s* $f(x)$ *kao izlazom na prvoj traci.*

Definicija 1.17 *Kažemo da je jezik* $\alpha \subseteq \Sigma_0^*$ ***rekurzivan****, ako je rekurzivna njegova karakteristična funkcija* χ_α*, definirana sa*

$$\chi_\alpha(x) = \begin{cases} 1 & , \text{ ako je } x \in \alpha \\ 0 & , \text{ ako } x \notin \alpha \end{cases}$$

[13] Za detalje i dokaze vidi [5].
[14] Vidi sljedeće podpoglavlje.
[15] engleski: recursive, computable

Definicija 1.18 *Kažemo da DTS* ***odlučuje***[16] ***jezik*** $\alpha \subseteq \Sigma_0^*$ *ako izračunava njegovu karakterističnu funkciju.*
Kažemo da je jezik $\alpha \subseteq \Sigma_0^*$ ***odlučiv***[17] *ako postoji DTS koji ga odlučuje.*

Propozicija 1.1 *Jezik* α *je rekurzivan ako i samo ako je odlučiv.*

Dokaz:
Neka je α rekurzivan jezik. Tada je rekurzivna njegova karakteristična funkcija. Funkcija je rekurzivna ako postoji DTS koji ju izračunava. Sada prema definiciji 1.18 slijedi da je α odlučiv jezik.

Obratno, neka je α odlučiv jezik. Tada postoji DTS koji izračunava njegovu karakterističnu funkciju. No to znači da je njegova karakteristična funkcija rekurzivna. Sada prema definiciji 1.17 slijedi da je α rekurzivan jezik. ■

Napomena 1.11 *Svaki je konačan jezik rekurzivan (konačan broj riječi uvijek možemo direktno prepoznati, a sve ostale odbaciti).*
Ako je jezik rekurzivan, tada je i njegov komplement rekurzivan (samo komplementiramo izlaz Turingovog stroja koji odlučuje dani jezik).

Definicija 1.19 *Kažemo da je jezik* $\alpha \subseteq \Sigma_0^*$, ***rekurzivno prebrojiv*** *ako je* $\alpha = \emptyset$ *ili ako je slika neke rekurzivne funkcije.*

Napomena 1.12 *Ako je* $f: \Sigma_0^* \to \Sigma_0^*$, *rekurzivna funkcija i ako je* α *njena slika, vrijedi:* $\alpha = \{f(w_0), f(w_1), \ldots\}$, *gdje je* $\Sigma_0^* = \{w_0, w_1, \ldots\}$. *Sada je puno jasnije zašto* α *nazivamo rekurzivno prebrojiv jezik.*

Pogledajmo sada neka svojstva i karakterizacije rekurzivnih i rekurzivno prebrojivih jezika.

Propozicija 1.2 *Jezik* α *je rekurzivno prebrojiv ako i samo ako je Turing prepoznatljiv.*

Dokaz:
Neka je α rekurzivno prebrojiv. Tada postoji rekurzivna funkcija f takva da je α njena slika. Traženi Turingov stroj neka za ulaz x računa redom $f(y)$ za svaki $y \in \Sigma_0^*$. Ako za neki $y \in \Sigma_0^*$ bude vrijedilo $f(y) = x$, tada će stroj prihvatiti x.
Primijetimo, ako je $x \in \alpha$, tada će stroj kad tad stati, a ako nije tada ne znamo kako će se ponašati. No, to je dovoljno za Turing prepoznatljivost.

[16] engleski: decides
[17] engleski: decidable

Obratno, neka je α Turing prepoznatljiv, te neka DTS T prepoznaje α. Ako je α prazan, tada je po definiciji rekurzivno prebrojiv. Zato pretpostavimo da nije prazan, odnosno da postoji $w_0 \in \alpha$.
Definirajmo funkciju $f\colon \mathbf{N} \to \Sigma_0^*$, na sljedeći način:
Neka je $n \in \mathbf{N}$ proizvoljan. Pogledajmo skup

$$\Sigma' = \bigcup_{0 \leq k \leq n} \Sigma_0^k.$$

Uzmimo da je dan neki uređaj na Σ. Njime je induciran leksikografski uređaj na Σ'.
Redom (leksikografski), za svaki $x \in \Sigma'$, izvršavamo prvih n koraka rada stroja T s ulazom x. Ako unutar tih n koraka T stane i prihvati x, tada definiramo $f(n) := x$, te ne uzimamo u obzir više x prilikom konstruiranja funkcije f za $j \in \mathbf{N}, j > n$. Inače, ako ni za jedan $x \in \Sigma'$ stroj T ne stane unutar n koraka, definiramo
$f(n) := w_0$.
Jasno je da ćemo na ovaj način pokupiti jedino elemente jezika α, jer je svaki element prepoznatljiv na T u konačno mnogo koraka.
Uočimo da smo sada skoro gotovi. Definiramo funkciju $g\colon \Sigma_0^* \to \mathbf{N}$, koja svakoj riječi pridružuje njenu duljinu (g je očito rekurzivna, stroj samo prolazi kroz danu riječ na ulazu i broji znakove).
Funkcija $f \circ g\colon \Sigma_0^* \to \Sigma_0^*$ je rekurzivna funkcija s traženim svojstvima. ■

Propozicija 1.3 *Svaki rekurzivan jezik je rekurzivno prebrojiv.*

Dokaz:
Po prethodnoj propoziciji treba pokazati da je takav jezik Turing prepoznatljiv. Kako je rekurzivan, postoji DTS koji ga odlučuje. No, tada taj DTS i prepoznaje jezik, pa je on i Turing prepoznatljiv. ■

Propozicija 1.4 *Jezik $\alpha \subseteq \Sigma_0^*$ je rekurzivan ako i samo ako su i α i $\Sigma_0^* \backslash \alpha$ rekurzivno prebrojivi.*

Dokaz:
Ako je α rekurzivan, tada je i njegov komplement rekurzivan, a prema prethodnoj propoziciji tada je i rekurzivno prebrojiv.
Obratno, ako su oba jezika rekurzivno prebrojiva, tada su i Turing prepoznatljiva. No sada konstruiramo Turingov stroj koji će za dani ulaz, izvoditi naizmjenice par koraka rada DTS koji prepoznaje α, a zatim par koraka rada DTS koji prepoznaje njegov komplement. Prije ili kasnije jedan od njih će prepoznati ulaz i tada smo gotovi. ■

Sada ćemo pokazati da postoje rekurzivno prebrojivi jezici koji nisu rekurzivni.
Neka je $T = (k, \Sigma, \Gamma, \delta)$ Turingov stroj. Sa α_T označimo jezik sastavljen od svih riječi $w \in \Sigma_0^*$ za koje T staje uz k-torku $(w, \ldots, w)$ kao ulaz (riječ w zapisana je na svim trakama kao ulaz).

Teorem 1.5 *Neka je $T = (k+1, \Sigma, \Gamma, \delta)$ univerzalni Turingov stroj. Tada je α_T rekurzivno prebrojiv, ali nije rekurzivan jezik.*

Dokaz:
Prema lemi 1.2 dovoljno je pokazati da je α_T Turing prepoznatljiv.
Promotrimo DTS $S = (k+1, \Sigma, \Gamma_S, \delta_S)$ koji za ulaz $w \in \Sigma_0^*$, radi na sljedeći način:

- kopira riječ sa prve trake na sve ostale trake
- simulira rad stroja T (sa $k+1$-torkom $(w, \ldots, w)$ kao ulazom)
- ako stroj T stane, tada prihvati ulaz.

Kako stroj T (uz $k+1$-torku $(w, \ldots, w)$ kao ulaz) staje ako i samo ako je $w \in \alpha_T$, slijedi da S prepoznaje α_T.

Ostaje pokazati da α_T nije rekurzivan (radi jednostavnosti dokazujemo slučaj samo za $k = 1$).
Pretpostavimo suprotno, odnosno da je α_T rekurzivan. Tada je rekurzivan i njegov komplement, tj. jezik $\alpha_T^c = \Sigma_0^* \backslash \alpha_T$. Kako je α_T^c rekurzivan, slijedi da postoji DTS $N = (1, \Sigma, \Gamma_N, \delta_N)$ koji ga odlučuje.
(U biti postoji DTS $N' = (s, \Sigma, \Gamma_{N'}, \delta_{N'})$ koji odlučuje α_T^c, ali njega po teoremu 1.1 zamijenimo jednotračnim strojem.)

Prepravimo stroj N tako da za riječi koje ne prihvaća radi beskonačno (ako ne prihvati riječ, zavrtimo ga u nekoj beskonačnoj petlji, na primjer da stalno ispisuje i briše jedan te isti znak). Ovako prepravljeni stroj N sada samo prepoznaje jezik α_T^c. Također, na ovaj smo način postigli da stroj N sa ulazom $w \in \Sigma_0^*$ staje ako i samo ako $w \notin \alpha_T$.

Kako je stroj T univerzalan, slijedi da postoji program $p \in \Sigma_0^*$ sa kojim T simulira stroj N. Pogledajmo što se događa kada stroj T pokrenemo sa p na obje trake. Kako T simulira N, stroj T će stati ako i samo ako N stane sa ulazom p. Sada jednostavno dolazimo do kontradikcije. Stroj T staje ako i samo ako $p \in \alpha_T$, a stroj N staje ako i samo ako $p \notin \alpha_T$. Dakle, ne mogu stati istovremeno. Kontradikcija. ■

Prethodni teorem jedan je od mnogo sličnih primjera jezika koji nisu rekurzivni (tj. odlučivi). Među njih spada i poznati halting problem (problem zaustavljanja) za Turingov stroj.
Neka je $T = (k, \Sigma, \Gamma, \delta)$ Turingov stroj. Sa T_{stop} označimo jezik svih riječi $w \in \Sigma_0^*$ za koje se T (sa ulazom w) zaustavlja. Halting problem je problem određivanja pripada li dana riječ jeziku T_{stop}. Vrijedi sljedeći teorem.

Teorem 1.6 *Postoji* 1*-tračni Turingov stroj čiji halting problem nije odlučiv.*

Neka je $T = (2, \Sigma, \Gamma, \delta)$ univerzalni Turingov stroj. Prema teoremu 1.1 postoji ekvivalentan jednotračni Turingov stroj $S = (1, \Sigma, \Gamma_S, \delta_S)$.
Nas će u ovom dokazu interesirati samo ulazni podaci oblika $(w, w), w \in \Sigma_0^*$ za stroj T. Zato, preradimo stroj S tako da on umjesto da prima uređeni par (w, w) kao ulaz, prima samo riječ w, ali nakon toga kopira tu riječ još jednom te dalje radi kao da je primio ulaz (w, w).

Neka je $w \in \Sigma_0^*$ proizvoljna riječ. Kako prema prethodnom teoremu nije odlučivo hoće li T sa ulazom (w, w) stati, tada nije odlučivo ni hoće li S sa ulazom w stati (prema teoremu 1.1 T se zaustavlja ako i samo ako se S zaustavlja).
S je traženi stroj. ■

Da ne bi ponavljali cijelu priču sada i za nedeterministički Turingov stroj, dokazati ćemo sljedeći teorem.

Teorem 1.7 *Jezik je Turing prepoznatljiv ako i samo ako ga prepoznaje neki NTS.*

Dokaz:
Neka je jezik Turing prepoznatljiv. Kako je svaki DTS ujedno i NTS, tada je taj DTS ujedno i NTS koji ga prepoznaje.
Obratno, tvrdnja slijedi iz teorema 1.2. ■

Korolar 1.1 *Jezik je rekurzivno prebrojiv ako i samo ako ga prepoznaje neki NTS.*

Dokaz:
Slijedi iz prethodnog teorema i iz propozicije 1.2. ■

Definicija 1.20 *Kažemo da NTS T,* ***odlučuje jezik*** *α, ako za svaki ulaz $x \in \Sigma_0^*$, svaka grana od T stane u konačno mnogo koraka s izlazom* 0 *ili* 1*.*

Napomena 1.13 *Ako sve grane imaju kao izlaz* 0*, tada riječ nije u jeziku, a ako bar jedna grana ima izlaz* 1*, tada riječ je u jeziku.*

Sada, jednostavno dobijemo:

Korolar 1.2 *Jezik je odlučiv ako i samo ako postoji NTS koji ga odlučuje.*

Korolar 1.3 *Jezik je rekurzivan ako i samo ako postoji NTS koji ga odlučuje.*

Dokaz:
Slijedi iz prethodnog korolara i iz propozicije 1.1. ■

1.5 Grafovi

U ovom poglavlju iznijet ćemo neke osnovne činjenice i rezultate o grafovima[18]. Grafovi će nam se često pojavljivati u raznim primjerima, pa čak i u dokazima nekih teorema.

Definicija 1.21 ***Graf*** *je uređena trojka $G = (V_G, E_G, \varphi)$ gdje je:*

- *V_G proizvoljan skup koji ćemo zvati skup vrhova (njegove elemente zvat ćemo vrhovi)*
- *E_G proizvoljan skup, disjunktan sa V_G, koji ćemo zvati skup bridova (njegove elemente zvat ćemo bridovi)*
- *φ funkcija koja svakom bridu e, pridružuje 2-člani multiskup vrhova $\{u, v\}$ ($\varphi(e) = \{u, v\}$) koji se zovu krajevi od e.*

Definicija 1.22 *Neka su G i H grafovi.*

- *Ako je $V_H \subseteq V_G$ i $E_H \subseteq E_G$, a svaki brid iz H ima iste krajeve u H kao što ih ima u G, onda kažemo da je H **podgraf** od G.*
- *Podgraf $H \subseteq G$ za koji vrijedi $V_H = V_G$ zovemo **razapinjujući podgraf** od G.*

Definicija 1.23 ***Put*** *u grafu G, je niz $P := v_0e_1v_1e_2 \dots e_kv_k$, gdje su:*

- *$v_i, 0 \leq i \leq k$, međusobno različiti vrhovi od G*
- *$e_i, 1 \leq i \leq k$, međusobno različiti bridovi od G takvi da su krajevi od e_i vrhovi v_{i-1} i v_i.*

*Kažemo da je vrh v_0 **početak**, a vrh v_k **kraj** puta P. Također kažemo da je P put od vrha v_0 do vrha v_k ili (v_0, v_k)-put. Broj k zovemo **duljina puta** P.*

[18] Detalje možete naći u [6].

Definicija 1.24 *Neka je G graf.* ***Udaljenost****, $d_G(u, v)$, dvaju vrhova u i v u grafu G je duljina najkraćeg puta među njima. Ako takvog puta nema, stavljamo $d_G(u, v) = \infty$*

Definicija 1.25 *Kažemo da je graf G* ***povezan****, ako su svaka dva njegova vrha povezana s nekim putom.*

Slično putu u grafu, imamo i:

Definicija 1.26 ***Ciklus*** *u grafu G, je niz $P := v_0e_1v_1e_2 \dots e_kv_k$, gdje su:*

- *$v_i, 0 \leq i \leq k-1$, međusobno različiti vrhovi od G, te $v_0 = v_k$*
- *$e_i, 1 \leq i \leq k$, međusobno različiti bridovi od G takvi da su krajevi od e_i vrhovi v_{i-1} i v_i.*

Broj k zovemo ***duljina ciklusa*** *P.*

Definicija 1.27 *Povezan graf u kojem nema ciklusa zove se* ***stablo****.*
Razapinjujuće stablo *grafa G je razapinjujući podgraf koji je stablo.*

Vrijedi sljedeći teorem[19].

Teorem 1.8 *Svaki povezan graf ima razapinjujuće stablo.*

Definirajmo sada jednu specijalnu vrstu grafa.

Definicija 1.28 ***Težinski graf*** *je graf u kojem je svakom bridu pridružen nenegativan broj w. Broj w zovemo težina brida.*

Ako je težinski graf povezan, tada prema teoremu 1.8 ima razapinjuće stablo. Ono razapinjuće stablo, za koje je zbroj težina na bridovima minimalan, zvati ćemo minimalno razapinjuće stablo.

Kratki podsjetnik o grafovima završavamo s još jednom vrstom grafa.

Definicija 1.29 ***Usmjereni graf*** *D je uređena trojka $D = (V, A, \varphi)$, gdje je:*

- *V proizvoljan skup kojeg ćemo zvati skup vrhova (njegove elemente zvat ćemo vrhovi)*
- *A proizvoljan skup, disjunktan sa V, koje zovemo skup lukova (njegove elemente zvat ćemo lukovi)*

[19] Za dokaz vidi [6, str. 263].

- $\varphi: A \to V \times V$, *funkcija koja svakom luku* a, *pridružuje uređeni par vrhova* (u, v), *koje zovemo početak i kraj od* a.

Primijetimo da je usmjereni graf u biti graf u kojem svaki brid ima smjer (orijentaciju) od početka prema kraju.

Napomena 1.14 *Za usmjereni graf definiramo slične pojmove kao i za obični graf (put, ciklus, stablo,...) samo što uvijek pazimo na orijentaciju*[20].

Ovime završavamo uvodno poglavlje i prelazimo na ključni dio ovog rada, tj. na uvođenje klasa složenosti.

[20] Vidi detalje u [6, str.275].

2 DTIME i DSPACE

2.1 Uvod

Odabirom Turingovog stroja kao računalnog modela, postavili smo temelje za uvođenje pojma klase složenosti. Vidjet ćemo da će postojati vrlo značajna razlika prilikom korištenja deterministočkog, odnosno nedeterminističkog Turingovog stroja, što nam donekle daje naslutiti i teorem 1.2. Osim već odabranog računalnog modela, potrebno je još specificirati resurse i funkciju složenosti.

Daleko najvažniji resursi su vrijeme i prostor, pa tako i govorimo o vremenskoj i prostornoj složenosti. Pokazat će se da je prostorna složenost uvijek bitno manja od vremenske i samim time bitno manje ograničavajuća.
Funkcije složenosti imaju ulogu granica. Slično kao i kod resursa, postoji velik broj funkcija složenosti, ali kao najvažnije pokazali su se polinomi i eksponencijalne funkcije.

