UNIVERZITET U TUZLI
PRIRODNO-MATEMATIČKI FAKULTET

Elvis Baraković

Kompleksnost i izračunljivost

- Skripta -

Tuzla, 2022.

Sadržaj

1 Vremenska kompleksnost

U ovom dijelu predstavljamo teoriju kompleksnosti, tj. istraživanje vremena, memorije ili drugih izvora koji su potrebni za rješavanje izračunljivih procesa. Počet ćemo sa vremenom. Također, cilj nam je predstaviti osnove vremenske teorije kompleksnosti gdje ćemo prvo predstaviti načine mjerenja vremena koje je potrebno da se riješi problem. Zatim ćemo pokazati kako se klasificiraju problemi prema količini zahtijevanog vremena. Nakon toga ćemo diskutovati o mogućnosti da određeni odlučiv problem zahtijeva preveliku količinu vremena i kako odrediti kada ste suočeni sa takvim problemom.

1.1 Mjerenje kompleksnosti

Posmatrajmo jezik $A = \{0^k 1^k | k \geqslant 0\}$, koji je očigledno odlučiv jezik. Postavlja se pitanje, koliko vremena je potrebno Turingovoj mašini sa jednom trakom da odluči jezik A? Zbog toga, ispitujemo sljedeću Turingovu mašinu sa jednom trakom, koju ćemo označiti sa M_1, za jezik A. Ovdje ćemo dati jednostavan opis Turingove mašine, uključujući i trenutno kretanje glave, tako da možemo izbrojati broj koraka mašine M_1 kada se pokrene.

M_1= "Na ulaznom stringu w:

1. Skeniraj duž trake i *odbij* ako je desno od 1 pronađena 0.
2. Ponavljaj ako oboje, i nule i jedinice, ostaju na traci:
3. Skeniraj duž trake, prekrižavanjem jedne nule i jedne jedinice.
4. Ako su ostale 0 nakon što su sve jedinice prekrižene, ili ako su ostale jedinice nakon što su sve nule prekrižene, *odbij*. U suprotnom, ako nije ostala nijedna nula niti ijedna jedinica, *prihvati*."

Analiziramo algoritam za Turingovu mašinu M_1 koja odlučuje jezik A određivanjem koliko vremena mašina koristi.

Broj koraka koje algoritam koristi za određeni ulaz može da ovisi od nekoliko parametara. Naprimjer, ako je ulaz graf, broj koraka može da ovisi od broja čvorova, broja ivica i maksimalnog stepena grafa, ili od neke kombinacije ovih i/ili drugih faktora. Zbog jednostavnosti, mi izračunavamo vrijeme algoritma čisto kao funkciju od dužine stringa koji predstavlja ulaz, i ne posmatramo nikakve druge parametre. U najgorem slučaju analize, posmatramo najduže vrijeme izvršenja svih ulaza određene dužine, dok u prosječnom slučaju, posmatramo prosjek svih vremena izvršenja ulaza određene dužine.

Definicija 1.1.1. *Neka je M deterministička Turingova mašina koja se zaustavlja na svim ulazima. Vrijeme izvršenja ili vremenska kompleksnost mašine M je funkcija $f : \mathbb{N} \to \mathbb{N}$, gdje je $f(n)$ maksimalan broj koraka koje mašina M koristi na ulazu dužine n. Ako je $f(n)$ vrijeme izvršenja mašine M, kažemo da se mašina M pokreće u vremenu $f(n)$ i da je mašina M ustvari $f(n)$ vremenska Turingova mašina. Uobičajeno sa n predstavljamo dužinu ulaza.*

Budući da je egzaktno vrijeme izvršenja algoritma često veoma složen izraz, mi se obično koristimo procjenama. Asimptotska analiza je jedan od uobičajenih formi za procjenu gdje tražimo da razumijemo vrijeme izvršenja algoritma koji se izvršava na velikim ulazima. To radimo posmatrajući samo najveći stepen u izrazu, zanemarujući koeficijent uz njega kao i manje stepene od njega, jer najveći stepen dominira nad svima njima.

Prisjetimo se definicija za asimptotske oznake O i o.

Definicija 1.1.2 (Veliko O notacija)**.** *Posmatrajmo funkcije $f, g : \mathbb{N} \to \mathbb{R}^+$. Kažemo da je $f = O(g)$ ako postoje pozitivni cijeli brojevi c i n_0 takvi da za svaki cio broj $n \geqslant n_0$ vrijedi $f(n) \leqslant cg(n)$.*

Kada je $f = O(g)$ kažemo da je $g(n)$ gornja granica za $f(n)$, ili preciznije da je $g(n)$ asimptotska gornja granica za $f(n)$ time naglašavajući da izbacujemo konstantne faktore. Intuitivno, $f = O(g)$ znači da je f manje ili jednako od g ako zanemarimo razlike do konstantnog faktora.