Sve spomenuto vodi nas do definicija raznih klasa složenosti, odnosno grubo govoreći, do klasa problema rješivih unutar danih vremenskih odnosno prostornih granica.

Dalje kroz poglavlje obradit ćemo neke poznate primjere koji će pripadati novo uvedenim klasama složenosti. Također, dokazat ćemo osnovne rezultate o odnosima klasa složenosti, kao i poznate Speed-up teoreme koji govore o mogućnosti ubrzanja računanja na Turingovom stroju za određene probleme.

2.2 Osnovne klase složenosti

Prije nego što definiramo neke od najvažnijih klasa složenosti, uvedimo par ključnih pojmova. Opet, napomenimo da se na sličan način sljedeći pojmovi definiraju i za ostale računalne modele, koji su po snazi ekvivalentni Turingovom stroju.

Definicija 2.1 *Neka je T Turingov stroj (deterministički ili nedeterministički).*

a) *Kažemo da je T* ***vremenski polinoman****, ili samo* ***polinoman****, ako postoji polinom f takav da je $time_T = O(f)$.*
Kažemo da je T ***vremenski eksponencijalan*** *(****eksponencijalan****), ako postoji eksponencijalna funkcija g takva da je $time_T = O(g)$.*

b) *Kažemo da je T* ***prostorno polinoman*** *ako postoji polinom f takav da je $space_T = O(f)$.*
Kažemo da je T ***prostorno eksponencijalan*** *ako postoji eksponencijalna funkcija g takva da je $space_T = O(g)$.*

Definicija 2.2 *Neka je Σ konačna abeceda koja sadrži prazan simbol $*$. Neka je $\Sigma_0 = \Sigma \backslash \{*\}$ te $\alpha \subseteq \Sigma_0^*$ proizvoljan jezik i neka su $f: \mathbf{N} \to \mathbf{N}, g: \mathbf{N} \to \mathbf{N}$ neke funkcije.*

a) *Kažemo da jezik α ima* ***vremensku složenost*** *najviše f, ako postoji Turingov stroj T koji ga odlučuje i vrijedi $time_T = O(f)$.*

b) *Kažemo da jezik α ima* ***prostornu složenost*** *najviše g, ako postoji Turingov stroj T koji ga odlučuje i vrijedi $space_T = O(g)$.*

Sada možemo definirati neke od osnovnih klasa složenosti.

Definicija 2.3 *Neka je $f: \mathbf{N} \to \mathbf{R}$, proizvoljna funkcija.*

- *Sa* **DTIME**(f) *označavamo klasu svih jezika odlučivih na nekom DTS T, za kojeg vrijedi $time_T = O(f)$.*
- *Sa* **DSPACE**(f) *označavamo klasu svih jezika odlučivih na nekom DTS T, za kojeg vrijedi $space_T = O(f)$.*

Sljedeća klasa je jedna od najznačajnijih u teoriji složenosti.

Definicija 2.4 PTIME *ili* **P** *je klasa svih jezika odlučivih s polinomnim, deterministič̆kim Turingovim strojem.*
Drugim riječima, $\mathrm{P} = \bigcup_{k \in \mathbf{N}} \mathrm{DTIME}(n^k)$.

Definicija 2.5 PSPACE *je klasa svih jezika odlučivih s deterministič̆kim, prostorno polinomnim Turingovim strojem, odnosno:*
$\mathrm{PSPACE} = \bigcup_{k \in \mathbf{N}} \mathrm{DSPACE}(n^k)$.

Napomena 2.1 *Često ćemo govoriti o složenosti nekog postupka ili algoritma (na primjer množenje dvaju brojeva) odnosno o pripadnosti tog postupka određenoj klasi (klase složenosti definirali smo kao klase jezika). Tada zapravo govorimo o gornjoj ogradi na složenost funkcije (funkcija je množenje cijelih brojeva), a ne o složenosti jezika. Naime, klase složenosti definirali smo kao klase jezika, ali slično se mogu definirati i odgovarajuće klase funkcija.*

Važnost klase P krije se u tome što velik broj poznatih i jako korištenih problema spada u nju. Također, klasa P više-manje odgovara klasi problema koji su rješivi u razumnom vremenu. Naravno, pitanje je smatramo li problem složenosti, na primjer, $n^k, k \geq 100$ rješivim u razumnom vremenu. Ipak, uzimanje polinomne vremenske složenosti kao svojevrsne granice pokazalo se korisnim.
Nadalje, važnost klase P leži i u tome što je ona invarijanta za sve računalne modele ekvivalentne polinomnom Turingovom stroju.

S druge strane, PSPACE je puno šira klasa problema i pokazat će se da je klasa P njen podskup, a vidjeti ćemo i puno zanimljivije rezultate. Navedimo sada neke od poznatih problema za koje ćemo dokazati da pripadaju klasi P.

2.3 Primjeri problema u PTIME

Krenimo od najjednostavnijih i najelementarnijih primjera, kao što su elementarne računske operacije: zbrajanje, oduzimanje, množenje i dijeljenje. Ako su brojevi zapisani u binarnom obliku, te ako je duljina zapisa n, uz općepoznate algoritme, zbrajanje i oduzimanje zahtijeva $O(n)$ vremena, dok su množenje i dijeljenje nešto složeniji i zahtijevaju $O(n^2)$ vremena.

Uzmimo sada ipak nešto složeniji primjer, Euklidov algoritam za računanje najvećeg zajedničkog djeljitelja dvaju brojeva x i y (oznaka: $nzm(x, y)$).

Euklidov algoritam:

- Neka su dana dva prirodna broja a i b. Bez smanjenja općenitosti neka je $a \leq b$.
- Ako je $a = 0$ tada je $nzm(a, b) = b$.
- U suprotnome, ako vrijedi $a > 0$, tada prema teoremu o dijeljenju sa ostatkom slijedi da postoje prirodni brojevi $q \geq 1$ i $0 \leq r < a$ takvi da vrijedi $b = q \cdot a + r$. Tada je $nzm(a, b) = nzm(r, a)$.

Kako je $r < a$, očito je da dana rekurzija završava u konačno mnogo koraka. Točnije, vrijedi sljedeća lema:

Propozicija 2.1 *Euklidov algoritam je vremenski polinoman. Točnije, zahtijeva* $O(\log_2 a + \log_2 b)$ *aritmetičkih operacija.*

Dokaz:
Kao što smo već vidjeli, zbog $r < a$ algoritam će završiti u konačno mnogo

koraka. Dokažimo da završava u polinomnom vremenu.
Promotrimo rad Euklidovog algoritma:

$$b = q_0 a + r_0$$

$$a = q_1 r_0 + r_1$$

$$r_0 = q_2 r_1 + r_2$$

$$\vdots$$

U prvom koraku vrijedi:

$$b \geq a + r_0 > 2r_0 \text{ iz čega slijedi } r_0 < \frac{b}{2} \text{ i dalje } ar_0 < \frac{ab}{2}.$$

U drugom na isti način dobivamo:

$$r_0 r_1 < \frac{r_0 a}{2}.$$

Za treći i svaki sljedeći korak vrijedi:

$$r_i r_{i+1} < \frac{r_i r_{i-1}}{2}.$$

Kombinirajući prva dva rezultata dobivamo:

$$r_0 r_1 < \frac{ab}{4}.$$

Analogno dalje slijedi:

$$r_1 r_2 < \frac{ab}{8}.$$

Sada vidimo da će u svakom sljedećem koraku umnožak dvaju uzastopnih ostataka biti manji za faktor 2, iz čega imamo da za $k = \lceil \log_2 (ab) \rceil$ ($\lceil x \rceil$ je najmanje cijelo od x), sigurno vrijedi:

$$r_{k-2} r_{k-1} < \frac{ab}{2^k} = \frac{ab}{2^{\lceil \log_2 (ab) \rceil}} \leq \frac{ab}{2^{\log_2 (ab)}} = \frac{ab}{ab} = 1$$

Slijedi da će nakon najviše $\lceil \log_2 (ab) \rceil$ koraka produkt odgovarajućih r_i, r_{i+1} biti strogo manji od 1. No, to znači da je jedan od njih jednak 0 čime algoritam završava. Kako se svaki korak sastoji od elementarnih računskih operacija koje su polinomne, slijedi da je ukupno vrijeme algoritma polinomno. ■

Promotrimo sada nešto drukčiju varijantu Euklidovog algoritma:

- Ako je $a = 0$ tada je $nzm(a, b) = b$.
- Ako je $a > b$ tada zamijenimo brojeve.
- Inače, za $0 < a \leq b$, definiramo $b := b - a$.

Iako oba algoritma korektno daju najveću zajedničku mjeru u konačno mnogo koraka, postoji velika razlika u njihovoj vremenskoj složenosti. Naime, prva varijanta je polinomna, a druga čak eksponencijalna.

Za računanje $nzm(1, b)$ potrebno je b koraka, što u terminima binarnog zapisa broja b (koji ima $\lceil \log b \rceil + 1$ znamenaka) daje eksponencijalnu vremensku složenost.

Dakle, iako promjene u konačnom izlazu algoritma nema, vidimo da promjene u strukturi algoritma mogu dovesti do drastičnih razlika u njihovoj vremenskoj složenosti.
Konkretno, ovdje razlog "sporosti" drugog algoritma leži u sljedećem:

$b = qa + r$ (prvi algoritam-jedan korak)

$b = b\underbrace{-a - a - \ldots - a}_{q} + r$ (drugi algoritam-q koraka).

Očito drugi algoritam radi $q - 1$ korak više za svaki korak prvog algoritma.

Uzmimo sada primjer polinomnog algoritma iz teorije grafova. Napomenimo da se ovdje kao veličina ulaza smatra veličina odabranog prikaza grafa. Međutim, razumno je pretpostaviti da veličina razumnog prikaza ovisi polinomno o broju čvorova grafa. Dakle, dovoljno je pokazati polinomnost grafa u odnosu na broj čvorova.

Kao primjer ćemo uzeti Primov algoritam za nalaženje minimalnog razapinjućeg stabla u povezanom težinskom grafu. Primov algoritam primjer je pohlepnog algoritma.

Primov algoritam:

- Neka je dan težinski graf $G = (V, E)$ sa skupom vrhova V te skupom bridova s težinama E. Neka je $|V| = n$.
- Odaberi proizvoljan vrh iz $v \in V$ i definiraj stablo $T = \{v\}$.
- Ponavljaj sljedeći korak sve dok T ima manje od $n - 1$ brid:
  Dodaj stablu T brid minimalne težine s jednim krajem u T, a drugim u $V \backslash T$.

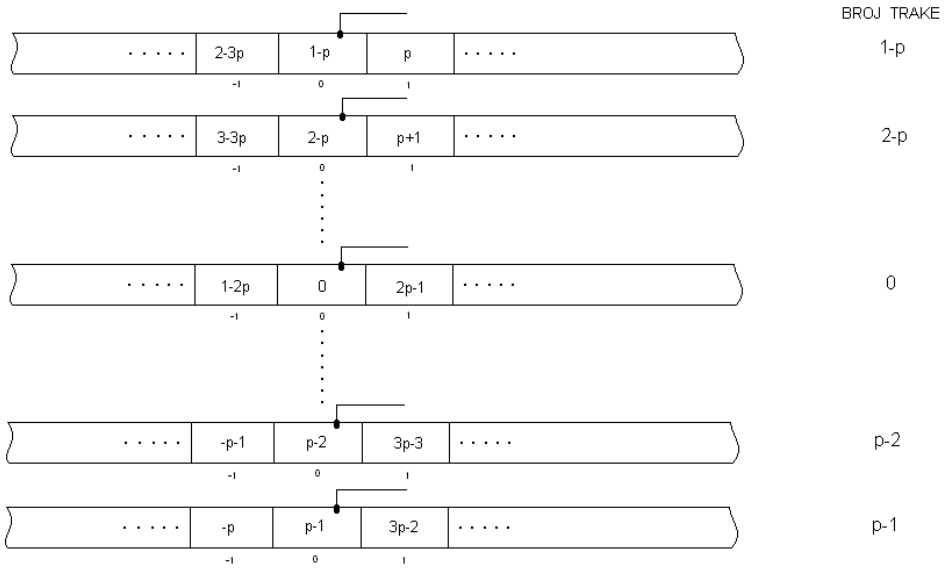
Propozicija 2.2 *Primov algoritam je vremenski polinoman reda složenosti* $O(n^2)$.

Dokaz:
Dokažimo prvo da je algoritmu potrebno $O(n^2)$ koraka. U tu svrhu definirajmo dva pomoćna polja: kandidat i težina.
Za vrh $v \in V \backslash T$ neka je kandidat$[v]$ jednak vrhu iz T koji mu je najbliži, a težina$[v]$ neka je težina brida između v i kandidat$[v]$. Primijetimo da u početnom trenutku, kada je $T = \{v\}$, polje kandidat na svim mjestima može sadržavati samo v.
Očito je da algoritam točno $n - 1$ put dodaje novi brid stablu T. Zato je dovoljno pokazati da je odabir brida s minimalnom težinom izvediv u $O(n)$ koraka.
No to je sada očito, jer svaki put samo prolazimo poljem kandidat, koje je duljine najviše $n-1$ i u svakom koraku se smanjuje za 1. Još ostaje pokazati da popravak polja kandidat također traži $O(n)$ koraka. Recimo da je odabran vrh $s \in V \backslash T$ za ubacivanje u stablo T. Popravak ide ovako: gledamo je li $d(v, s) < d(v, kandidat[v])$ za svaki $v \in V \backslash T$ te ako je to ispunjeno, postavimo kandidat$[v] = s$, te težina$[v] = d(v, s)$.
Dakle, Primov algoritam je reda složenosti $O(n^2)$. Kako u svakom koraku obavljamo samo elementarne operacije uspoređivanja i zamjene brojeva koje su vremenski polinomne, slijedi da je cijeli algoritam vremenski polinoman. ■

Postoji još velik broj poznatih problema koji pripadaju klasi P. Međutim, možda zanimljivije pitanje od pripadnosti nekog problema pojedinoj klasi jest pitanje međusobnog odnosa samih klasa složenosti.
Sljedeći odjeljak dati će nam bolji uvid u odnose među raznim klasama složenosti.

2.4 Odnosi među klasama složenosti

Pitanje brzine, odnosno mogućnosti ubrzanja bilo algoritma bilo računala, temeljno je pitanje u modernom računarstvu. Naravno, promatrano kroz aspekt ovog rada, ekvivalentno je pitanje mogućnosti ubrzanja rada Turingovog stroja. Uzmemo li u obzir teorem 1.1, razumno je zapitati se možemo li dodavanjem novih traka, kompliciranjem Turingovog stroja, smanjiti vremensku složenost. Pokazuje se da je određen pomak stvarno moguć, što dokazuje sljedeći teorem.



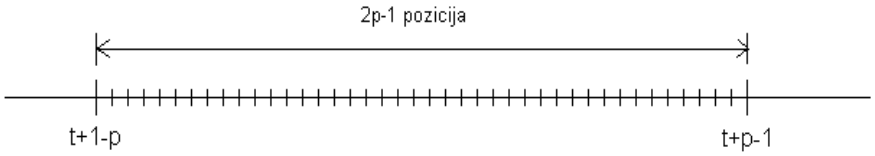
Slika 1: Način indeksiranja ćelija stroja S na početnim i radnim trakama.

Teorem 2.1 (Linear Speed-up Teorem) *Neka je α proizvoljan rekurzivan jezik. Za svaki Turingov stroj T koji odlučuje α, i $c \in \mathbf{R}, c > 0$, postoji Turingov stroj S nad istom abecedom koji odlučuje α te za kojega vrijedi:*

$$time_S(n) \leq c \cdot time_T(n) + n, \forall n \in \mathbf{N}.$$

Dokaz:
Uzmimo prvo da T ima samo jednu radnu traku. Također možemo pretpostaviti da je c oblika $\frac{1}{p}, p \in \mathbf{N}$ (možemo uzeti p takav da je $\frac{1}{p} \leq c$, pa tvrdnju dokažemo za $\frac{1}{p}$, iz čega slijedi i tvrdnja za c). Krenimo sada na definiranje stroja S.

Neka Turingov stroj S ima jednu ulaznu traku, jednu izlaznu traku, $2p-1$ traka koje ćemo zvati početne trake, te još $2p-1$ radnih traka. Označimo i početne i radne trake redom brojevima od $1-p$ do $p-1$. Slično, reindeksirajmo ćelije na početnim, te radnim trakama od S ovako: j-tu ćeliju na i-toj traci označimo sa $j(2p-1)+i$.
Primijetimo da su skupovi indeksa za bilo koje dvije radne (početne) trake disjunktni, te da unijom svih indeksa ćelija radnih (početnih) traka dobivamo sve moguće indekse ćelija, odnosno skup $\mathbf{Z}$. Ovakvim smo načinom indeksiranja postigli da i-ta ćelija ulazne (radne) trake od T odgovara i-toj ćeliji, na točno jednoj, početnoj (radnoj) traci od S.



Slika 2: Interval na kojem su pozicionirane glave na početnim trakama TS S, nakon $p \cdot k$ koraka rada TS T (analogno za radne trake).

Definirajmo induktivno rad stroja S.
Neka S prvo kopira svaki znak ulaznog podatka s ulazne trake u ćeliju s odgovarajućim indeksom na nekoj od početnih traka, te neka vrati sve glave na ćeliju 0 (po starom načinu indeksiranja). Primijetimo da glave stroja S, u ovom trenutku, stoje na ćelijama s indeksima od $1 - p$ do $p - 1$ (položaj glava prikazan je na slici 1).
Cilj nam je jednim korakom rada od S simulirati p uzastopnih koraka rada od T, čime ćemo dobiti tvrdnju teorema. Zato, promatrajmo rad stroja T u segmentima od po p koraka.
Neka se nakon pk koraka, $k \in \mathbf{N}$, glava za pisanje na ulaznoj, odnosno na radnoj traci od T nalaze na t-toj, odnosno s-toj ćeliji. Neka ćelije na početnim (radnim) trakama od S sadržavaju iste znakove kao i ćelije s istim indeksima na ulaznoj (radnoj) traci od T. Također, neka su glave početnih traka stroja S pozicionirane na ćelijama s indeksima

$$t + 1 - p, \ldots, t + p - 1,$$

a glave radnih traka stroja S na ćelijama s indeksima

$$s + 1 - p, \ldots, s + p - 1.$$

Pogledajmo kakva će biti konfiguracija stroja S nakon još p koraka Turingovog stroja T.
Kako S svojim glavama na početnim trakama pokriva ćelije s indeksima

$$t + 1 - p, \ldots, t + p - 1, \tag{1}$$

a glavama na radnim trakama ćelije s indeksima od

$$s + 1 - p, \ldots, s + p - 1, \tag{2}$$

stroj S točno zna što T može pročitati na ulaznoj, odnosno radnoj traci u još najviše $p-1$ koraka. Zato, S može predvidjeti kretanje glava od T kao i eventualne promjene u sadržaju traka od T, u sljedećih p koraka.
Pretpostavimo da su nakon još p koraka, glave od T pozicionirane na ćelijama s indeksima $t+i, s+j$, za $0 < i, j \leq p-1$.
(Uzeli smo da su se glave od T više puta pomaknule u desno nego u lijevo pa je zato $i, j > 0$, ali ostali slučajevi su analogni. Također, kako se u svakom koraku glava Turingovog stroja može pomaknuti najviše za jedno mjesto, slijedi da je $i, j \leq p-1$.)
Isto tako, glava radne trake od T mogla je promijeniti samo ćelije u intervalu

$$s+1-p, \ldots, s+p-1.$$

Stroj S mora odsimulirati p uzastopnih koraka od T (i to u jednom koraku). To radi na sljedeći način.
Pročita znakove sa svih traka, i pomoću svoje funkcije prijelaza odredi:

- koje će biti unutarnje stanje od T nakon p koraka, i zapamti ga
- koje znakove treba zapisati na trake da bi stanje na njima odgovaralo stanju traka od T, i zapiše ih (primijetimo da to možemo izvesti u jednom koraku jer imamo glave na ćelijama u intervalima (1) i (2))
- pomak glava (objašnjeno ispod).

Objasnimo kako S pomiče glave (vidi sliku 3 na sljedećoj strani).
Treba pomaknuti glave na početnim (radnim) trakama tako da pokrivaju interval

$$t+i+1-p, \ldots, t+i+p-1 \ (s+j+1-p, \ldots, s+j+p-1).$$

Kako već imamo glave na početnim (radnim) trakama koje pokrivaju ćelije na pozicijama

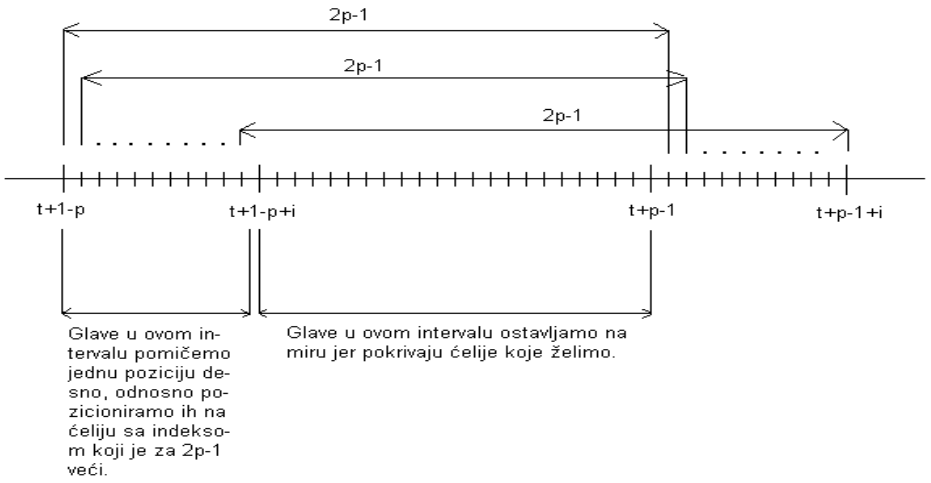
$$t+1-p+i, \ldots, t+p-1 \ (s+1-p+j, \ldots, s+p-1)$$

dovoljno je pomaknuti glave koje pokazuju na ćelije u intervalu

$$t+1-p, \ldots, t-p+i \ (s+1-p, \ldots, s-p+j)$$

jedno mjesto u desno (Po konstrukciji susjedne ćelije na istoj traci su udaljene za $2p-1$.). Time te glave pokazuju na ćelije u intervalu

$$t+p, \ldots, t+i+p-1 (s+p, \ldots, s+j+p-1)$$



Slika 3: Pomicanje glava TS S.

čime dobivamo željeni interval. U slučaju da T završi s radom unutar razmatranih p koraka neka i S završi s radom.

Ovako konstruirani Turingov stroj S radi p-puta manje koraka nego T, a odlučuje jezik α. Međutim, kako čitanje ulaznog podatka nije moguće ubrzati, slijedi tvrdnja teorema. Time je dokazan slučaj kada T ima samo jednu radnu traku. Slučaj kada T ima $l > 1$ radnih traka slijedi odmah ako uzmemo $l \cdot (2p - 1)$ radnih traka za stroj S (za svaku radnu traku uzmemo $2p - 1$ radnih traka). ■

Iako teorem 2.1 govori da je Turingov stroj moguće ubrzati, primijetimo da se efektivno ubrzanje postiže samo ako je dovoljno vremena potrošeno na računanje, odnosno ako vrijedi $time_T(n) > n$. Također, iz teorema 2.1 vidimo da vremenska složenost ne ovisi o konstantnom faktoru.

Analogon teorema 2.1 vrijedi i za prostorno ubrzanje.

Teorem 2.2 *Neka je α proizvoljan odlučiv jezik.*
Za svaki Turingov stroj T koji odlučuje α i $c \in \mathbf{R}, c > 0$, postoji Turingov stroj S koji odlučuje α, te za kojega vrijedi:

$$space_S(n) \leq c \cdot space_T(n) + n, \forall n \in \mathbf{N}.$$

Ideja dokaza:
Ideja dokaza svodi se na dodavanje novih znakova abecedi. Nove znakove

koristit ćemo za kodiranje riječi. Princip je sličan kao i kod zapisa nekog broja u određenoj bazi. Što je veći kardinalni broj baze, to će zapis biti kraći.
Ako uzmemo dovoljno velike blokove (blokove ćelija, ali u biti kodiramo riječi koje oni sadrže) i njih kodiramo jednim znakom, broj korištenih ćelija se smanjuje upravo za faktor veličine bloka. Naravno, potrebno je i konstruirati DTS koji bi simulirao rad stroja T i koji radi s takvim blokovima. □

Sada možemo izvesti prve, najjednostavnije rezultate o odnosima klasa složenosti.

Propozicija 2.3 *Neka je* $f: \mathbf{N} \to \mathbf{R}$*, proizvoljna funkcija za koju vrijedi* $f(n) > n$*. Tada vrijedi*

$$\mathrm{DTIME}(f) \subseteq \mathrm{DSPACE}(f)$$

Dokaz:
Neka je $\alpha \in \mathrm{DTIME}(f)$ proizvoljan jezik.
Tada postoji neki k-tračni DTS T, vremenske složenosti f, koji odlučuje α.
Kako u jednom koraku T može promijentiti najviše k ćelija (na svakoj od traka po jednu), slijedi da je njegova prostorna složenost najviše k puta veća od vremenske.
Kako stroj T odlučuje α, tada vrijedi $\alpha \in \mathrm{DSPACE}(k \cdot f)$.
Ako pokažemo, da za svaki $k \in \mathbf{N}$, vrijedi

$$\mathrm{DSPACE}(k \cdot f) = \mathrm{DSPACE}(f)$$

tvrdnja propozicije je dokazana.
Neka je k proizvoljan. Očito vrijedi

$$\mathrm{DSPACE}(f) \subseteq \mathrm{DSPACE}(k \cdot f).$$

Obratno, neka je $\beta \in \mathrm{DSPACE}(k \cdot f)$, proizvoljan jezik.
Tada postoji DTS M, prostorne složenosti $k \cdot f$, koji odlučuje β.
Prema teoremu 2.2 (za $c = \frac{1}{k}$), slijedi da postoji DTS S, vremenske složenosti najviše $c \cdot k \cdot f(n) + n = f(n) + n$ za svaki $n \in \mathbf{N}$, koji odlučuje β.
Kako je $f(n)+n < 2f(n) = O(f(n))$, te S odlučuje β (koji je bio proizvoljan) vrijedi

$$\mathrm{DSPACE}(k \cdot f) \subseteq \mathrm{DSPACE}(f).$$

Time je ova propozicija dokazana. ■

Sada jednostavno dobivamo jednu jako bitnu činjenicu.

Korolar 2.1 *Vrijedi*

$$\mathrm{P} \subseteq \mathrm{PSPACE}.$$

Dokaz:
Neka je $\alpha \in \mathrm{P}$, proizvoljan jezik.
Tada postoji $k \in \mathbf{N}$, takav da je $\alpha \in \mathrm{DTIME}(n^k)$.
Prema prethodnoj propoziciji imamo da vrijedi $\alpha \in \mathrm{DSPACE}(n^k)$.
Dakle, $\mathrm{P} \subseteq \mathrm{PSPACE}$. ■

Pogledajmo što možemo reći o vremenskoj složenosti nekog jezika za kojeg znamo prostornu složenost.

Propozicija 2.4 *Neka je $f: \mathbf{N} \to \mathbf{R}$ proizvoljna funkcija, takva da je $f(n) > \log n$. Tada vrijedi:*
Ako je $\alpha \in \mathrm{DSPACE}(f)$, tada postoji $m \in \mathbf{N}$, takav da je

$$\alpha \in \mathrm{DTIME}((m+1)^{f(n)})$$

Dokaz:
Prema pretpostavci propozicije, postoji DTS $T = (k, \Sigma, \Gamma, \delta)$, prostorne složenosti najviše f koji odlučuje α. Neka je $m = |\Sigma|$, te neka je $s = |\Gamma|$.
Prema napomeni 1.10 broj različitih konfiguracija za stroj T je

$$s \cdot (2f(n)+1)^k \cdot (m^{2f(n)+1})^k.$$

Međutim, izračunamo li malo pažljivije, te promatramo samo one ćelije koje efektivno mijenjamo vrijedi da je broj različitih konfiguracija ograničen sa

$$max = s \cdot f(n)^k \cdot m^{f(n)}.$$

Kako T odlučuje α (odnosno, za svaki ulaz sigurno staje), dvije se iste konfiguracije ne mogu ponoviti jer bi u tom slučaju T ušao u ciklus te ne bi nikad stao. Dakle, broj koraka koje T radi za proizvoljni ulaz sigurno je manji ili jednak max.
Međutim, prema teoremu 2.1 vremenska složenost jezika α ne ovisi o konstantnim faktorima pa možemo konstruirati DTS koji ga odlučuje u vremenu $O(f(n)^k m^{f(n)})$.
Po pretpostavci vrijedi da je $f(n) > \log n$, pa $m^{f(n)}$ raste brže nego $f(n)^k$. No tada je, $f(n)^k m^{f(n)} = O((m+1)^{f(n)})$.
Dakle,

$$\alpha \in \mathrm{DTIME}((m+1)^{f(n)})$$

■

Prije nego što nastavimo dalje uvedimo novi pojam.

Definicija 2.6 *Za funkciju $f: \mathbf{N} \to \mathbf{N}$ kažemo da je* ***potpuno vremenski konstruktibilna***[21]*, ako postoji Turingov stroj koji za svaki ulaz duljine n radi točno $f(n)$ koraka.*

Napomena 2.2 *Funkcija f iz prethodne definicije mora zadovoljavati $f(n) > n$, jer uzimamo da Turingov stroj mora barem pročitati ulaz i još jedan znak koji označava kraj ulaza.*

Jasno je i da je svaka potpuno vremenski konstruktibilna funkcija i rekurzivna. Obrat ne vrijedi jer možemo uzeti rekurzivnu funkciju f za koju je $f(n) \leq n$.
Također, osnovne su funkcije, kao $n^2, 2^n, n!, ..$ potpuno vremenski konstruktibilne.
Pokažimo, na primjer, da je funkcija $f(n) = n^2$ potpuno vremenski konstruktibilna.
Neka je dan 2-tračni Turingov stroj kojem je na ulazu riječ duljine n. Dajemo grubi opis rada stroja:

- Kopira riječ s ulaza na drugu traku, s time da u prvom koraku, na prvoj traci briše prvi znak ulazne riječi (ukupno n koraka, a na ulazu je ostala riječ duljine $n-1$).
- Za svaki od $n-1$ preostalih znakova na prvoj traci pročita cijelu riječ na drugoj traci ($(n-1)n$ koraka).

Sljedeća lema daje nam još mnogo potpuno vremenski konstruktibilnih funkcija.

Lema 2.1 *Za svaku rekurzivnu funkciju $f: \mathbf{N} \to \mathbf{N}$ postoji potpuno vremenski konstruktibilna funkcija g takva da vrijedi $f \leq g$.*

Dokaz:
Kako je f rekurzivna tada postoji DTS $T = (k, \Sigma, \Gamma_T, \delta_T)$ koji za svaki $n \in \mathbf{N}$, izračunava $f(n)$ u konačno mnogo koraka.
Označimo sa $step_T(n)$ broj koraka potrebnih stroju T da bi za ulaz n izračunao $f(n)$.

Neka je $M = (l, \Sigma, \Gamma_M, \delta_M)$ DTS koji za ulaz $w \in \Sigma_0^*$ računa $|w|$. Također, označimo sa $step_M(n)$ broj koraka koje M radi za riječi duljine n.

Nadalje, neka je $S = (d, \Sigma, \Gamma_S, \delta_S)$ DTS koji za ulaz $n \in \mathbf{N}$ kao izlaz daje niz od n istih znakova (odaberemo neki znak koji ne koristimo; nije bitno

[21] engleski: fully time-constructibile

za dokaz). Neka je $step_S(n)$ potreban broj koraka da bi S izveo taj posao (primijetimo da je $step_S(n) \geq n$ jer moramo zapisati ako ništa drugo n znakova).

Neka je funkcija $g: \mathbf{N} \to \mathbf{N}$ koju, za $n \in \mathbf{N}$, definiramo ovako:

$$g(n) = step_M(n) + step_T(n) + step_S(f(n)).$$

Po konstrukciji vrijedi $f(n) \leq g(n)$, jer je samo $step_S(f(n)) \geq f(n)$. Nadalje, spajanjem strojeva M, T i S (u tom redoslijedu) dobivamo Turingov stroj nad abecedom Σ, koji za svaku riječ iz Σ_0^*, duljine n, radi točno $g(n)$ koraka. ■

Pokažimo sada da nema klase koja bi mogla obuhvatiti sve jezike, odnosno da postoje jezici proizvoljno velike vremenske složenosti. U tome će nam pomoći rekurzivne funkcije.

Teorem 2.3 *Za svaku rekurzivnu funkciju f postoji rekurzivan jezik α takav da $\alpha \notin \mathrm{DTIME}(f)$.*

Dokaz:
Neka je $f: \mathbf{N} \to \mathbf{N}$ rekurzivna funkcija. Prema prethodnoj lemi postoji potpuno vremenski konstruktibilna funkcija g, takava da je $f \leq g$. Ako dokažemo tvrdnju teorema za g, tada će postojati rekurzivan jezik β takav da $\beta \notin \mathrm{DTIME}(g)$. Kako je $f \leq g$, vrijedi $\mathrm{DTIME}(f) \subseteq \mathrm{DTIME}(g)$. No sada je očito da je jezik β takav da $\beta \notin \mathrm{DTIME}(f)$.
Dakle, možemo uzeti da je f potpuno vremenski konstruktibilna funkcija.
Tada vrijedi $f(n) > n$ za svaki $n \in \mathbf{N}$. Neka je $T = (2, \Sigma, \Gamma, \delta)$, 2-tračni univerzalni Turingov stroj.
Neka se jezik α sastoji od svih riječi $x \in \Sigma_0^*$, takvih da za ulaz (x, x) stroj T staje u najviše $f(|x|)^4$ koraka. Jezik α je po konstrukciji rekurzivan.

Tvrdimo da $\alpha \notin \mathrm{DTIME}(f)$. Pretpostavimo suprotno, odnosno da vrijedi $\alpha \in \mathrm{DTIME}(f)$.
Tada postoji k-tračni Turingov stroj vremenske složenosti f koji odlučuje α. Prema teoremu 1.1 možemo konstruirati ekvivalentni 1-tračni Turingov stroj M čija je vremenska složenost najviše $cf(n)^2$.
Za dovoljno velike n, recimo $n > c$, vrijedi

$$cf(n)^2 < nf(n)^2 < f(n)f(n)^2 = f(n)^3.$$

Željeli bi konstruirati DTS koji odlučuje α i uvijek stane u vremenu $f(n)^3$. S dovoljno dugim riječima nemamo problema, te koristimo stroj M.
Trebamo još pokazati da riječi duljine $n \leq c$ možemo prepoznati u vremenu

$f(n)^3$ (primijetimo da ih ima konačno mnogo). No, konačno mnogo riječi uvijek možemo eksplicitno zapisati u neku vrstu tablice ili pratiti pomoću unutarnjih stanja stroja. U svakom slučaju, prepoznavanje takvih riječi možemo izvesti u linearnom vremenu, samo moramo zakomplicirati sam stroj M.
Pa, neka je N neki tako modificirani stroj koji uvijek staje u $f(n)^3$ koraka i odlučuje α.

Modificirajmo N tako da ne staje za $x \in \alpha$, ali da staje za $x \in \Sigma_0^* \backslash \alpha$. Označimo modificirani Turingov stroj sa S.
Kako je T univerzalni 2-tračni Turingov stroj slijedi da na njemu možemo simulirati S s nekim programom p (Vidi definiciju 1.15.).