Primjer 1.1. *Posmatrajmo funkciju $f_1(n) = 5n^3 + 2n^2 + 22n + 6$, tada je $f_1(n) = O(n^3)$, jer je najveći stepen, bez koeficijenta n^3. Zaista, ako odaberemo $c = 6$ i $n_0 = 10$, dobijamo da je $5n^3 + 2n^2 + 22n + 6 \leqslant 6n^3$, za svako $n \geqslant 10$. Također, vrijedi $f_1(n) = O(n^4)$ ali ne vrijedi $f_1(n) = O(n^2)$.* ∎

Primjer 1.2. *Veliko O sa logaritmima djeluje na određeni način. Obično kada koristimo logaritme, moramo naglasiti bazu logaritma, kao naprimjer $x = \log_2 n$, što je ekvivalentno sa $2^x = n$. Mijenjanjem baze b vrijednost logaritma $\log_b n$ se mijenja do na konstatni faktor jer je $\log_b n = \frac{\log_2 n}{\log_2 b}$. Zbog toga, kada pišemo $f(n) = O(\log n)$, naglašavamo da baza više nije potrebna jer svakako izbacujemo konstatne faktore.*

Neka je $f_2(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2$. U ovom slučaju imamo da je $f_2(n) = O(n \log n)$ jer $\log n$ dominira nad $\log \log n$. ∎

Ukoliko imamo, naprimjer da je $f(n) = O(n^2) + O(n)$, tada je $f(n) = O(n^2)$ jer izraz $O(n^2)$ dominira nad $O(n)$. Ista je situacija kada se $O(n)$ pojavljuje kao eksponent, kao naprimjer $f(n) = 2^{O(n)}$. Ovaj izraz predstavlja gornju granicu od 2^{cn}, za neku konstantu c.

Izraz $f(n) = 2^{O(n \log n)}$ se također pojavljuje u nekim analizama. Koristeći identitet $n = 2^{\log_2 n}$, dobijamo $n^c = 2^{c \log_2 n}$, odakle zaključujemo da $2^{O(\log n)}$ predstavlja gornju granicu za n^c za neku konstatu c. Izraz $n^{O(1)}$ predstavlja istu granicu na drugačiji način, jer izraz $O(1)$ predstavlja vrijednost koja nikad nije veća od fiksne konstante.

Veoma često izvodimo granice u obliku n^c, za $c > 0$. Takve granice se nazivaju polinomijalne granice. Granice oblika $2^{(n^\delta)}$ se nazivaju eksponencijalne granice kada je $\delta > 0$.

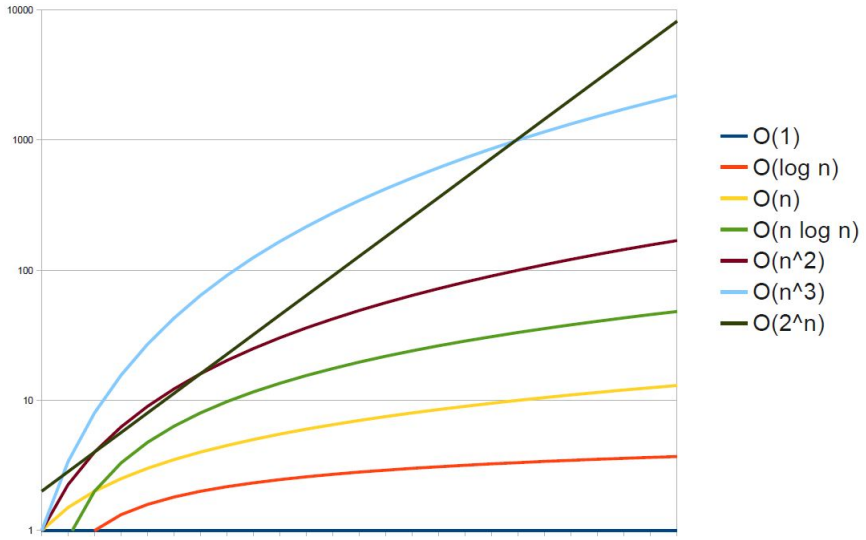
Definicija 1.1.3 (Malo o notacija)**.** *Posmatrajmo funkcije* $f, g : \mathbb{N} \to \mathbb{R}^+$*. Kažemo da je* $f = o(g)$ *ako vrijedi* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Drugim riječima, $f = o(g)$ znači da za proizvoljan realan broj $c > 0$ i proizoljno n_0 vrijedi $f(n) < cg(n)$ za sve $n \geqslant n_0$.