Moguća su dva slučaja; $p \in \alpha$ ili $p \notin \alpha$.
Ako vrijedi $p \in \alpha$, tada za ulaz (p, p) stroj T staje, jer smo tako konstruirali jezik α. Međutim, kako T simulira S s programom p, slijedi da bi i S trebao stati za ulaz p. No, to je kontradikcija s načinom na koji smo definirali kada se S zaustavlja.
Dakle, moralo bi vrijediti $p \notin \alpha$.
No, sada po konstrukciji stroja S, slijedi da S staje za ulaz p i to u vremenu $|p|f(|p|)^3 < f(|p|)^4$. Zato, i T staje u $f(|p|)^4$ koraka. No, tada je po definiciji jezika α, $p \in \alpha$, što je naravno, kontradikcija.
Dakle, vrijedi $\alpha \notin \mathrm{DTIME}(f)$. ■

Sljedeći teorem pokazat će nam da činjenica da za dvije funkcije f i g vrijedi $f \leq g$, ne znači nužno da vrijedi $\mathrm{DTIME}(f) \subsetneq \mathrm{DTIME}(g)$. Opet, rekurzivnost će se pokazati kao zahvalno svojstvo za konstrukciju kontraprimjera.

Teorem 2.4 (Gap Teorem) *Za svaku rekurzivnu funkciju $g: \mathbf{N} \to \mathbf{N}$, takvu da vrijedi $g(n) \geq n$, postoji rekurzivna funkcija $f: \mathbf{N} \to \mathbf{N}$ takva da:*

$$\mathrm{DTIME}(g \circ f) = \mathrm{DTIME}(f).$$

Dokaz:
Neka je $T = (2, \Sigma, \Gamma, \delta_T)$ 2-tračni, univerzalni Turingov stroj. Označimo s $\tau(x, y)$ vrijeme računanja od T za ulaz (x, y), gdje su $x, y \in \Sigma_0^*$, a Σ abeceda stroja T (primijetimo da $\tau(x, y)$ može biti i beskonačno).
Dokazat ćemo dvije tvrdnje:

1) Postoji rekurzivna funkcija $f: \mathbf{N} \to \mathbf{N}$ takva da za sve $n > 0$ i za sve $x, y \in \Sigma_0^*$ vrijedi:
   Ako je $|x|, |y| \leq n$ onda je $\tau(x, y) \leq f(n)$ ili $\tau(x, y) \geq (g(f(n)))^3$.

2) Za takvu funkciju f vrijedi $\mathrm{DTIME}(g \circ f) = \mathrm{DTIME}(f)$.

Dokaz tvrdnje 1):
Funkciju f konstruiramo na sljedeći način:
Fiksirajmo proizvoljan $n \in \mathbf{N}$. Želimo definirati $f(n)$. U tu svrhu, prvo ćemo definirati broj $t \in \mathbf{N}$.
Uzmimo prvo $t := n + 1$. Sve uređene parove $(x, y) \in \Sigma_0^*$ za koje vrijedi $|x|, |y| \leq n$, poredajmo u konačan niz (primijetimo da ih zbog $|x|, |y| \leq n$, ima konačno mnogo).
Uzmimo prvi element (x, y) u nizu i dajmo ga Turingovom stroju T kao ulaz.
Dalje postupamo ovako da bi definirali t, odnosno $f(n)$:

a) Ako T stane unutar t koraka, izbacimo (x, y) iz niza.

b) Ako T stane u s koraka, gdje je $t < s < g(t)^3$, stavimo $t := s$ i opet, izbacimo (x, y) iz niza (primijetimo da ovdje koristimo činjenicu da postoji DTS koji računa $g(t)$ u konačno mnogo koraka, jer nam treba kao gornja ograda; $g(t)^3$ tada lako dobijemo).

c) Ako T ne stane niti nakon $g(t)^3$ koraka, tada ga zaustavimo te stavimo (x, y) na kraj niza.

d) Ako smo prošli cijeli niz bez da smo izbacili bilo koji njegov element, zaustavimo T i definirajmo $f(n) := t$.

Provjerimo da ovako definirana funkcija f zadovoljava uvjete tvrdnje 1).
Neka su $n \in \mathbf{N}$, $(x, y) \in \Sigma_0^*, |x|, |y| \leq n$, proizvoljni.
Pretpostavimo da ne vrijedi $\tau(x, y) \leq f(n)$, tj. da vrijedi $\tau(x, y) > f(n)$.
Kada bi vrijedilo $\tau(x, y) < (g(f(n)))^3$, tada prema slučaju *b*) iz prethodne definicije funkcije f, znamo da vrijedi, $f(n) \geq \tau(x, y)$, što je naravno kontradikcija. Dakle, vrijedi $\tau(x, y) \geq (g(f(n)))^3$.

Dokaz tvrdnje 2):
Kako je $g(f(n)) \geq f(n)$ (zbog $g(n) \geq n, \forall n$), odmah dobivamo

$$\mathrm{DTIME}(f) \subseteq \mathrm{DTIME}(g \circ f).$$

Dokažimo i obratnu inkluziju.
Neka je $\alpha \in \mathrm{DTIME}(g \circ f)$ proizvoljan jezik. Tada postoji Turingov stroj vremenske složenosti $g \circ f$ koji odlučuje α. No, tada postoji i ekvivalentni 1-tračni Turingov stroj S vremenske složenosti $(g(f(n)))^2$, koji odlučuje α (teorem 1.1). Sada S možemo simulirati na T s nekim programom p u $|p| \cdot g(f(n))^2$ koraka. Za dovoljno velike n ($f(n) \geq p$), slijedi da T za svaki ulaz oblika (y, p) ($|y| \leq n$) radi najviše $g(f(n))^3$ koraka (Opet, one y za koje

to ne vrijedi, prepoznamo direktno). No sada po definiciji funkcije f slijedi da T radi za svaki ulaz najviše $f(n)$ koraka. Dakle, T prepoznaje α u najviše $f(n)$ koraka, što povlači $\alpha \in \mathrm{DTIME}(f)$. Time smo dokazali i

$$\mathrm{DTIME}(g \circ f) \subseteq \mathrm{DTIME}(f)$$

čime je teorem dokazan. ■

Primjer 2.1 *Uočimo da prema upravo dokazanom teoremu postoje funkcije f_1, f_2 (rekurzivne) takve da vrijedi, na primjer:*

$$\mathrm{DTIME}(f_1(n)^2) = \mathrm{DTIME}(f_1(n))$$

$$\mathrm{DTIME}(2^{2^{f_2(n)}}) = \mathrm{DTIME}(f_2(n)).$$

Tvrdnje slijede, ako kao funkciju g iz prethodnog teorema uzmemo funkciju $g_1(n) = n^2$, odnosno $g_2(n) = 2^{2^n}$

Napomena 2.3 *Kao posljedica prethodnog teorema slijedi da postoji rekurzivna funkcija f za koju vrijedi*

$$\mathrm{DTIME}((m+1)^{f(n)}) = \mathrm{DTIME}(f(n))$$

a samim time za nju vrijedi i

$$\mathrm{DTIME}(f) = \mathrm{DSPACE}(f)$$

(pogledaj propoziciju 2.4).

Vratimo se na trenutak opet teoremu 2.1. Može se reći da je cilj teorije složenosti (i teorije algoritama, naravno), odrediti najbolje moguće rješenje za dani problem. Ipak, mi smo vidjeli (teorem 2.1) da kada se radi o Turingovim strojevima, jedinstvenog, najboljeg rješenja u biti nema. Sljedeći teorem opet nam to pokazuje (doduše u nešto slabijoj varijanti).

Teorem 2.5 (Speed-up Teorem) *Za svaku rekurzivnu funkciju $g: \mathbf{N} \to \mathbf{N}$ postoji rekurzivan jezik α takav da za svaki Turingov stroj T koji odlučuje α, postoji Turingov stroj S koji također odlučuje α, ali za kojega vrijedi*

$$g(time_S(n)) < time_T(n), \text{ za sve } n \in \mathbf{N}.$$

Dokaz:
Osnovna ideja, koja se provlači kroz dokaz teorema, je pronalazak riječi s određenim svojstvima, čije ćemo prepoznavanje prepustiti kontrolnoj jedinici. Naravno, time se Turingov stroj izrazito komplicira. Isto tako, konstrukcija

jezika α morat će biti takva da s ostalim riječima stroj radi efikasno.

Po lemi 2.1, postoji potpuno vremenski konstruktibilna funkcija f takva da je $g \leq f$. Ako dokažemo tvrdnju teorema za f, tada imamo:

$$g(time_S(n)) \leq f(time_S(n)) < time_T(n).$$

Zato možemo pretpostaviti da je g potpuno vremenski konstruktibilna.
Također, prema napomeni 2.2, možemo sada pretpostaviti i $g(n) > n$. Definirajmo, rekurzivno, novu funkciju $h\colon \mathbf{N} \to \mathbf{N}$:

$$h(0) = 1,\ h(n) = (g(h(n-1)))^3.$$

Funkcija h je monotono rastuća funkcija jer vrijedi

$$h(n+1) = (g(h(n)))^3 > (h(n))^3 \geq h(n)$$

te potpuno vremenski konstruktibilna jer je definirana pomoću g.

Označimo s $T_0 = (2, \Sigma, \Gamma, \delta_{T_0})$ 2-tračni univerzalni Turingov stroj, a za $x, y \in \Sigma_0^*$, s $\tau(x, y)$ vrijeme rada T_0 za ulaz (x, y) (koje može biti i beskonačno ako T_0 ne stane).
Par (x, y), zvati ćemo "brzi", ako vrijedi

$$|y| \leq |x| \text{ i } \tau(x, y) \leq h(|x| - |y|).$$

Riječi iz Σ_0^* poredajmo uzlazno po dužini i to tako da one iste dužine poredamo leksikografski. Označimo dobiveni niz s $x_1, x_2, \ldots$.
Za svaki $x_i, i = 1, 2, \ldots$ pokušat ćemo izdvojiti (ako je to moguće) riječ $y \in \{x_1, x_2, \ldots\}$ i označiti je sa y_i, ako vrijedi:

- y nije još izdvojen za $j < i$
- par (x_i, y) je brz
- ako ima više riječi koje zadovoljavaju prethodna dva uvjeta, y je najkraća među njima.

Označimo s α jezik koji se sastoji od svih x_i za koje postoji y_i s prethodno navedenim svojstvima, te za koje Turingov stroj T_0 staje za ulaz (x_i, y_i) s upisanom 0 na prvoj traci (tj. T_0 ne prihvaća x_i s programom y_i).
Prije definicije traženog Turingovog stroja S dokazujemo tri pomoćne tvrdnje.

Pomoćna tvrdnja 1*:*
Pokažimo prvo da je upravo definirani jezik α rekurzivan, štoviše, da je za svaki $k \in \mathbf{N}$ pitanje je li $x \in \alpha$ odlučivo u $h(n-k)$ koraka, ako je n dovoljno velik ($|x| = n$).
Po konstrukciji jezika α odlučiti da li je $x_i \in \alpha$ može se i na sljedeći način:

- odlučimo postoji li pripadni y_i
- u slučaju da postoji, nađemo pripadni y_i
- pokrenemo Turingov stroj T_0 s ulazom (x_i, y_i) (u najviše $h(|x_i| - |y_i|)$ koraka).

Samo zadnji korak prebacuje željeno vrijeme rada ($h(n-k)$), ako je $|y_i| \leq k$. Potrebno je izvršiti preinake na stroju (počinje kompliciranje Turingovog stroja T_0).
Kako svih parova oblika (x_i, y_i), gdje je $|y_i| \leq k$, ima konačno mnogo (zbog konačnosti abecede), možemo "listu" tih elemenata pohraniti direktno u centralnoj jedinici Turingovog stroja.
Od sada provjeru da li je $x \in \alpha$ počinjemo provjerom je li $x = x_i$ za neki par (x_i, y_i) iz te liste. Ako je, tada x prihvatimo (primijetimo da izuzev čitanja riječi x, ova provjera traje jedan korak).
Pretpostavimo sada da x nije u listi. Tada za njegov y_i (ako uopće postoji) vrijedi $|y_i| > k$. U tom slučaju, provjeravamo postoji li y, $k < |y| \leq |x|$, takav da je (x, y) brzi par.
Neka je $m = |\Sigma|$. Riječi duljine $k+1$ u abecedi Σ ima m^{k+1}, riječi duljine $k+2$ ima $m^{k+2}, \ldots$, riječi duljine n ima m^n. Za svaku od tih riječi moramo provjeriti čini li sa x brzi par. Ako je riječ y duljine l tada za provjeru čini li ona sa x brzi par trebamo najviše $h(|x| - l)$ koraka. Zato za provjeru svih riječi $k < |y| \leq |x|$ trebamo najviše

$$m^{k+1}h(n-k-1) + m^{k+2}h(n-k-2) + \cdots + m^n h(0)$$

koraka. Kako je h monotono rastuća funkcija tada je gornji izraz manji ili jednak od

$$m^{k+1}h(n-k-1) + m^{k+2}h(n-k-1) + \cdots + m^{n-1}h(n-k-1) + m^n$$

što je za $m > 1$ jednako

$$m^{k+1}\frac{m^{n-k-1}-1}{m-1}h(n-k-1) + m^n.$$

Indukcijom po $k \in \mathbf{N}$ za dovoljno velike $n \in \mathbf{N}$ i proizvoljan, ali fiksiran $m \in \mathbf{N}$ pokaže se da je prethodni izraz manji ili jednak od $(h(n-k-1))^3$. Sada vrijedi

$$h(n-k) = (g(h(n-k-1)))^3 > (h(n-k-1))^3.$$

Dakle, možemo u najviše $h(n-k)$ koraka provjeriti postoji li y duljine $k < |y| \leq |x|$ za kojeg vrijedi da je (x, y) brzi par.

Pomoćna tvrdnja 2*:*
Pokažimo sada da ako stroj T_0 s programom y odlučuje jezik α, tada y ne može biti jednak niti jednoj izdvojenoj riječi y_i.
Pretpostavimo da ipak vrijedi $y = y_i$ za neki $i \in \mathbf{N}$, te neka je x_i njegov pripadni par. Moguća su dva slučaja:

a) $x_i \in \alpha$
Tada T_0 za ulaz (x_i, y_i) daje izlaz 1 jer T_0 sa programom $y = y_i$ odlučuje jezik α. No tada po konstrukciji jezika α, imamo $x_i \notin \alpha$ čime je dobivena kontradikcija.

b) $x_i \notin \alpha$
Tada po konstrukciji jezika α, stroj T_0 za ulaz (x_i, y_i) daje izlaz 1. Kako ujedno T_0 sa programom $y = y_i$ odlučuje α slijedi da vrijedi $x_i \in \alpha$. Time je dobivena kontradikcija.

Kako smo dobili kontradikciju u oba slučaja, tvrdnja vrijedi.

Pomoćna tvrdnja 3*:*
Pokažimo još da ako stroj T_0 s programom y odlučuje α onda par (x, y) može biti brz samo za konačno mnogo riječi x.
Pretpostavimo da je (x, y) brz (odnosno da je $x = x_i$). To znači da je y jedan od kandidata za y_i. Specijalno, zbog načina odabira y_i, sada sigurno znamo da će vrijediti $|y_i| \leq |y|$. Promotrimo sada sve brze parove oblika (x_j, y_j), gdje je $|y_j| \leq |y|$ (njih je konačno mnogo jer je riječi y_j za koje vrijedi $|y_j| \leq |y|$ konačno mnogo). Kada x ne bi bio među njima, tada bi za njegov y moralo vrijediti $|y| > |y|$, što naravno ne vrijedi.

Nakon svega, preostaje nam još samo definirati Turingov stroj S iz tvrdnje teorema.
Kako je jezik α rekurzivan, tada po definiciji postoji k-tračni Turingov stroj T koji ga odlučuje. Neka je T_1 1-tračni Turingov stroj koji također odlučuje α u vremenu (postoji po teoremu 1.1)

$$time_{T_1}(n) \leq c \cdot (time_T(n))^2,$$

za neku konstantu $c > 0$. Kako je T_0 univerzalni Turingov stroj tada može simulirati T_1 s nekim programom y u vremenu

$$\tau(x, y) \leq |y| \cdot c \cdot (time_T(|x|))^2 \leq (time_T(|x|))^3$$

za sve dovoljno duge riječi x (recimo $|x| \geq |y| \cdot c$).
Prema dokazanom, (*Pomoćna tvrdnja* (3)) znamo da vrijedi

$$\tau(x, y) \geq h(|x| - |y|)$$

za sve osim konačno mnogo riječi x. Samim time vrijedi i

$$(h(n - |y|))^{\frac{1}{3}} \leq time_T(n).$$

Iz dokazane *pomoćne tvrdnje* 1, slijedi da možemo konstruirati Turingov stroj S koji odlučuje α u vremenu $h(n-|y|-1)$ (uzmemo $k = |y|-1$), te za kojega vrijedi

$$g(time_S(n)) \leq g(h(n - |y| - 1)) \leq (h(n - |y|))^{\frac{1}{3}} \leq time_T(n).$$

Time je ovaj teorem dokazan. ■

Napomena 2.4 *Primijetimo da ovaj teorem govori samo o postojanju ubrzivog jezika, za razliku od Linear Speedup Teorema, koji je primjenjiv na svaki odlučivi jezik. Za sada, za proizvoljan odlučiv jezik, bolje od linearnog ubrzanja, ne znamo.*

Ovim teoremom završavamo ovo poglavlje. Pokazali smo neke općenite rezultate o klasama složenosti kao i rezultate o nekim zanimljivim odnosima među istima. Ipak, sljedeće poglavlje uvodi u igru nedeterminizam. Pokazat će se da će to dovesti do interesantnih saznanja.

3 NTIME i NSPACE

3.1 Uvod

U prošlom smo poglavlju iznijeli neke od važnih rezultata o odnosima klasa složenosti. Svi ti rezulati bili su usko vezani uz Turingov stroj, odnosno puno točnije, uz deterministički Turingov stroj. Što će se dogoditi kada uvedemo nedeterminizam u igru? Znamo s jedne strane, nedeterministički je stroj puno brži od ekvivalentnog mu deterističkog (vidi teorem 1.2). No, štedi li i prostor? Kakav će biti odnos prostornih i vremenskih klasa uz determinizam i bez njega?
Na ova, a i neka druga pitanja, odgovor ćemo dati u ovom poglavlju.

3.2 Nedeterminističke klase složenosti

Krenimo odmah sa definiranjem novih klasa.

Definicija 3.1 *Neka je* $f: \mathbf{N} \to \mathbf{R}$*, proizvoljna funkcija.*

- *Sa* $\mathbf{NTIME}(f)$ *označavamo klasu svih jezika odlučivih na nekom NTS* T*, za kojeg vrijedi* $time_T = O(f)$*.*
- *Sa* $\mathbf{NSPACE}(f)$ *označavamo klasu svih jezika odlučivih na nekom NTS* T*, za kojeg vrijedi* $space_T = O(f)$*.*

Ove klase nedeterministički su analogoni klasa $\mathrm{DTIME}(f)$ i $\mathrm{DSPACE}(f)$.

Prije nego što definiramo sljedeću klasu trebati će nam jedan pomoćni pojam i neki rezultati u vezi njega.

Definicija 3.2 *Neka su* $f: \mathbf{N} \to \mathbf{N}$ *i* $g: \mathbf{N} \to \mathbf{N}$ *dvije funkcije. Neka vrijedi* $g(n) \geq n$*, za svaki* $n \in \mathbf{N}$*.*
Kažemo da je jezik $\alpha_0 \in \mathrm{DTIME}(g)$ ***svjedok*** *duljine* f *i vremena* g *za jezik* $L \subseteq \Sigma_0^*$*, ako vrijedi:*

$$x \in L \iff \exists \text{ riječ } y \in \Sigma_0^* \text{ takva da } |y| \leq f(|x|) \text{ i } x\&y \in \alpha_0.$$

Ovdje je znak & rezerviran za razdvajanje riječi x *i* y*.*

Teorem 3.1 *Neka su* $f, g: \mathbf{N} \to \mathbf{N}$*, dvije funkcije.*

a) Svaki jezik $\alpha \in \mathrm{NTIME}(f)$ *ima svjedoka duljine* $O(f)$ *i vremena* $O(n)$*.*

b) Ako jezik α ima svjedoka duljine f i vremena g tada je $\alpha \in \text{NTIME}(g(f(n)+n+1))$

Dokaz:
a)
Kako je $\alpha \in \text{NTIME}(f)$ tada postoji NTS $T = (k, \Sigma, \Gamma, \Phi)$ koji odlučuje α u vremenu f.

Svakoj riječi $x \in \alpha$ pridružit ćemo postupak izračunavanja od T koji prihvaća x u vremenu $f(|x|)$ (vidi definiciju 1.12).
Neka je α_0 jezik svih riječi oblika $x\&y$, gdje je y postupak izračunavanja koji prihvaća x.
Pokažimo da je $\alpha_0 \in \text{DTIME}(O(n))$.
Definirajmo DTS $M = (k+1, \Sigma, \Gamma_M, \delta)$. Neka se na prvoj traci nalazi $x\&y$. Treba provjeriti da li je y postupak izračunavanja koji prihvaća x.
Stroj M neka prvo prepiše x na drugu traku, a na prvoj neka pobriše x i $\&$. Neka su glave stroja M na trakama $2, \ldots, k+1$ pozicionirane na nultim ćelijama, a glava prve trake neka je pozicionirana na početku riječi y. Stroj M redom čita petorke na prvoj traci i simulira njihovo djelovanje na trakama $2, \ldots, k+1$.
Naravno u svakom koraku rada, stanje na trakama mora odgovarati zahtijevima koje petorke traže. Kako su petorke konstantne duljine, provjeru je moguće izvesti u konstantnom broju koraka. Ako u bilo kojem trenutku, stanje na trakama ne odgovara zahtijevima pročitane petorke (ili obratno), M ne privača ulaz. Inače, ako pročita sve petorke i uspješno simulira postupak izračunavanja, te postupak izračunavanja prihvati riječ x, tada i M prihvati svoj ulaz.
Jasno je da M samo jednom prolazi kroz riječ zapisanu na prvoj traci, te nakon toga uvijek staje. Dakle, $\alpha_0 \in \text{DTIME}(O(n))$.

Da bi α_0 bio svjedok za α, moramo još pokazati da je svaki postupak izračunavanja koji prihvača x, duljine $O(f(|x|))$. No, to slijedi iz napomene 1.9.

b)
Neka je α_0 svjedok za α, duljine f i vremena g, te neka je S DTS koji odlučuje α_0 u vremenu g.

Konstruirajmo 2-tračni NTS T na sljedeći način.
Za ulaz $x \in \Sigma_0^*$, zapisan na prvoj traci, neka deterministički izračuna $f(|x|)$ (tu izgleda pretpostavljamo da je f dobro izračunljiva ???????????) i neka zapiše $f(|x|)$ znakova 1 na drugu traku. Nakon toga, neka zapiše znak & na kraju ulaza x.
Sada dolazimo do nedeterminizma. Plan je generirati (nedeterministički) sve

riječi $y \in \Sigma_0^*$ takve da je $|y| \leq f(|x|)$, i nadovezati ih na $x\&$.
Dok god T čita znak 1 s druge trake, nedeterministički postupa na sljedećih $|\Sigma_0| + 1$ načina:

- Piše znak $s \in \Sigma_0$ na prvu traku, pomiče glavu na prvoj traci jedno mjesto u desno, a na drugoj jedno mjesto u lijevo. ($|\Sigma_0|$ načina)
- Ne piše ništa, ne miče glave i prelazi u novo stanje, označimo ga sa START2. (1 način)

Primijetimo da smo na ovaj način generirali sve riječ iz Σ_0, duljine najviše $f(|x|)$.
Nakon što T pročita na prvoj traci prvi znak koji nije 1, T ne radeči ništa prelazi u stanje START2 (ovime smo osigurali da sve grane dođu do stanja START2, to nam je bitno jer T mora odlučivati α pa sve grane moraju stati).

U stanju START2, stroj T radi sljedeće:

- pomakne glavu prve trake na početnu (nultu) ćeliju (da bude na početku riječi $x\&y$)
- obriše sadržaj druge trake
- radi isto što i S (od ovog trenutka stroj T je isti kao i stroj S).

Kako stroj S odlučuje α_0 sve grane od T staju sa 1 ili 0 kao izlazom. Ako sve grane stanu s izlazom 0, tada ne postoji riječ $y \in \Sigma_0^*$, duljine najviše $f(|x|)$ takva da je $x\&y \in \alpha_0$, pa $x \notin \alpha$ jer je α_0 svjedok za α. Ako jedna grana stane sa izlazom 1, tada analogno imamo $x \in \alpha$.
Dakle, T odlučuje α.

Još nam ostaje pokazati da je zadovoljeno i vrijeme rada za T, iz tvrdnje teorema.
Prvi dio (računanje $f(|x|)$ i generiranje riječi y) traži $O(f(|x|))$, a drugi dio (provjera je li $x\&y$ u α_0), $g(|x|+1+f(|x|))$ vremena. Kako je po pretpostavci, $g(n) \geq n$, tvrdnja teorema vrijedi. ■

Korolar 3.1 *Neka je $\alpha \subset \Sigma_0^*$, proizvoljan jezik. Sljedeće tvrdnje su ekvivalentne:*

- *Postoji polinoman NTS koji odlučuje α.*
- *α ima svjedoka polinomne duljine i polinomnog vremena.*

Dokaz:
Neka je T NTS koji odlučuje α u vremenu p, gdje je p polinom. Tada je $T \in \text{NTIME}(p)$. Po prethodnom teoremu α ima svjedoka duljine $O(p)$ i vremena $O(n)$. Dakle, α ima svjedoka polinomne duljine i vremena.

Obratno, neka α ima svjedoka polinomne duljine, recimo p, i polinomnog vremena, recimo h. Opet, prema prethodnom teoremu slijedi da je $\alpha \in \text{NTIME}(h(p(n)+n+1))$. Kako je kompozicija polinoma opet polinom, tvrdnja vrijedi. ■

Definicija 3.3 *Sa* **NP** *označit ćemo klasu jezika koja ima svojstva iz prethodnog korolara, tj.* NP *je klasa jezika odlučivih na polinomnom NTS.*

Klasa NP jednako je važna klasa kao i P. Vidjeti ćemo da ona također sadrži velik broj poznatih i često korištenih problema. Kako je svaki DTS ujedno i NTS odmah slijedi

$$\text{DTIME}(f) \subseteq \text{NTIME}(f).$$

Specijalno vrijedi i $\text{P} \subseteq \text{NP}$. No vrijedi li stroga inkluzija nije dokazano, iako se vjeruje da vrijedi.
Postoje mnoge varijante pitanja vrijedi li stroga inkluzija ili ne. Taj je problem poznat kao P versus NP pitanje. Važnu ulogu tu će imati i takozvani, NP-potpuni problemi, u neku ruku, pravi predstavnici klase NP. O NP-potpunim problemima više ćemo reći u sljedećem poglavlju.
Spomenimo sada neke od problema koji pripadaju klasi NP.

3.3 Primjeri problema u klasi NP

Ovaj put počinjemo s problemom iz teorije grafova. Razmotrit ćemo problem traženja Hamiltonovog puta u usmjerenom grafu.

Definicija 3.4 *Neka je G usmjereni graf.* ***Hamiltonov put*** *u grafu G je put koji sadrži sve vrhove grafa G. Hamiltonov put obilazi svaki čvor grafa točno jednom.*

U sljedećem teoremu pokazat ćemo da je jednostavno provjeriti je li put u nekom grafu Hamiltonov. Međutim, pokazat ćemo i da je problem pronalaska, odnosno problem egzistencije Hamiltonovog puta u grafu jako težak.

Označimo s $HPUT$ jezik sastavljen od svih uređenih trojki (G, v, w), gdje je G usmjereni graf u kojem postoji Hamiltonov put od vrha v do vrha w (tj. put sa početkom u vrhu v i krajem u vrhu w koji je Hamiltonov[22]).
Dokažimo da je $HPUT \in \text{NP}$.

[22]Vidi definiciju 1.23.

Teorem 3.2 $HPUT \in \mathrm{NP}$

Dokaz:
Neka je $G = (V_G, E_G, \varphi)$ proizvoljan usmjereni graf, te neka su v, w proizvoljni vrhovi iz E_G. Označim sa $m = |V_G|$.

Konstruirat ćemo NTS koji za ulaz (G, v, w) provjerava postoji li u G Hamiltonov put od v do w. Kao što smo rekli, a i pokazat ćemo, provjera je li put Hamiltonov je relativno jednostavna. Ono što oduzima najviše vremena je generiranje svih puteva koji sadrže točno m vrhova.

Promotrimo NTS N koji za ulaz (G, v, w), radi na sljedeći način:

1) Generira niz $v_1, \ldots, v_m$ od m vrhova. Niz generira tako da u svakom koraku nedeterministički odabere jedan od m vrhova grafa.

2) Provjeri da li u nizu ima vrhova koji se ponavljaju. Ako ima odbaci ulaz.

3) Provjeri je li $v_1 = v$ i $v_m = w$. Ako nije odbaci ulaz.

4) Za svaki $i \in \{1, \ldots, m-1\}$, provjeri postoji li luk u G od v_i do v_{i+1}. Ako svi lukovi postoje, tada prihvati ulaz, a inače ga odbaci.

Da bi dokazali teorem, dovoljno je pokazati da je svaki od ovih koraka vremenski polinoman.

U 1. koraku trebamo generirati niz vrhova duljine m. Kako je svaki vrh kodiran konstantnim brojem znakova, cijeli niz možemo generirati u $O(m)$ koraka. Kako je N nedeterministički pa zato svaka grana generira drugačiji niz, slijedi da N radi $O(m)$ koraka da bi generirao sve nizove vrhova duljine m.

2. korak možemo izvesti tako da za svaki vrh v_i iz niza još jednom prođemo kroz cijeli niz i provjerimo postoji li vrh $v_j, j \neq i$, takav da je $v_i = v_j$. Očito, to možemo izvesti u $O(m^2)$ koraka (Primijetimo da iako je N nedeterministički, ovaj dio njega je deterministički.).

3. korak je jako jednostavan i traži samo jedan prolazak kroz niz vrhova, odnosno $O(m)$ koraka (I ovaj dio stroja N je deterministički.).

4. korak za svaka dva susjedna vrha u nizu provjerava postoji li luk među njima. Slično kao bridove, i lukove kodiramo s konstantnim brojem znakova. Zato cijeli skup E_G možemo kodirati sa $O(|E_G|)$ znakova. Dakle, za svaki susjedan par vrhova u nizu pretražujemo skup bridova, što zahtijeva

$O(m|E_G|)$ koraka (Opet, deterministički dio stroja.).

Dakle, sva četiri koraka N izvodi u polinomijalnom vremenu. Naravno, 1. korak je za razliku od ostalih, nedeterministički, te je on i razlog zašto je ovaj jezik u NP. ■

Sada ćemo obraditi jedan problem koji je usko vezan za logiku. Riječ je o problemu ispunjivosti formule logike sudova. Definirajmo potrebne pojmove.

Definicija 3.5 ***Alfabet logike sudova*** *je unija skupova A_1, A_2 i A_3, gdje je:*

- $A_1 = \{X_0, X_1, X_2, \ldots\}$ *prebrojiv skup čije elemente nazivamo propozicionalne varijable*
- $A_2 = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ *skup pomoćnih veznika*
- $A_3 = \{(,)\}$ *skup pomoćnih simbola.*

Definirajmo sada posebnu vrstu riječi nad alfabetom logike sudova.

Definicija 3.6 ***Atomarna formula*** *je svaka propozicionalna varijabla. Pojam* ***formule*** *definiramo induktivno:*

a) svaka atomarna formula je formula

b) ako su A i B formule tada su i riječi $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$ i $(A \leftrightarrow B)$ također formule

c) riječ alfabeta logike sudova je formula ako je nastala primjenom konačno mnogo koraka uvjeta a) i b).

Moramo još definirati pojam ispunjivosti formule.

Definicija 3.7 *Svako preslikavanje sa skupa svih propozicionalnih varijabli u skup $\{0, 1\}$, tj. $I\colon \{X_0, X_1, X_2, \ldots\} \rightarrow \{0, 1\}$, nazivamo* ***interpretacija****.*

Definicija 3.8 *Neka je I interpretacija. Tada vrijednost interpretacije I na proizvoljnoj formuli definiramo induktivno:*

$$\begin{aligned}
I(\neg A) = 1 \quad & \textit{ako i samo ako} \quad I(A) = 0 \\
I(A \wedge B) = 1 \quad & \textit{ako i samo ako} \quad I(A) = 1 \textit{ i } I(B) = 1 \\
I(A \vee B) = 1 \quad & \textit{ako i samo ako} \quad I(A) = 1 \textit{ ili } I(B) = 1 \\
I(A \rightarrow B) = 1 \quad & \textit{ako i samo ako} \quad I(A) = 0 \textit{ ili } I(B) = 1 \\
I(A \leftrightarrow B) = 1 \quad & \textit{ako i samo ako} \quad I(A) = I(B).
\end{aligned}$$

Definicija 3.9 *Ako je vrijednost interpretacije J na formuli F jednaka 1, tj. $J(F) = 1$, tada kažemo da je formula F* ***istinita za interpretaciju*** *J. Inače, ako je $J(F) = 0$, tada kažemo da je formula F* ***neistinita za interpretaciju*** *J.*
Kažemo da je formula F ***ispunjiva****, ako postoji interpretacija I takva da vrijedi $I(F) = 1$.*

Definicija 3.10 *Kažemo da su formule A i B* ***logički ekvivalentne****, i označavamo $A \Leftrightarrow B$, ako za svaku interpretaciju I vrijedi $I(A) = I(B)$.*

Sljedeći pojmovi bitno će nam olakšati proučavanje formula.

Definicija 3.11

- *Atomarnu formulu i njezinu negaciju zovemo* ***literal****.*
- *Neka su $A_1, \ldots, A_n$ proizvoljne formule. Formulu koja ima oblik $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ nazivamo konjunkcija, a formulu koja ima oblik $A_1 \vee A_2 \vee \ldots \vee A_n$ nazivamo disjunkcija.*
- ***Elementarna konjunkcija*** *je konjunkcija literala, a* ***elementarna disjunkcija*** *je disjunkcija literala.*
- ***Konjunktivna normalna forma*** *je konjunkcija elementarnih disjunkcija.*
- ***Disjunktivna normalna forma*** *je disjunkcija elementarnih konjunkcija.*

Definicija 3.12 *Neka je A formula, neka je B konjunktivna normalna forma, te neka je C disjunktivna normalna forma.*
Kažemo da je B ***konjunktivna normalna forma za*** *A ako vrijedi $A \Leftrightarrow B$.*
Kažemo da je C ***disjunktivna normalna forma za*** *A ako vrijedi $A \Leftrightarrow C$.*

Sljedeći teorem dajemo bez dokaza[23].

Teorem 3.3 *Za svaku formulu logike sudova postoji konjunktivna i disjunktivna normalna forma.*

Ovim smo teoremom pitanje ispunjivosti formule logike sudova prebacili na pitanje disjunktivne, odnosno konjunktivne normalne forme.

Prije nego što krenemo na sam problem, kažimo nešto o alfabetu logike sudova. Alfabet logike sudova je prebrojiv, a mi smo abecedu definirali kao

[23] Za dokaz vidi [7, str. 21]

konačan skup. Da ne bi bilo nesporazuma, primijetimo da svaku propozicionalnu varijablu možemo kodirati samo znakovima $X, 0$ i 1. Naime, indeks propozicionalne varijable možemo zapisati binarno. Dakle, za zapis proizvoljne riječi nad alfabetom logike sudova dovoljan nam je skup $\{X, 0, 1\} \cup A_2 \cup A_3$, koji je konačan. Sada možemo krenuti dalje.