Primjer 1.3. *Lako je provjeriti da vrijedi:*

1. $\sqrt{n} = o(n)$
2. $n = o(n \log\log n)$.
3. $n \log\log n = o(n \log n)$.
4. $n \log n = o(n^2)$.
5. $n^2 = o(n^3)$.

Također, nikada ne vrijedi $f(n) = o(f(n))$. ∎



Da bismo analizirali mašinu M_1, posmatrat ćemo svaki korak odvojeno.

U prvom koraku, mašina skenira duž trake da bi verifikovala da je ulaz oblika 0^*1^*. Ovo mašina izvede u n koraka. Kao što smo ranije naglasili, sa n označavamo dužinu trake. Ponovnim pozicioniranjem glave na lijevi kraj trake, izvodi još n koraka. Dakle, ukupno u ovom koraku izvodi $2n$ koraka ili to možemo zapisati kao $O(n)$ koraka. Primijetimo da u opisu mašine nismo opisali ponovno pozicioniranje glave. Korištenje asimptotske notacije nam omogućava da izostavimo detalje u opisu mašine koje utiču na vrijeme izvršenja do najviše konstantan faktor.

U drugom i trećem koraku, mašina ponavljanjem skenira traku i prekrižava nule i jedinice u svakom skeniranju. Svako skeniranje koristi $O(n)$ koraka. Budući da svako

skeniranje prekrižava dva simbola, tada se desi najviše $n/2$ skeniranja. Dakle, ukupno vrijeme u drugo i trećem koraku iznosi $n/2O(n) = O(n^2)$ koraka.

U četvrtom koraku mašina uradi samo jedno skeniranje da bi odlučila da li prihvata ili ne, pa je vrijeme u ovom koraku najviše $O(n)$.

Dakle, ukupno vrijeme mašine M_1 na ulazu dužine n je $O(n)+O(n^2)+O(n)$ tj. $O(n^2)$.

Dalje ćemo dati definiciju za klasificiranje jezika prema njihovim vremenskim zahtjevima.

Definicija 1.1.4. *Neka je $t : \mathbb{N} \to \mathbb{R}^+$ proizvoljna funkcija. Klasa vremenske kompleksnosti u oznaci $TIME(t(n))$ predstavlja kolekciju svih jezika koji su od Turingove mašine odlučivi u $O(t(n))$ vremenu.*

Prethodna analiza je pokazala da za jezik $A = \{0^k 1^k | k \geqslant 0\}$ vrijedi $A \in TIME(n^2)$, jer mašina M_1 odlučuje jezik A u vremenu $O(n^2)$ i $TIME(n^2)$ sadrži sve jezike koji mogu biti odlučeni u $O(n^2)$ vremenu.

Postavlja se pitanje postoji li mašina koja će odlučiti jezik A brže nego mašina M_1. Drugim riječima, da li je moguće $A \in TIME(o(n^2))$?

Mi možemo poboljšati vrijeme izvršenja prekrižavanjem po dvije nule i dvije jedinice u svakom skeniranju, umjesto po jednu. Radeći tako, smanjujemo broj skeniranja za pola. Ali to poboljšava vrijeme izvršenja samo za faktor 2 i ne utiče na asimptotsko vrijeme izvršenja.

Sljedeća mašina M_2 koristi drugačiju metodu da odluči jezik A asimptotski brže. Pokazuje se da je za ovu mašinu $A \in TIME(n \log n)$. Mašina je data sa

M_2= "Na ulaznom stringu w:

1. Skeniraj duž trake i *odbij* ako je desno od 1 pronađena 0.
2. Ponavljaj sve dok oboje, i nule i jedinice, ostaju na traci:
3. Skeniraj duž trake, provjeravajući da li je ukupan broj nula i jedinica paran ili neparan. Ako je neparan, *odbij*.
4. Skeniraj ponovo duž trake prekrižavanjem svake druge (preskočiti jednu!) nule počevši sa 0 i prekrižavanjem svake druge (preskočiti jednu!) jedinice počevši sa 1.
5. Ako nema više nula ni jedinica na traci, *prihvati*. U suprotnom, *odbij*."

Prije analize mašine M_2, provjerimo da li doista odlučuje jezik A. Na svakom skeniranju koje se izvodi u četvrtom koraku, ukupan broj preostalih nula je upolovljen i sve ostalo se odbacuje. Tako, ako krenemo recimo sa 13 nula, poslije jednog izvršenja četvrtog koraka preostaje samo 6 nula, poslije drugog izvršenja 3 a onda 0 nula. Istu situaciju imamo i sa jedinicama.

Analizirajmo sada jednakost broja nula i jedinica u izvršenju trećeg koraka. Posmatrajmo ponovo slučaj kada imamo 13 nula i 13 jedinica. Prvo izvršenje u trećoj fazi pronalazi neparan broj nula (13 je neparan broj) i neparan broj jedinica. Nakon toga, izvršenje trećeg koraka prolazi paran broj (6), pa neparan broj (3) i na kraju neparan broj (1). Ovaj korak ne izvršavamo na 0 nula i 0 jedinica jer je to određeno u

drugom koraku. U nizu jednakosti koje smo pronašli (neparan, paran, neparan, neparan), ako paran zamijenimo sa 0 a neparan sa 1 te obrnemo niz, dobijamo binarni zapis broja 13,odnosno broj nula i jedinica na početku. Niz jednakosti uvijek daje, u obrnutom redoslijedu, binarnu reprezentaciju.

Da bi analizirali vrijeme mašine M_2, uočimo da je vrijeme izvršenja svakog koraka $O(n)$. zatim ćemo odrediti koliko je puta svaki korak izvršen. Koraci 1 i 5 su izvršeni samo jednom, što je ukupno $O(n)$ vremena. Korak 4 prekrižava najmanje pola nula i pola jedinica svaki put kad se izvrši, pa se zbog toga najviše $1+\log_2 n$ iteracija u petlji dešava prije nego se sve prekriže. Prema tome, ukupno vrijeme koraka 2,3 i 4 je $(1+\log_2 n)O(n)$ ili $O(n \log n)$. Ukupno vrijeme mašine M_2 je $O(n \log n)$.

Ranije smo pokazali da je $A \in TIME(n^2)$, a sada smo pokazali da je $A \in TIME(n \log n)$. Ovaj rezultat se ne može poboljšati dalje s Turingovom mašinom sa jednom trakom. U stvari, jezik koji se može odlučiti u $O(n \log n)$ vremenu na Turingovoj mašini sa jednom trakom, je regularan.

Ako Turingova mašina ima i drugu traku, tada se jezik A može odlučiti u $O(n)$ vremenu (često se kaže u *linearnom vremenu*!). Sljedeća Turingova mašina sa dvije trake to radi. Ona radi drugačije nego prethodne dvije, ona jednostavno kopira nule na svoju drugu traku i onda ih uspoređuje sa jedinicama.

Mašina je data sa

M_3= "Na ulaznom stringu w:

1. Skeniraj duž trake i *odbij* ako je desno od 1 pronađena 0.
2. Skeniraj duž nula na prvoj traci sve do prve jedinice. Istovremeno, kopiraj nule na drugu traku.
3. Skeniraj duž jedinica na prvoj traci sve do kraja ulaza. Za svaku pročitanu jedinicu na prvoj traci, prekriži nula na drugoj traci. Ako su sve nule prekrižene prije nego su sve jedinice očitane, *odbij*.
4. Ako u sve nule prekrižene, *prihvati*. Ako je preostala nula, *odbij*."

Ovu mašinu je lako analizirati. Svaka od 4 faze koristi $O(n)$ koraka, pa je ukupno vrijeme $O(n)$. Dakle, vrijeme je linearno. Primijetimo da je ukupno vrijeme izvršenja najbolje vrijeme, jer je samo za čitanje ulaza potrebno n koraka.

Iz svega ovoga, možemo zaključiti da kompleksnost jezika A ovisi od modela izračunavanja kojeg smo koristili. Ovo opravdava činjenicu da je veoma bitno posebno posmatrati teoriju kompleksnosti od teorije izračunljivosti. U teoriji izračunljivosti, Church-Turingova teza implicira da su svi razumni modeli izračunavanja ekvivaloentni, tj. svi oni odlučuju istu klasu jezika. U teoriji kompleksnosti, izbor modela utiče na vremensku kompleksnost jezika. Jezici koji su izračunljivi u jednom modelu recimo u linearnom vremenu, nisu takvi u nekom drugom modelu.

U teoriji kompleksnosti, izračunljive probleme klasificiramo prema njihovoj vremenskoj kompleksnosti. Ali se može postaviti pitanje, sa kojim modelom mjerimo vrijeme? Isti jezik može imati različite zahtijeve za vremena izvršenja sa različitim modelima.