Označimo sa SAT jezik sastavljen od svih ispunjivih konjunktivnih normalnih formi. Napokon možemo iskazati sljedeći teorem.

Teorem 3.4 SAT $\in$ NP

Dokaz:
Neka je w proizvoljna konjunktivna normalna forma, te neka je $|w| = n$. Neka je $m \leq n$ broj različitih propozicionalnih varijabli koje se pojavljuju u formuli w.

Ideja dokaza je relativno jednostavna: pridružiti na sve moguće načine vrijednost 0 ili 1 propozicionalnim varijablama u formuli w. Jasno je da moramo generirati 2^m različtih pridruživanja, i da je taj dio vremenski najzahtjevniji. Međutim NTS to, zahvaljujući nedeterminizmu (za razliku od DTS), relativno lako napravi. Ostatak rada stroja je deterministički.

Promotrimo NTS T koji se sastoji od sljedeća dva dijela:

a) Pridružuje na sve moguće načine vrijednost 0 ili 1 propozicionalnim varijablama u formuli w.

b) Nakon što je generirao pridruživanje, provjerava je li formula istinita za dano pridruživanje.

Opišimo sada detaljnije rad dijela stroja T za pridruživanje (nedeterministički dio).
Možemo ga podijeliti na sljedeća tri koraka:

1) Prepisuje na posebnu, izdvojenu traku (zvat ćemo je generator) sve znakove dok ne dođe do zapisa neke propozicionalne varijable ili dok ne dođe do kraja ulaznog podatka. U prvom slučaju ode na korak 2), a u drugom na korak 3).

2) Pročita propoziconalnu varijablu. Provjeri je li već toj propozicionalnoj varijabli pridružio neku vrijednost (imamo i posebnu traku, prateću traku, na koju zapisuje koju vrijednost ima koja propozicionalna varijabla). Ako je, tada na traku generator zapiše tu istu vrijednost.
U slučaju da se ta propozicionalna varijabla prvi puta pojavljuje, tada joj nedeterministički pridruži vrijednost 1, odnosno vrijednost 0. Zapiše

pridruženu vrijednost na traku generator, odnosno prateću traku. Ode u korak 1).

3) Primijetimo da je u ovom trenutku na traci generator zapisana formula w sa 0 ili 1 na mjestu literala. Sada može početi provjeru istinitosti formule zapisane na traci generator. Stroj samo prelazi u dio za provjeru istinitosti formule.

Sada slijedi dio stroja T koji je deterministički. Njegov je zadatak provjeriti istinitost formule zapisane na traci generator. Kako je formula w konjunktivna normalna forma, rad stroja koncentrirat će se na elementarne disjunkcije u w.

1) Promatra sljedeću elementarnu disjunkciju.

2) Ako se ona sastoji od bar dva literala (u biti 0 ili 1) ode na korak 3), a inače na korak 4).

3) Uzme sljedeća dva literala X i Y, (opet, sada su to 0 ili 1) u promatranoj elementarnoj disjunkciji, i pogleda koja je vrijednost formule $X \vee Y$, obriši je, i na traku zapiši samo njenu vrijednost. Ode na korak 2).

4) Ako je na kraju ostala samo 0 tada ova grana ne prihvaća ulaz jer je vrijednost od w jednaka 0. Ako je vrijednost 1, tada ako je to zadnja elementarna disjunkcija u formuli, prihvati ulaz, a inače ode na korak 1).

Ako sve grane stroja T ne prihvate ulaz tada formula nije ispunjiva. Inače, ako bar jedna grana prihvati ulaz tada je formula ispunjiva.
Da bi dokazali teorem moramo još pokazati da je T vremenski polinoman.

Prvi dio stroja se za svaku propozicionalnu varijablu grana na dva slučaja, jedan u kojem toj propozicionalnoj varijabli pridružuje vrijednost 1, te drugi u kojoj joj pridružuje vrijednost 0. Iz konstrukcije je jasno da stroj prolazi formulom w samo jednom. Ipak tu treba u svakom koraku uračunati i vrijeme potrebno za provjeru je li nekoj propozicionalnoj varijabli već pridružena vrijednost. Propozicionalnih varijabli ima $m \leq n$, pa je i to moguće napraviti u $O(n)$ koraka. Kako nas zanima najduža grana, slijedi da je vremenska složenost dijela za pridruživanje najviše $O(n^2)$.

Dio za ispitivanje je deterministički, a i puno jednostavniji. Krećemo se kroz formulu promatrajući u njoj elementarne disjunkcije. U svakoj elementarnoj disjunkciji smanjujemo broj literala za jedan dok na kraju ne ostane samo

jedan literal. Očito, broj koraka je polinoman u ovisnosti o broju elementarnih disjunkcija ($< n$) i o duljinama elementarnih disjunkcija ($< n$). U svakom slučaju, i ovaj dio traži polinoman broj koraka.
Oba dijela stroja rade polinoman broj koraka pa je teorem dokazan. ■

Na jezik SAT još ćemo se vratiti kad budemo govorili o NP-potpunim problemima.
Sljedeći odjeljak povezat će deterministične i nedeterministične klase, te pružiti bolji uvid u neka poznate probleme.

3.4 Opći rezultati o odnosima klasa složenosti

Prvo definirajmo jedan pomoćni pojam.

Definicija 3.13 *Kažemo da je funkcija $f\colon \mathbf{N} \to \mathbf{N}$, **dobro izračunljiva**, ako postoji deterministički Turingov stroj, T, koji izračunava funkciju f u vremenu $O(f)$.*

Napomena 3.1 *Koristimo li unarni prikaz podataka funkcija f iz prethodne definicije zadovoljavat će $f(n) > n$, jer Turingov stroj mora barem pročitati ulaz i još jedan znak koji označava kraj ulaza.*
Koristimo li binarno kodiranje, zbog istih će razloga vrijediti $f(n) > \log_2 n$.

Krenimo sada na prve rezultate.

Teorem 3.5 *Neka je $f\colon \mathbf{N} \to \mathbf{N}$ dobro izračunljiva funkcija. Tada vrijedi:*

a) $\mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$

b) $\mathrm{NSPACE}(f) \subseteq \bigcup_{c>0} \mathrm{DTIME}(2^{cf(n)})$

Dokaz:
a)
Neka je $\alpha \in \mathrm{NTIME}(f)$ proizvoljan jezik. Neka je $T = (k, \Sigma, \Gamma, \Phi)$ neki NTS koji odlučuje jezik α u vremenu f. To znači da najduža grana od T pri odlučivanju radi $f(n)$ koraka za ulaz duljine n.
Ideja dokaza je isprobavati redom sve postupke izračunavanja od T (odnosno, sve grane), koristeći pritom samo onoliko prostora koliko koristi jedan postupak izračunavanja. Trebat će nam i još malo dodatnog prostora za praćenje već provjerenih postupaka izračunavanja.

Fiksirajmo proizvoljne uređaje na Σ i Γ. Time smo dobili i leksikografski uređaj na Φ.
Konstruirat ćemo DTS $S = (k+1, \Sigma, \Gamma_S, \delta)$ koji će simulirati rad T.

Stroj S ima jednu dodatnu traku, koju ćemo zvati adresna traka, na koju će zapisivati koji je postupak izračunavanja odabrao.
Neka je stroju S dan neki ulaz x. Stroj S u svakom koraku bira jedan od mogućih načina (bira po redoslijedu induciranom leksikografskim uređajem na Φ) na koji T može postupiti (znamo, T je NTS, pa potencijalno ima više mogućnosti). Odabrani način (petorku) zapisuje na adresnu traku i opet tako postupa dalje.
Kako T odlučuje α, sve grane staju, pa će tako i odabrani postupak izračunavanja stati. Ako ta grana prihvati riječ x tada i S prihvaća x. Inače, ako odabrani postupak izračunavanja ne prihvati x, tada S na adresnoj traci traži zadnji način (petorku) gdje je mogao postupiti drugačije. Neka je to j-ta petorka. Sada S ponovno simulira postupak izračunavanja koji se nalazi na adresnoj traci, ali samo do j-tog koraka (sve petorke, počevši od j-te pa nadalje, može obrisati). Zatim, umjesto j-te petorke bira prvu sljedeću (opet, u smislu leksikografskog uređaja na Φ) i, naravno, zapisuje je na adresnu traku. Sada s radom nastavlja kao i prije.
Ako ni jedna grana od T ne prihvati x, tada ni S ne prihvaća x.

DTS S, redom isprobava sve moguće postupke izračunavanja od T. Pri tome uvijek koristi onoliko prostora koliko i T (što je najviše $kf(n)$), jer u svakom koraku može promijeniti najviše po jednu ćeliju na svakoj od k traka), s time da još moramo uračunati prostor potrošen na adresnoj traci za zapisivanje postupka izračunavanja (koji iznosi, $O(f(n))$[24]).
Dakle, S je DTS koji odlućuje α, koristeći pritom $O(f)$ ćelija.
Time je tvrdnja a) dokazana.

b)
Neka je $T = (k, \Sigma, \Gamma, \Phi)$ neki NTS prostorne složenosti f, koji odlučuje α.
Ovaj put umjesto postupka izračunavanja promatrat ćemo sve moguće konfiguracije stroja[25]. Kako T odlučuje α, jasno je da se dvije identične konfiguracije ne mogu pojaviti jer bi se T zavrtio u ciklusu, te ne bi stao. Kao što smo vidjeli, broj različitih konfiguracija je najviše $2^{O(f(n))}$.
Ideja dokaza je za zadani ulaz x naći niz konfiguracija koje će odgovarati postupku izračunavanja i prihvaćanja riječi x. Problem ćemo modelirati usmjerenim grafom.

Promotrimo usmjereni graf G kojemu je skup vrhova skup svih mogućih konfiguracija, a luk između vrha u i v postoji ako i samo ako postoji uređena petorka $p \in \Phi$, takva da T prelazi iz konfiguracije u u konfiguraciju v u jednom koraku pomoću p.

[24] Vidi napomenu 1.9.
[25] Vidi definicju 1.13 i napomenu 1.10.

Stroj T prihvaća x ako i samo kao postoji usmjereni put od početne konfiguracije (u kojoj je stanje START, a na trakama se nalazi samo x), do neke završne konfiguracije (u kojoj je stanje STOP, a u nultoj ćeliji prve trake zapisan je znak 1).
U biti tražimo maksimalno stablo (po broju vrhova) s početnom konfiguracijom kao korijenom. Ako neka završna konfiguracija bude u tom stablu, tada prihvatimo ulaz. Inače, ako niti jedna završna konfiguracija nije u stablu, tada ulaz odbacujemo.
Ovdje, možemo iskoristiti propoziciju 2.2 u specijalnom slučaju kada svaki luk ima jednaku težinu.
Prema tome, možemo odlučiti pripada li x jeziku α u vremenu

$$O(|V(G)|^2) = O((2^{O(f(n))})^2) = 2^{O(f(n))}$$

čime je teorem dokazan. ■

Znamo da nedeterminizam štedi vrijeme. Sada nam je cilj pokazati da nedeterminizmom ne možemo uštedjeti puno prostora. O toj činjenici govori poznati Savitchev teorem. Ipak, prije toga će nam opet trebati novi pojam.

Za dokaz Savitchevog teorema opet ćemo koristiti usmjereni graf G definiran u dokazu prethodnog teorema. Ipak, nećemo konstruirati cijeli graf, jer bi samo time na veliko prebacili dozvoljenu količinu prostora. Zato ćemo pretpostaviti da nam je graf G dan s takozvanim "prorokom". Prorok je (bar za nas) konstrukt, koji na postavljeno pitanje trenutno odgovara (točno). Za nas, prorok će moći odmah odovoriti postoji li luk između dva proizvoljno odabrana vrha u grafu G.

Prilagodimo pojam proroka za graf G Turingovom stroju.
Prvo, prorok trenutno odgovara, što će u našem slučaju značiti da daje odgovor u jednom koraku rada stroja.

Definicija 3.14 ***Turingov stroj s prorokom***[26] *za graf G je Turingov stroj koji ima:*

- 3 *izdvojene trake koje koristi samo prorok.*
- *Izdvojeno stanje koje zovemo PROROK. U tom stanju stroj u jednom koraku odgovara jesu li riječi na prvim dvjema trakama vrhovi grafa G povezani lukom, i ovisno o tome na treću traku kao izlaz daje* 1 *ili* 0. *Nakon toga stroj iz stanja PROROK prelazi u stanje START.*

[26] engleski: Turing machine with oracle

Kada stroj uđe u stanje PROROK kažemo da postavlja pitanje proroku. Pitanje je dano kao par riječi zapisanih na prvim dvjema trakama stroja. Odgovor se ispisuje na treću traku.

Vrijedi sljedeća lema.

Lema 3.1 *Neka je dan graf G kojem skup vrhova čine sve riječi duljine t. Tada postoji Turingov stroj s prorokom za G koji za vrhove u, v i prirodan broj q odlučuje postoji li put duljine najviše 2^q od vrha u do vrha v. Također, stroj koristi najviše $O(qt)$ ćelija.*

Dokaz:
Konstruirat ćemo Turingov stroj koji ima još dvije dodatne trake, povrh onih za proroka.
Rad stroja odvija se u fazama, i to rekurzivno.

Na početku rada stroja prva traka sadrži par (u, q), a druga par (v, q). Neka u nekom stanju obje trake sadrže nekoliko parova (x, r), gdje je x ime vrha, a $r \leq q$, prirodan broj.
Neka su (x, r), (y, s) zadnji par na prvoj, odnosno na drugoj traci. Neka je $Min = min\{r, s\}$. Stroj postavlja pitanje proroku postoji li put duljine najviše 2^{Min} između vrhova x i y.

U slučaju da je $Min = 0$ tada tražimo put duljine najviše 1. No, to je ekvivalentno pitanju jesu li x i y jedan te isti vrh (put duljine 0) ili jesu li oni povezani lukom (put duljine 1). U svakom slučaju, prorok nam u jednom koraku daje odgovor.

Ako je $Min \geq 1$, tada definiramo $min = Min - 1$. Želimo naći vrh w u G, takav da postoje putevi od w do y i od x do w, duljine najviše 2^{min}. Oni bi dali put između x i y, duljine najviše 2^{Min}.
Uzmimo da je na skupu vrhova zadan neki leksikografski uređaj. Rad stroja se dalje odvija ovako:

1) Odabere prvi (leksikografski) vrh w i napiše par (w, min) na kraj prve trake (Pod krajem trake mislimo na kraj zapisa koji se nalazi na traci.). Odredimo rekurzivno postoji li put od w do y duljine najviše 2^{min}.

2) Ako postoji, tada napišemo par (w, min) na kraj druge trake i izbrišemo ga s kraja prve trake. Opet, odredimo rekurzivno postoji li put od x do w duljine najviše 2^{min}.

3) Ako postoji i taj put obrišemo par (w, min) s zadnje trake. Sada znamo da postoji put između x i y duljine najviše 2^{Min}.

4) Ako u bilo kojem od koraka 1) ili 2) traženi put ne postoji, tada odaberemo sljedeći po redu (leksikografski) vrh, i pokušamo s njim.

5) Ako smo iscrpili sve mogućnosti za izbor vrhova tada traženi put između x i y duljine najviše 2^{Min} ne postoji.

Po konstrukciji svaki put kada dodamo novi par na kraj reda, njegova druga koordinata bar je za jedan manja od druge koordinate para koji se nalazi iza njega. Zato svaka od traka u jednom trenutku može sadržavati najviše q parova.
Svaki par za zapis zahtijeva $O(t + \log q)$ simbola, a samim time stroj koristi $O(q(t + \log q))$ prostora. ■

Sada možemo izreći i dokazati Savitchev teorem.

Teorem 3.6 (Savitchev teorem) *Ako je* $f: \mathbf{N} \to \mathbf{N}$ *dobro izračunljiva funkcija za koju vrijedi* $f(n) \geq \log n$*, za sve* $n \in \mathbf{N}$ *tada je*

$$\mathrm{NSPACE}(f(n)) \subseteq \mathrm{DSPACE}(f(n)^2).$$

Dokaz:
Neka je $T = (k, \Sigma, \Gamma, \Phi)$ neki NTS koji odlučuje α i pritom koristi najviše $f(n)$ prostora za ulaz duljine n. Nadalje uzimamo da je n proizvoljan, ali ga fiksiramo.

Neka je graf G kao u dokazu teorema 3.5 (b)). Ideja je ista kao i prije. Za neki ulaz x treba provjeriti postoji li put od vrha v_x (početne konfiguracije s ulazom x) do nekog vrha v_0 (koji odgovara konfiguraciji stroja koja prihvača x), što odgovara postojanju postupka računanja koji prihvaća x.

Željeli bi nekako iskoristiti prethodnu lemu, odnosno Turingov stroj s prorokom. Da bi koristili proroka, trebamo se uvjeriti da je odgovor na pitanje postoji li luk između dva vrha u grafu G stvarno izvediv u jednom koraku, i što je za nas važnije u ovom dokazu, bez korištenja previše dodatnog prostora.
No, po konstrukciji grafa G, dva su vrha spojena lukom ako i samo ako konfiguracija koja odgovara drugom vrhu slijedi iz one koja odgovara prvom vrhu u samo jednom koraku koristeći relaciju Φ. Znači, dovoljno je naći $p \in \Phi$ koji prevodi jednu konfiguraciju u drugu. Kako je broj elemenata u Φ konačan, možemo je zapisati s konačnim i konstantnim brojem znakova. Dakle, za dovoljno velike n (recimo veće od duljine zapisa ϕ), stvarno možemo koristiti proroka (kao i puno puta do sada, za konačno mnogo riječi, odluku odlučivanja uvijek možemo prepustiti kontrolnoj jedinici).