Na svu sreću, vremenski zahtjevi se ne razlikuju previše za tipične deterministčke modele. Tako, ako naš sistem klasificiranja nije previše osjetljiv na relčativno male razlike

u kompleksnosti, izbor deterministčkog modela i nije ključan. O tome ćemo više u narednim sekcijama.

1.1.1 Veze izračunljivosti među modelima

U ovom dijelu proučavamo kako izbor izračunljivog modela utiče na vremensku kompleksnost jezika. Posmatramo tri modela: Turingovu mašinu sa jednom trakom, Turingovu mašinu sa više traka i nedeterminističku Turingovu mašinu.

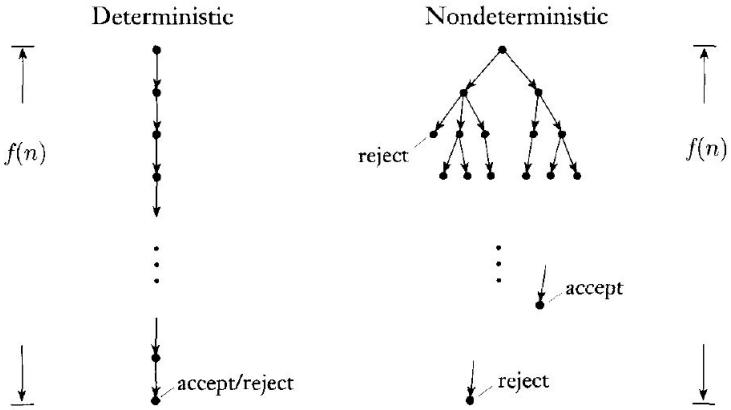
Teorem 1.1.1. *Neka je $t(n)$ funkcija, pri čemu je $t(n) \geqslant n$. Tada svaka $t(n)$ Turingova mašina sa više traka ima ekvivalentnu $O(t^2(n))$ vremensku Turingovu mašinu sa jednom trakom.*

Dokaz ove teoreme se može naći u [2].

Dalje ćemo posmatrati analogni teorem za nedeterminističke Turingove mašine sa jednom trakom. Pokazat ćemo da svaki jezik koji je odlučiv na takvoj mašini je odlučiv i na deterministčkoj Turingovoj mašini sa jednom trakom, koja zahtijeva mnogo više vremena. Prije toga, moramo definisati vrijeme izvođenja za nederminističku Turingovu mašinu. Prisjetimo se da nedeterministička Turingova mašina odlučuje ako se sve njene grane izračunavanja zaustavljaju na svim ulazima.

Definicija 1.1.5. *Neka je N nedeterministička Turingova mašina koja odlučuje. Vrijeme izvođenja mašine N je funkcija $f : \mathbb{N} \to \mathbb{N}$, gdje je $f(n)$ maksimalan broj koraka koje mašina koristi na nekoj grani svog izračunavanja na bilo kojem ulazu dužine n.*

Ovu definiciju možemo predstaviti i sljedećom slikom:



Definicija izvršenja vremena nedeterminističke Turingove mašine nije namijenjena da odgovara bilo kojem uređaju za izračunavanje u realnom životu. To je više matematička definicija koj nam pomaže u karakterizaciji veoma važne klase izračunljivih problema.

Teorem 1.1.2. *Neka je $t(n)$ funkcija, pri čemu je $t(n) \geqslant n$. Tada svaka $t(n)$ nedeterministička Turingova mašina sa jednom trakom ima ekvivalentnu $2^{O(t(n))}$ determinističku Turingovu mašinu sa jednom trakom..*

Dokaz ove teoreme se isto može naći u [2].

1.2 P klasa

U prethodnom izlaganju smo pokazali veoma bitnu podjelu algoritama. U jednu ruku, pokazali smo najviše kvadratnu ili polinomsku razliku između vremenske kompleksnosti problema mjerenih deterministčkom Turingovom mašinom sa jednom trakom i sa više traka. U drugu ruku, pokazali smo da je najviše eksponencijalna razlika između vremenske kompeksnosti problema na deterministčkim i nedeterministčkim Turingovim mašinama.

1.2.1 Polinomijalno vrijeme

Za naše svrhe, polinomijalne razlike u vremenima izvršenja se smtraju malim dok se eksponencijalne razike smatraju velikim. Razmotrimo sada zašto biramo da razdvajamo između polinomijalnih i eksponencijalnih radije nego između nekih drugih klasa funkcija.

Prije svega uočimo dramatičnu razliku između brzine rasta tipičnih polinomijalnih kao naprimjer n^3 i tipičnih eksponencijalnih kao naprimjer 2^n. Naprimjer, neka je $n = 1000$ veličina ulaza algoritma. U tom slučaju, n^3 je milijarda, koj je velik ali ipak pogodan broj, dok je 2^n broj koji je veći od broja svih atoma u svemiru. Polinomijalni algoritmi su dovoljno brzi za mnoge svrhe dok se eksponencijalni algoritmi veoma rijetko koriste.

Size	1	lg n	n	n log n	n^2	n^3	2^n
100	1µs	7µs	100µs	0.7ms	10ms	<1min	40 quadrillion yrs
200	1µs	8µs	200µs	1.5ms	40ms	<1min	More than that
300	1µs	8µs	300µs	2.5ms	90ms	1min	
400	1µs	9µs	400µs	3.5ms	160ms	2min	
500	1µs	9µs	500µs	4.5ms	250ms	4min	
600	1µs	9µs	600µs	5.5ms	360ms	6min	
700	1µs	9µs	700µs	6.6ms	490ms	9min	
800	1µs	10µs	800µs	7.7ms	640ms	12min	
900	1µs	10µs	900µs	8.8ms	810ms	17min	
1000	1µs	10µs	1000µs	10ms	1000ms	22min	
1100	1µs	10µs	1100µs	11ms	1200ms	29min	
1200	1µs	10µs	1200µs	12ms	1400ms	37min	
1300	1µs	10µs	1300µs	13ms	1700ms	45min	
1400	1µs	10µs	1400µs	15ms	2000ms	56min	

Eksponencijalni algoritmi se obično javljaju kada rješavamo probleme *sirovom snagom (brute force)*, tojest iscrpnim pretraživanjem u prostoru rješenja. Naprimjer, jedan od načina faktorizacije broja na njegove proste faktore je pretraživanje kroz sve potencijalne djelioce. Veličina prostora u kojem pretražujemo je eksponencijalna, pa je vrijeme pretraživanja eksponencijalno. Ponekad, sirova snaga se može zaobići kroz bolje i dublje shvatanje problema, što može otkriti vremenski polinomijalan algoritam koji nam je od veće koristi.

Svi razumni deterministički izračunljivi modeli su polinomijalno ekvivalentni. To znači, da jedan može da simulira drugi algoritam sa polinomijalnim povećanjem vremena. Kada kažemo da su svi razumni deterministički modeli polinomijalno ekvivalentni, ne ulazimo u definisanje pojma razumnosti modela. Međutim, imamo na umu pojam dovoljno širok da uključi modele koji su imaju približno vrijeme izvršenja na stvarnim kompjuterima.

Naprimjer, Teorem 1.1.1 nam kaže da deterministički modeli Turingove mašine sa jednom i sa više traka su polinomijalno ekvivalentni.

Dalje se fokusiramo samo na aspekte vremenske kompleksnosti na koje ne utiče polinomijalne razlike u vremenu izvršenja. Smatramo ih beznačajnim te ih ignorišemo. Radeći na taj način, možemo da izaberemo određeni model za izračunavanje.

U stvarnosti, u programiranju, naravno da nam je cilj dobiti što brže algoritme. Ranije smo, uvodeći asimptotske notacije, zanemarili razlike do konstantnih faktora. Sada predlažemo da se zanemare i mnogo veće polinomijalne razlike, kao što je razlika između vremena n i n^3. To ne znači da takve razlike smatramo nevažnim, naprotiv, smatramo veoma važnim razliku između vremena n i n^3. Ali neka pitanja, kao što je polinomijalnost ili nepolinomijalnost problema faktorizacije, ne ovise o polinomijalnim razlikama.

Sada dolazimo do važne definicije u teoriji kompleksnosti.

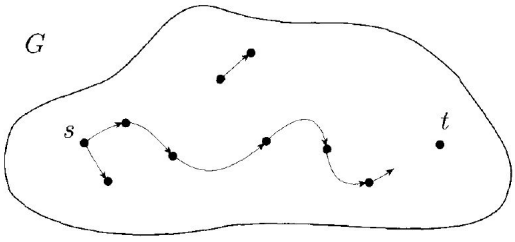
Definicija 1.2.1. **P** *klasa je klasa jezika koji su rješivi (odlučivi) u polinomijalnom vremenu na determinističkoj Turingovoj mašini sa jednom trakom. Drugim riječima,*

$$\mathbf{P} = \bigcup_{k=0}^{+\infty} TIME(n^k).$$

P klasa igra centralnu ulogu u našoj teoriji i važna je iz više razloga. Jedan od njih je da **P** klasa je invarijantna za sve modele izračunavanja koji su polinomijalno ekvivalentni determinističkoj Turingovoj mašini sa jednom trakom. Drugi je taj što **P** klasa odgovara klasi problema koji su realno rješivi na kompjuteru.