Sada možemo iskoristiti prethodnu lemu za $t = q = O(f(n))$, čime jednostavno dobivamo traženu prostornu složenost od $O(f(n)^2)$. ■

Definicija 3.15 *Sa* **NPSPACE** *označavamo klasu jezika odlučivih na nedeterminističkom Turingovom stroju s polinomnom prostornom složenosti.*

Iz Savitchevog teorema slijedi jedan jako bitan rezultat koji pokazuje da uvođenjem klase NPSPACE, efektivno ništa novo nismo definirali.

Korolar 3.2 PSPACE = NPSPACE

Dokaz:
Kako je svaki DTS ujedno i NTS slijedi odmah PSPACE $\subseteq$ NPSPACE.
Obratno, kako je kvadart polinoma opet polinom, iz Savitchevog teorema dobivamo i obratnu inkluziju. ■

Uz ovu činjenicu možemo nešto novo zaključiti i o odnosu nekih drugh klasa, a i odnosu klasa P i NP.
Znamo da vrijedi P $\subseteq$ PSPACE. Kada bi vrijedila jednakost, tada bi imali

$$\text{P} = \text{PSPACE} = \text{NPSPACE} \supseteq \text{NP}$$

odnosno vrijedilo bi P=NP, što, bar za sada, nije pokazano da vrijedi. No, to su sve samo pretpostavke. Pravih dokaza (ma koliko se to jasno i opravdano činilo), još nema.
Još jedan pogled na prethodna pitanja dat će jedan poseban podskup NP klase, klasa NP-potpunih problema.

4 NP-potpunost

4.1 Uvod

Velik se dio teorije složenosti bavi upravo klasom NP. Odijeliti probleme koji su rješivi u polinomnom vremenu, od onih koji nisu, pokazao se kao vrlo zahtjevan zadatak. Tu se NP-potpunost pokazala kao važan, možda i najvažniji alat.
NP-potpunost je u biti svojstvo. Problem koji je NP-potpun, najvjerojatnije neće biti sadržan u P. S druge strane, nađe li se samo jedan NP- potpun problem sadržan u P, tada će to za posljedicu imati NP = P. Dakle, vidimo da se pitanje odnosa klasa NP i P, prebacilo na pitanje NP-potpunosti.
Za primjer NP-potpunog jezika uzet ćemo, i detaljno obraditi jezik SAT, tj. poznati Cook-Levinov teorem.

4.2 Polinomna reducibilnost i NP-potpunost

Definicija 4.1 *Kažemo da je jezik $\alpha_1 \subseteq \Sigma_1^*$ **polinomno reducibilan** na jezik $\alpha_2 \subseteq \Sigma_2^*$, ako postoji funkcija $f\colon \Sigma_1^* \to \Sigma_2^*$ izračunljiva u polinomnom vremenu, takva da za svaki $x \in \Sigma_1^*$, vrijedi:*

$$x \in \alpha_1 \text{ ako i samo ako } f(x) \in \alpha_2.$$

*Za funkciju f kažemo da je **polinomna redukcija** sa α_1 na α_2.*

Ova definicija daje efikasan način testiranja pripadnosti nekom jeziku, testiranjem pripadnosti nekom drugom, ali uz efikasnu konverziju među jezicima. Dajmo primjer polinomno reducibilnih jezika.

Primjer 4.1 *Neka je $A \subseteq \{0,1,2,3,4,5,6,7,8,9\}^*$ jezik sastavljen od svih prirodnih brojeva, te neka je $B \subseteq \{0,1\}^*$ jezik sastavljen od svih binarnih brojeva.*
Funkcija $f\colon \{0,1,2,3,4,5,6,7,8,9\}^* \to \{0,1\}^*$, dana sa $f(w) = [w]_2$, koja prirodnom broju pridružuje njegov binarni zapis je polinomno izračunljiva, pa su A i B polinomno reducibilni jezici.

Pokažimo da je relacija polinomne reducibilnosti tranzitivna.

Propozicija 4.1 *Ako je jezik $\alpha_1 \subseteq \Sigma_1^*$ polinomno reducibilan na jezik $\alpha_2 \subseteq \Sigma_2^*$, te ako je α_2 polinomno reducibilan na jezik $\alpha_3 \subseteq \Sigma_3^*$, tada je i α_1 polinomno reducibilan na α_3.*

Dokaz:
Po definiciji 4.1 slijedi da postoje polinomno izračunljive funkcije $f_1\colon \Sigma_1^* \to \Sigma_2^*$ i $f_2\colon \Sigma_2^* \to \Sigma_3^*$, takve da, za svaki $w \in \Sigma_1^*, w' \in \Sigma_2^*$, vrijedi

$$w \in \alpha_1 \iff f_1(w) \in \alpha_2$$

$$w' \in \alpha_2 \iff f_2(w') \in \alpha_3.$$

Neka je sada $w \in \Sigma_1^*$, proizvoljan. Iz prethodnog sada slijedi:

$$x \in \alpha_1 \iff f_1(x) \in \alpha_2 \iff f_2(f_1(x)) \in \alpha_3$$

odnosno

$$x \in \alpha_1 \iff (f_2 \circ f_1)(x) \in \alpha_3.$$

Da bi $f_2 \circ f_1$ bila polinomna redukcija sa α_1 na α_3, trebamo još pokazati da je izračunljiva u polinomnom vremenu.
Promotrimo DTS T, koji za ulaz $w \in \Sigma_1^*$, radi sljedeće:

1) pokrene DTS M koji računa f_1 s ulazom w.

2) pokrene DTS S koji računa f_2 s ulazom $f_1(w)$.

(M i S su odgovarajući DTS iz definicije 4.1.)
Kako su koraci 1) i 2) vremenski polinomni, slijedi da je i T vremenski polinoman.
Dakle, α_1 je polinomno reducibilan na α_3. ■

Pretpostavimo da imamo polinoman algoritam za neki jezik. Tada za svaki jezik koji je polinomno reducibilan na njega dobivamo polinoman algoritam. Upravo tu leži snaga polinomne reducibilnosti. Točnije, vrijedi sljedeća propozicija.

Propozicija 4.2 *Ako je α_2 jezik u* P (NP)*, tada je svaki jezik α_1 koji je polinomno reducibilan na njega također u* P (NP)*.*

Dokaz:
Uzmimo slučaj da je $\alpha_2 \in$ P (dokaz za NP slijedi potpuno analogno).
Neka je $\alpha_2 \subseteq \Sigma_2^*$. Neka je $\alpha_1 \subseteq \Sigma_1^*$ proizvoljan jezik polinomno reducibilan na α_2.
Neka je $f\colon \Sigma_1^* \to \Sigma_2^*$ polinomna redukcija sa α_1 na α_2.
Neka je M neki DTS koji odlučuje α_2 u polinomnom vremenu, a N neki DTS koji računa f u polinomnom vremenu.

Kako je α_1 polinomno reducibilan na α_2, za svaki $w \in \Sigma_1^*$ vrijedi:

$$w \in \alpha_1 \iff f(w) \in \alpha_2.$$

Zato je dovoljno konstruirati DTS koji će u polinomnom vremenu za ulaz w kao izlaz davati 1 ako i samo ako je $f(w) \in \alpha_2$ (0 inače).
Promotrimo DTS T koji za ulaz w radi na sljedeći način:

1) Pokrene N s ulazom w (na izlazu dobijemo $f(w)$).

2) Pokrene M s ulazom $f(w)$ (na izlazu dobijemo 0 ili 1).

3) Ako je izlaz 1, prihvati riječ w, a inače je odbaci.

Kako M odlučuje α_2, to je upravo DTS s traženim svojstvom.
Naravno, kako su M i N polinomni strojevi slijedi i da je T polinoman stroj.
Dakle, vrijedi $\alpha_1 \in$ P. ■

Definicija 4.2 *Kažemo da je jezik* α **NP-potpun**, *ako zadovoljava sljedeća dva uvjeta:*

1) $\alpha \in \text{NP}$.

2) *Svaki jezik* $\beta \in \text{NP}$ *je polinomno reducibilan na njega.*

NP-potpuni problemi, najteži su problemi unutar klase NP. Riječ potpun trebala bi asocirati da algoritamskim rješenjem nekog NP-potpunog problema dolazimo do algoritamskog rješenja svih ostalih problema. Primijetimo da ako se ikada nađe polinoman algoritam za neki NP-potpun problem, tada odmah slijedi NP = P.
Obično su jezici povezani polinomnom redukcijom pripadali istoj klasi jezika. Iznimka nisu ni NP-potpuni jezici.

Propozicija 4.3 *Neka je* α_1 NP*-potpun jezik i* $\alpha_2 \in \text{NP}$.
Ako je α_1 *polinomno reducibilan na* α_2 *tada je i* α_2 NP*-potpun.*

Dokaz:
Kako je prema pretpostavci $\alpha_2 \in \text{NP}$. Treba pokazati da je proizvoljan jezik $\beta \in \text{NP}$ polinomno reducibilan na njega.
Po pretpostavci imamo da je α_1 polinomno reducibilan na α_2. S druge strane, kako je α_1 NP-potpun, β je polinomno reducibilan na njega.
Sada smo u uvjetima propozicije 4.1, koja kaže da je relacija polinomne reducibilnosti tranzitivna. Dakle, β je polinomno reducibilan na α_2. ■

Zbog svojstva koje ima NP-potpun jezik u biti se dovoljno ograničiti na proučavanje i rješavanje jednog NP-potpunog jezika.
Međutim pitanje je: Postoje li uopće NP-potpuni jezici? Postoje, i mi ćemo dati detaljno razrađeni primjer jednog takvog jezika.

4.3 Cook-Levinov Teorem

Bitan doprinos o pitanju odnosa P i NP klasa dali su, u ranim 1970-im, radovi Stephena Cooka i Leonida Levina. Oni su i prvi otkrili NP-potpune probleme. Jedan od takvih problema je pitanje ispunjivosti konjuktivne normalne formule. Što je konjuktivna normalna forma, jezik SAT, i još neke pojmove iznijeli smo u poglavlju 3.3. Krenimo odmah na poznati Cook-Levinov teorem.

Teorem 4.1 (Cook-Levinov Teorem) *Jezik* SAT *je* NP*-potpun.*

Dokaz:
Trebamo pokazati:

a) SAT $\in$ NP.

b) Svaki jezik iz NP je polinomno reducibilan na SAT.

Dokaz:
a)
Činjenicu da je SAT element od NP, pokazali smo teoremom 3.4 u poglavlju 3.3.

b)
Preostaje pokazati da je proizvoljan jezik iz NP polinomno reducibilan na SAT.
Jedino što možemo iskoristiti je činjenica da za svaki jezik iz NP postoji vremenski polinoman NTS koji ga odlučuje.
Ideja je rad stroja opisati formulama logike sudova.

Neka je $\alpha \in$ NP proizvoljan jezik. Tada postoji NTS $T = (1, \Sigma, \Gamma, \Phi)$ koji odlučuje α u vremenu $O(n^c)$ za neku konstantu $c > 0$.
Neka je $w = h_1 \dots h_n$ proizvoljna riječ iz Σ_0^*. Tada T odlučuje pripada li w jeziku α u $N = \lceil c_1 n^c \rceil$ koraka, gdje je $c_1 > 0$ konstanta (slijedi iz definicije 1.2).
Uvedimo oznake za sljedeće propozicionalne varijable:

a) $X_{n,g}$, za $0 \leq n \leq N, g \in \Gamma$

b) $Y_{n,p}$, za $0 \leq n \leq N, -N \leq p \leq N$

c) $Z_{n,p,h}$, za $0 \leq n \leq N, -N \leq p \leq N, h \in \Sigma$.

Ogradili smo p sa $-N$ i N, jer stroj T u N koraka može doseći najviše ćeliju s indeksom N, odnosno $-N$.
Definirane propozicionalne varijable pomoći će nam u opisivanju rada stroja T. Njihova uloga bit će puno jasnija nakon što objasnimo na koji ćemo im način pridružiti vrijednost 0 ili 1.

Prateći postupak izračunavanja stroja T, varijablama vrijednosti pridružujemo ovako:

$X_{n,g} = 1$, ako se nakon n-tog koraka stroj T nalazi u stanju g

$Y_{n,p} = 1$, ako se nakon n-tog koraka, glava od T nalazi na p-toj ćeliji

$Z_{n,p,h} = 1$, ako nakon n-tog koraka p-ta ćelija sadrži znak h.

(Definiranu interpretaciju ne označavamo, tj. pišemo $p = 1$ umjesto $I(p) = 1$.)
Prema definiciji rada NTS, u svakom koraku rada stroja vrijednost svake od ovih varijabli jedinstveno je određena, tj. iz njih možemo kasnije rekonstruirati rad stroja. Obratno ne vrijedi, jer na primjer, proizvoljno pridruživanje vrijednosti može rezultirati time da se stroj u nekom koraku nalazi u dva različita stanja, što je naravno nemoguće.

Nadalje, promatrajući rad NTS, sljedeće formule neće biti teško shvatiti.

(1) $\bigvee_{g\in\Gamma} X_{n,g}$, za $0 \leq n \leq N$. Ova formula je za definirani postupak izračunavanja stroja uvijek istinita, jer se stroj u svakom koraku nalazi u jednom od stanja.

(2) $\neg X_{n,g} \vee \neg X_{n,g'}$, za $g, g' \in \Gamma, g \neq g', 0 \leq n \leq N$. Ova formula je također istinita za definirani postupak izračunavanja stroja, jer se stroj ne može istovremeno nalaziti u dva stanja.

Kako je sa START označeno početno stanje stroja, sa STOP završno stanje, te kako je na početku rada glava stroja pozicionirana na nultu ćeliju, vrijedi:

(3) $X_{0,START} = 1, X_{N,STOP} = 1, Y_{0,0} = 1$

Također, na početku rada je kao ulazni podatak zapisana riječ $w = h_1 \dots h_n$, ostatak trake je prazan, a na kraju je, ako je $w \in \alpha$, u nultoj ćeliji zapisan 1 (stroj prihvača w). Zato imamo:

(4) $Z_{0,i-1,h_i} = 1$, za $1 \leq i \leq n$

(5) $Z_{0,i,*} = 1$, za $0 < i$ i $i \geq n$

(6) $Z_{N,0,1} = 1$.

Izrazimo formulama i funkciju prijelaza stroja.
Za sve $h, h' \in \Sigma, g, g' \in \Gamma, \epsilon \in \{-1, 0, 1\}, -N \leq p \leq N$ uvedimo formule:

$$(X_{n,g} \wedge Y_{n,p} \wedge Z_{n,p,h}) \rightarrow \neg(X_{n+1,g'} \wedge Y_{n+1,p+\epsilon} \wedge Z_{n+1,p,h'})$$

što je logički ekvivalentno sa

(7) $\neg X_{n,g} \vee \neg Y_{n,p} \vee \neg Z_{n,p,h} \vee \neg X_{n+1,g'} \vee \neg Y_{n+1,p+\epsilon} \vee \neg Z_{n+1,p,h'}$.

Tamo gdje glave stroja nema, sadržaj trake se ne mijenja, pa imamo:

$$\neg Y_{n,p} \rightarrow (Z_{n,p,h} \Leftrightarrow Z_{n+1,p,h})$$

što je logički ekvivalentno sa

(8) $(Y_{n,p} \vee Z_{n,p,h} \vee \neg Z_{n+1,p,h}) \wedge (Y_{n,p} \vee \neg Z_{n,p,h} \vee Z_{n+1,p,h})$.

Označimo sa F_w formulu dobivenu konjunkcijom formula (1),...,(8). Sada dolazimo do ključnog dijela dokaza. Tvrdimo da je funkcija koja postupku izračunavanja za riječ $w \in \Sigma_0^*$, pridružuje formulu F_w, polinomna redukcija sa α na SAT.
Dokažimo prvo da vrijedi:

$$w \in \alpha \text{ ako i samo ako } F_w \in \text{SAT}.$$

Ako je $w \in \alpha$, tada T prepoznaje α, te iz opisane konstrukcije čitamo vrijednosti interpretacije na svakoj od propozicionalnih varijabli koje se pojavljuju u formuli F_w. Kako je $w \in \alpha$, tada je po opisanoj konstrukciji F_w očito istinita.
Obratno, ako je F_w istinita, tada je svaka od elementarnih disjunkcija istinita, no tada po konstrukciji slijedi da T prihvača w.

Trebamo još pokazati da je za $w \in \alpha$ formulu F_w moguće konstruirati u polinomnom vremenu.
Primijetimo da su sve formule u $(1), \ldots, (8)$ konstantne duljine. Kako formula pod (7) ima daleko najviše, dovoljno je dokazati da je njih moguće konstruirati u polinomnom vremenu.
Označimo sa br, broj različitih fromula pod (7). Tada vrijedi

$$br = \binom{|\Sigma|}{2} \cdot \binom{|\Gamma|}{2} \cdot 3 \cdot (2N+1)$$

gdje je $\binom{|\Sigma|}{2}$ broj odabira dva znaka iz Σ, $\binom{|\Gamma|}{2}$ broj odabira dva stanja iz Γ, $3 = |\{-1, 0, 1\}|$.
Kako su Σ i Γ konačni, slijedi da postoje prirodni brojevi l_1, l_2 takvi da vrijedi

$$\binom{|\Sigma|}{2} \leq n^{l_1} \text{ i } \binom{|\Gamma|}{2} \leq n^{l_2}.$$

Sada imamo da je

$$br \leq n^{l_1} \cdot n^{l_2} \cdot 3 \cdot (2N+1)$$

što je očito polinoman broj koraka u ovisnosti o $|w| = n$.
Kako konstrukcija svake formule u (7) zahtijeva konstantan broj koraka, slijedi da je sve formule u (7) moguće konstruirati u polinomnom broju koraka.
■

Sada ćemo obraditi neke specijalne slučajeve SAT problema. Sljedećih par teorema pokazat će kako i najmanje reduciranje jezika može rezultirati postojanjem neusporedivo boljeg algoritma.