1.2.2 Primjeri problema P klase

Prvi primjer kojeg ćemo posmatrati je usmjereni graf. Usmjereni graf G sadrži čvorove s i t, kao što je prikazano na sljedećoj slici.



Probem puta (PATH) je problem određivanja da li postoji usmjereni put od s do t. Neka je

$$\text{PATH}=\{(G,s,t)|\ G \text{ je usmjereni graf koji ima usmjereni put od } s \text{ do } t\}$$

Teorem 1.2.1. *PATH* $\in$ **P**.

Ovaj teorem ćemo dokazati predstavljanjem algoritama sa polinomijalnim vremenom koji rješava PATH. Prije nego ga opišemo, razmotrimo algoritam sirove snage i uočimo da za ovaj problem on nije odgovarajuće brz.

Algoritam sirove snage za PATH nastavlja da pregleda sve moguće puteve u G i određuje koji je usmjereno do čvora s do čvora t. Mogući put je niz čvorova grafa G koji ima dužinu najviše n, pri čemu je n broj čvorova u grafu G. Naravno, ovdje podrazumijevamo da nema ponavljajnja na nekom čvoru. Ali, ukupan broj mogućih puteva je n^n, što je eksponencijalno po broju čvorova u G. Zato, algoritam sirove snage koristi eksponencijalno vrijeme, pa nam nije interesantan.

Da bi dobili algoritam sa polinomijalnim vremenom, moramo da uradimo nešto da izbjegnemo sirovu snagu. Jedan od načina je da koristimo pretraživanje grafa kao što je pretraživanje po širini. U njemu, uzastopno označavamo sve čvorove u G koji se mogu dostići iz čvora s usmjerenim putevima, prvo dužine 1, pa dužine 2, pa sve do dužine n. Ograničenje vremena u ovoj strategiji sa polinomom je veoma lako.

Polinomijalni algoritam za ovaj problem izgleda kao:

M= "Na ulazu (G, s, t), gdje je G usmjereni graf sa čvorovima s i t

1. Označiti čvor s.
2. Ponavljaj sve dok se dodatni čvor ne označi:
3. Skeniraj sve ivice od G. Ako ivica (a, b) ide od označenog čvora a do neoznačenog čvora b, označi čvor b.
4. Ako je označen čvor t, *prihvati*. U suprotnom, *odbij*."

Analizirajmo algoritam M. Očigledno je da se prva i četvrta faza izvršavaju samo jednom. Faza 3 se izvršava najviše n puta je svaki put, osim posljednjeg, označava dodatni čvor u G. Dakle, najviše se izvršava $1 + 1 + n$ koraka, što je polinomijalno po dužini od G.

Prva i četvrta faza se lako implementiraju u polinomijalnom vremenu na bilo kojem razumnom determinističkom modelu. Treća faza uključuje skeniranje ulaza i testiranje da li su određeni čvorovi označeni, što je također izvodivo u polinomijanom vremenu. Dakle, M je algoritam sa polinomijalnim vremenom za problem PATH.

Posmatrajmo sada drugi primjer algoritma sa polinomijalnim vremenom. Za dva broja kažemo da su *relativno prosti* ako je broj 1 najveći cio broj koji ih oba dijeli. Naprimjer, brojevi 10 i 21 su relativno prosti, dok to nisu brojevi 10 i 22, jer su oba djeljiva sa 2. Neka je RELPRIME problem testiranja da li su dva broja relativno prosta, tojest:

$$\text{RELPRIME}=\{(x, y) |\ x \text{ i } y \text{ su relativno prosti}\}.$$

Teorem 1.2.2. *RELPRIME* $\in \mathbf{P}$.

Jedan od algoritama koji rješava ovaj problem jeste da pretražuje sve moguće djelioce oba broja i prihvata ako nijedan nije veći od 1. Ipak, veličina broja predstavljenog u binarnom zapisu ili nekoj drugoj bazi $k \geqslant 2$ je eksponencijalna po dužini njegove reprezentacije. Prema tome, algoritam sirove snage pretražuje u eksponencijalnom broju mogućih djelilaca pa ima eksponencijalno vrijeme izvršenja.