Definicija 4.3 *Konjunktivna normalna forma zove se* ***k-forma*** *ako svaka od elementarnih disjunkcija sadrži točno k literala.*

Označimo sa k-SAT jezik sastavljen od svih ispunjivih k-formi.
Cilj nam je dokazati da vrijedi 2-SAT $\in$ P, ali da je već 3-SAT $\in$ NP.
Za dokaz činjenice da je 2-SAT $\in$ P trebat će nam nekoliko pomoćnih rezultata.

Lema 4.1 *Neka je B 2-forma sastavljena od propozicionalnih varijabli $x_1 \dots x_n$. Za svaki $i = 1, \dots, n$ označimo sa x_i^1 varijable x_i, a sa x_i^0 njihove negacije. Neka je G usmjereni graf sa skupom vrhova*

$$V(G) = \{x_1^1, \dots, x_n^1, x_1^0, \dots, x_n^0\}.$$

Lukovi grafa G neka su definirani na sljedeći način: vrh x_i^ϵ spojit ćemo sa vrhom x_j^δ, ako se u B pojavljuje elementarna disjunkcija $x_i^{1-\epsilon} \vee x_j^\delta$.
Tada postoji deterministički Turingov stroj koji u polinomnom vremenu odgovara na pitanje postoji li u grafu G usmjereni ciklus između neke propozicionalne varijable x_i i njene negacije.

Ideja dokaza:
Pretpostavljamo da nam je dan neki razuman način kodiranja usmjerenih grafova.
Prvo što treba napraviti je za danu 2-formu konstruirati pripadni usmjereni graf, tj. potrebno je konstruirati skup vrhova te skup lukova. Očito, oba skupa moguće je konstruirati u polinomnom (linearnom) vremenu. Skup vrhova zahtijeva samo identificiranje propozicionalnih varijabli u 2-formi, dok je egzistencija luka uvjetovana postojanjem specifične elementarne disjunkcije.

Nakon što smo konstruirali graf potražimo zadane cikluse. Pogledajmo sljedeći postupak označavanja vrhova koji ponavljamo za svaki $i = 1, \dots, n$:
Uzmimo vrh x_i^1 i označimo ga. U svakom sljedećem koraku označavamo neoznačene vrhove u koje ulazi luk iz bar jednog već označenog vrha. Stanemo kad dođemo do vrha x_i^0 ili kad više nema novooznačenih vrhova. Ako dođemo do vrha x_i^0 tada ponovimo isti postupak označavanja vrhova počevši sa njim. Označimo li vrh x_i^1 tada smo našli traženi ciklus i zaustavljamo se. Prođemo li sve propozicionalne varijable i ne nađemo dani ciklus tada se zaustavljamo jer traženog ciklusa nema.
Činjenica da je postupak označavanja polinoman odmah slijedi iz opisa samog postupka. ■

Primijetimo da je disjunkcija $x_i^{1-\epsilon} \vee x_j^{\delta}$ logički ekvivalentna implikaciji $x_i^{\epsilon} \to x_j^{\delta}$. Također, primijetimo da ako postoji luk od x_i^{ϵ} do x_j^{δ} da tada postoji i luk od $x_j^{1-\delta}$ do $x_i^{1-\epsilon}$ jer vrijedi

$$x_j^{1-(1-\delta)} \vee x_i^{1-\epsilon} \equiv x_j^{\delta} \vee x_i^{1-\epsilon} \equiv x_i^{1-\epsilon} \vee x_j^{\delta}.$$

Lema 4.2 *Neka su B i G kao u prethodnoj lemi. 2-forma B je ispunjiva ako i samo ako u grafu G ne postoji usmjereni ciklus između neke propozicionalne varijable i njene negacije.*

Dokaz:
Neka je formula B ispunjiva. Pretpostavimo suprotno, odnosno da postoji usmjereni ciklus između neke propozicionalne varijable x_i^1 i njene negacije x_i^0. Taj ciklus sastoji se od dva usmjerena puta, puta od x_i^0 do x_i^1 i puta od x_i^1 do x_i^0. Po definiciji grafa G i zbog činjenice da je disjunkcija $x_i^{1-\epsilon} \vee x_j^{\delta}$ logički ekvivalentna implikaciji $x_i^{\epsilon} \to x_j^{\delta}$ slijedi da postoje sljedeća dva niza implikacija:

1) $P^{1,0} \equiv x_i^1 \to x_{i_1} \to \cdots \to x_{i_r} \to x_i^0$ (put od x_i^1 do x_i^0)

2) $P^{0,1} \equiv x_i^0 \to x_{j_1} \to \cdots \to x_{j_s} \to x_i^1$ (put od x_i^0 do x_i^1).

Kako je formula B ispunjiva, slijedi da postoji interpretacija I za koju je $I(B) = 1$. Znamo, moralo bi vrijediti ili $I(x_i^1) = 1$ i $I(x_i^0) = 0$ ili $I(x_i^1) = 0$ i $I(x_i^0) = 1$. Sada jednostavno dolazimo do kontradikcije.
Bez smanjenja općenitosti neka vrijedi $I(x_i^1) = 1$ i $I(x_i^0) = 0$.
Kako je $I(x_i^1) = 1$ tada zbog činjenice da postoji luk iz x_i^1 u x_{i_1}, mora vrijediti i $I(x_{i_1}) = 1$ jer inače elementarna disjunkcija $x_i^0 \vee x_{i_1} \equiv x_i^1 \to x_{i_1}$ ne bi bila ispunjena pa ni formula B ne bi bila istinita za I što je kontradikcija sa pretpostavkom da je $I(B) = 1$. Dakle, $I(x_{i_1}) = 1$. No sada promatrajući x_{i_1} i x_{i_2} zbog istih argumenata imamo da mora vrijediti i $I(x_{i_2}) = 1$. Analogno dalje dolazimo do zaključka da svaka propozicionalna varijabla koja je dio puta $P^{1,0}$ mora biti istinita. Specijalno bi tada vrijedilo i $I(x_i^0) = 1$ što je kontradikcija jer smo krenuli sa pretpostavkom da je $I(x_i^0) = 0$.
Slučaj kada vrijedi $I(x_i^1) = 0$ i $I(x_i^0) = 1$ potpuno je analogan samo moramo promatrati put $P^{0,1}$.

Obratno, neka u grafu G ne postoji usmjereni ciklus između neke propozicionalne varijable i njene negacije. Tada ne postoji varijabla x_i takva da vrijedi: postoji put od x_i^1 do x_i^0 i postoji put od x_i^0 do x_i^1. Odaberimo, ako postoji, takvu propozicionalnu varijablu x_i za koju ne postoji ni put od x_i^1 do x_i^0 ni put od x_i^0 do x_i^1. Pogledajmo sljedeći postupak:
Dodajmo u graf G luk od x_i^1 do x_i^0. Time naravno nismo mogli pokvariti

pretpostavku da ne postoji ciklus koj sadrži varijablu i njenu negaciju (dodavanjem jednog luka ne možemo stvoriti dva različita puta između krajeva tog luka, ako ih prije nije bilo).
Ponavljanjem ovog postupka u graf G dodajemo nove lukove, uvijek od varijable prema njenoj negaciji. Na kraju dobijemo graf u kojem između svake varijable i njene negacije postoji točno jedan put i to u točno jednom smjeru (ili put od propozicionalne varijable prema njenoj negaciji ili put od negacije propozicionalne varijable prema toj propozicionalnoj varijabli).

Promotrimo interpretaciju I koja je na propoziconalnim varijablama iz B ovako definirana:

$I(x_i) = 1$ ako usmjereni put vodi od x_i^0 do x_i^1

$I(x_i) = 0$ ako usmjereni put vodi od x_i^1 do x_i^0

Interpretacija I neka je na ostalim propozicionalnim varijablama proizvoljno definirana. Tvrdimo da za ovako definiranu interpretaciju I vrijedi $I(B) = 1$.
Pretpostavimo suprotno odnosno da je $I(B) = 0$.
Tada je barem jedna od elementarnih disjunkcija lažna. Uzmimo jednu takvu elementarnu disjunkciju i označimo je sa ED. Nadalje, označimo propozicionalne varijable koje se pojavljuju u ED sa x_i i x_j. Elementarna disjunkcija ED može ovako izgledati:

a) $\text{ED} \equiv x_i^1 \vee x_j^1$

b) $\text{ED} \equiv x_i^0 \vee x_j^1$

c) $\text{ED} \equiv x_i^1 \vee x_j^0$

d) $\text{ED} \equiv x_i^0 \vee x_j^0$

Za ilustraciju uzmimo slučaj pod *a)* (ostali slučajevi su analogni):
Kako je $I(x_i^1) = 0$ i $I(x_j^1) = 0$, iz načina na koji smo definirali interpretaciju I slijedi da postoji usmjereni put od x_i^1 do x_i^0, označimo ga sa $P_i^{1,0}$, i usmjereni put od x_j^1 do x_j^0, označimo ga sa $P_j^{1,0}$. No, zbog postojanja elementarne disjunkcije ED te načina na koji smo povlačili lukove prilikom definiranja grafa G slijedi da postoji luk iz x_i^0 u x_j^1 te da postoji luk iz x_j^0 u x_i^1. No sada imamo luk iz x_i^0 u x_j^1, put $P_j^{1,0}$ te luk iz x_j^0 u x_i^1 što zajedno daje put od x_i^0 do x_i^1. Sa druge strane već znamo da postoji put od x_i^1 do x_i^0 ($P_i^{1,0}$). Dakle, imamo i put od x_i^0 do x_i^1 i put od x_i^1 do x_i^0 što znači da postoji usmjereni ciklus između x_i^0 i x_i^1 u grafu G. No, time smo dobili kontradikciju zbog pretpostavke da ne postoji usmjereni ciklus između propozicionalne varijable i njene negacije te činjenice da opisani postupak ubacivanja novih lukova to

svojstvo čuva. ■

Pomoću prethodne dvije leme lako dokazujemo sljedeći teorem.

Teorem 4.2 *2-SAT* $\in P$

Dokaz:
Neka je dana proizvoljna 2-forma B. Treba provjeriti je li ona ispunjiva. Označimo sa G graf iz leme 4.1. Prema lemi 4.2 dovoljno je pokazati da u grafu G ne postoji usmjereni ciklus između neke propozicionalne varijable i njene negacije. Kako je po lemi 4.1 to moguće provjeriti u polinomnom vremenu, tvrdnja teorem slijedi. ■

Iako smo upravo vidjeli da je 2-SAT ∈ P već 3-SAT pokazat će se kao puno složeniji jezik.

Teorem 4.3 *3-SAT je NP-potpun jezik.*

Dokaz:
Trebamo pokazati:

a) 3-SAT ∈ NP

b) svaki jezik $\beta \in$ NP je polinomno reducibilan na 3-SAT.

Pokažimo prvo *b*).
Po Cook-Levinovom teoremu znamo da je svaki jezik $\beta \in$ NP polinomno reducibilan na SAT. Pokažemo li da je SAT polinomno reducibilan na 3-SAT tada će prema propoziciji 4.1 slijediti tražena tvrdnja.

Pogledajmo prvo kako elementarnu disjunkciju pretvoriti u logički ekvivalentnu 3-formu. Neka je $C \equiv L_1 \vee \ldots \vee L_k$ elementarna disjunkcija. Sa V_i ćemo označiti novo uvedene varijable. U ovisnosti o k definiramo:

- za $k = 1$ je $C \equiv L_1$ pa je tada 3-forma dana sa:
  $C' \equiv (L_1 \vee V_1 \vee V_2) \wedge (L_1 \vee \neg V_1 \vee V_2) \wedge (L_1 \vee V_1 \vee \neg V_2) \wedge (L_1 \vee \neg V_1 \vee \neg V_2)$
- za $k = 2$ je $C \equiv L_1 \vee L_2$ te je 3-forma:
  $C' \equiv (L_1 \vee L_2 \vee V_1) \wedge (L_1 \vee L_2 \vee \neg V_1)$
- za $k = 3$ je upravo C tražena 3-forma
- za $k > 3$, 3-forma izgleda ovako:
  $C' \equiv (L_1 \vee L_2 \vee V_1) \wedge (\bigwedge_{i=1}^{k-4} \neg V_i \vee L_{i+2} \vee V_{i+1}) \wedge (\neg V_{k-3} \vee L_{k-1} \vee L_k)$

Ostaje pokazati kako proizvoljnu konjunktivnu normalnu formu pretvoriti u logički ekvivalentnu 3-formu. Označimo sa $CF \equiv C_1 \wedge \ldots \wedge C_h$ proizvoljnu konjunktivnu normalnu formu čije su elementarne disjunkcije $C_1, \ldots, C_h$. Sada za svaku od elementarnih disjunkcija $C_1, \ldots, C_h$ konstruiramo logički ekvivalentnu 3-formu $C_1', \ldots, C_h'$ prema opisanom postupku. Radi lakše konstrukcije možemo uzeti da su skupovi novo uvedenih varijabli za svaku od elementarnih disjunkcija disjunktni.
Definiramo formulu $CF' \equiv C_1' \wedge \ldots \wedge C_h'$. Formula CF' je tražena 3-forma.

Vidimo da u sva četiri slučaja (ovisno o k) konstrukcija nove 3-forme ovisi linerano o broju literala u njoj. Kako u proizvoljnoj konjunktivnoj normalnoj formi w ne može biti više od $m = |w|$ elementarnih disjunkcija slijedi da ovaj postupak zahtijeva najviše $O(m^2)$ koraka.

Činjenicu da je 3-SAT $\in$ NP nećemo detaljno diskutirati (dokaz se može naći u [4]). Primijetimo samo da kad bi vrijedilo 3-SAT $\in$ P tada bi po dokazanom pod *b*) imali P = NP. Ipak, do sada se nije pokazalo da je 3-SAT $\in$ P. ■

Iz dokaza prethodnog teorema te propozicije 4.2 lako dobivamo:

Korolar 4.1 *Za $k \geq 3$ je k-SAT $\in$ NP.*

4.4 Zaključak

Ovaj je rad zamišljen tako da prezentira osnove teorije složenosti, kao jedne od temeljnih disciplina modernog teorijskog računarstva. Samim time naglasak je stavljen na najpoznatije i najvažnije klase složenosti te samo na dva najčešće razmatrana resursa, vrijeme i prostor.

U prvom poglavlju detaljno su obrađeni pojmovi na kojima se bazira teorija složenosti. Tu je svakako bilo najvažnije uvođenje i odabir Turingovog stroja kao računalnog modela.
Poglavljem **DTIME i DSPACE** uveli smo najbitnije deterministične klase složenosti: DTIME, DSPACE, P, PSPACE. Posebnu pozornost posvetili smo klasi P kao i primjerima problema koji u nju spadaju. Od teorema obrađenih u ovom poglavlju posebno valja istaknuti Speed-up teoreme koji govore o mogučnostima ubrzanja računanja na Turingovom stroju za pojedine probleme.
Trećim poglavljem uveli smo u igru nedeterminizam te nedeterminističke analogone klasa definiranih u prethodnom poglavlju. Središnje mjesto zauzela je klasa NP kao i neki primjeri problema u toj klasi. Savitchev teorem pokazao nam je da nedeterminizmom (nasuprot determinizma) ne možemo uštedjeti puno prostora te je i bio najzvučniji rezultat u trećem poglavlju.
U zadnjem poglavlju detaljnije smo obradili klasu NP kroz NP-potpune probleme. Cook-Levinov teorem trebao je biti svojevrsan vrhunac ovog rada, teorem koji bi zaokružio do tada napravljeno, a i potaknuo na daljnje proučavanje.

U literaturi se često, zbog njihove važnosti, pojavljuju i posebni nazivi za neke klase koje nismo obradili, kao na primjer: LOGTIME i LOGSPACE tj. klase jezika odlučivih u logaritamskom vremenu odnosno prostoru na determinističkom Turingovom stroju te EXP i EXPSPACE tj. klase jezika odlučivih u eksponencijalnom vremenu odnosno prostoru na determinističkom Turingovom stroju (NEXP i NEXPSPACE su nedeterministički analogoni).
Spomenimo još i klasu $co-$NP, koju nismo obradili, ali se također često spominje u literaturi. Kažemo da je jezik u $co-$NP ako i samo ako je njegov komplement u NP (Klasa $co-$NEXP se slično definira. Zainteresirane upućujemo na knjige [4] i [5]).
Iz ovih par, netom uvedenih klasa, vidljivo je da ima još dosta toga što ovaj rad nije dotakao. Također, Turingov stroj i njemu ekvivalentni računalni modeli spadaju u uglavnom sekvencijalne modele pa se postavlja pitanje recimo, dobija li se što uvođenjem paralelnih računalnih modela. Ili možda, postoji li način karakterizacije klasa složenosti koji bi omogučio bolji uvid u problematiku? Deskriptivna teorija složenosti jedan je od takvih načina. Dok

klasična teorija složenosti probleme klasificira na temelju resursa potrebnih za njihovo rješavanje, deskriptivna teorija složenosti nastoji klase složenosti obuhvatiti odgovarajućim logikama. Na taj način je za dokazivanje novih činjenica moguće iskoristiti već poznate rezultate iz korištenih logika (puno više o deskriptivnoj teoriji složenosti može se naći u [2]).
Prostora za daljnje proučavanje ima te se nadam da će netko ovim radom biti motiviran za to.

Literatura

[1] S. Arora, *Computational Complexity: A Modern Approach*, skripta, http://www.cs.princeton.edu/∼arora

[2] M. Botinčan, *Deskriptivna teorija složenosti: Verifikacija modela*, Magistarski rad, Zagreb, 2005.

[3] P. Gacs, L. Lovasz, *Complexity of Algorithms*, skripta, Spring, 1999., http://research.microsoft.com/users/lovasz/notes.htm

[4] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley Publishing Company, 1994.

[5] M. Sipser, *Introduction to the theory of computation*, PWS Publishing Company, Boston, 1997.

[6] D. Veljan, *Kombinatorna i diskretna matematika*, Algoritam, Zagreb, 2001.

[7] M. Vuković, *Matematička logika 1*, PMF-Matematički odjel, Zagreb, 2004.