Umjesto toga, ovaj problem ćemo riješiti pomoću Euklidovog algoritma koji određuje najveći zajednički djelilac. Najveći zajednički djelilac prirodnih brojeva x i y, kojeg označavamo sa $nzd(x, y)$, je najveći cio broj koji dijeli i x i y. Naprimjer $nzd(18, 24) = 6$. Očigledno, brojevi x i y su uzajamno prosti ako je $nzd(x, y) = 1$. Euklidov algoritam

ćemo opisati kao algoritam E, koji koristi funkciju mod, pri čemi je $x \bmod y$ ostatak prilikom dijeljenja cijelog broja x sa cijelim brojem y.

Euklidov algoritam glasi

E= "Na ulazu (x, y), gdje su x i y prirodni brojevi u binarnom zapisu:

1. Ponavljaj sve dok je $y = 0$:
2. Dodijeli $x \leftarrow x \bmod y$.
3. Zamijeni x i y.
4. Izlaz x."

Algoritam R koji rješava RELPRIME, koriteći algoritam E, glasi:

R= "Na ulazu (x, y), gdje su x i y prirodni brojevi u binarnom zapisu:

1. Pokreni E na (x, y).
2. Ako je rezultat 1, *prihvati*. U suprotnom, *odbij*."

Jasno je da, ako se algoritam E ispravno izvršava u polinomijalnom vremenu, onda se i R tako izvršava. Zbog toga samo trebamo analizirati algoritam E u smislu vremena i ispravnosti. Ispravnost ovog algoritma je dobro poznata, pa ćemo samo analizirati njegovo vrijeme.

Da bi analizirali vremensku kompleknost algoritma E, prvo ćemo pokazati da je svako izvršenje druge faze (osim možda prvi put), smanjuje vrijednost od x najmanje za pola. Poslije druge faze vrijedi da je $x < y$, zbog prirode funkcije mod. Poslije treće faze je $x > y$ jer su se ova dva zamijenila. Odavdje zaključujemo, da kada se druga faza naknadno izvrši, vrijedi $x > y$ faza. Ako je $\frac{x}{2} \geqslant y$, tada je $x \bmod y < y \leqslant \frac{x}{2}$ pa x se smanji bar za pola. Ako je $\frac{x}{2} < y$, tada je $x \bmod y = x - y < \frac{x}{2}$, pa x se smanji bar za pola. Vrijednosti od x i y se zamijene svaki put kada se treća faza izvrši, pa se svaka od početnih vrijednosti x i y smanji bar za pola svaki put u petlji. Prema tome, maksimalan broj izvršenja druge i treće faze je manji od $2 \log_2 x$ i $2 \log_2 y$. Ovi logaritmi su proporcionalni sa dužinom njihovih reprezentacija, što daje da je broj izvršenih faza jednak $O(n)$. Svaka faza koristi polinomijalno vrijeme, pa je ukupno vrijeme izvršenja također polinomijalno.

Na kraju, naglasimo još jednu bitnu činjenicu iskazanu sljedećim teoremom.

Teorem 1.2.3. *Svaki kontekstno nezavisan jezik je u klasi* $\in \mathbf{P}$.

Dokaz ove teoreme se isto može naći u [2].

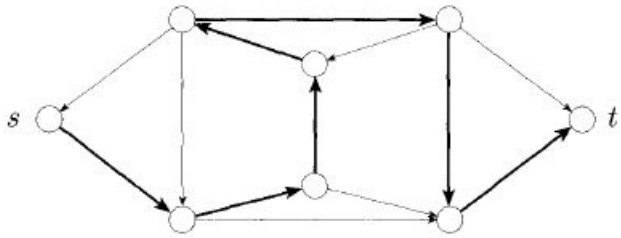
1.3 NP klasa

Kao što smo mogli da vidimo do sada, mi možemo izbjeći brute force pretraživanje kod mnogih problema i dobiti rješenja sa polinomskim vremenom. Međutim, pokušaji izbjegavanja brute forcea kod određenih problema, uključujući neke interesantne i korisne, pokazali su se kao neuspješni i da algoritmi sa polinomskim vremenom koji ih rješavaju, nisu nam poznati.

Možemo postaviti pitanje: zažto nismo imali uspjeha u pronalaženju algoritama sa polinomskim vremenom za takve probleme? Odgovor na ovo pitanje ne znamo. Možda ovi problemi imaju još neotkrivene algoritme sa polinomskim vremenom koji počivaju na nepoznatim pricipima. Ili možda neki od ovih problema uopšte ne mogu biti riješeni u polinomskom vremenu.

Nevjerovatno otkriće vezano za ovo pitanje jeste da su kompleksnosti mnogih problema povezane. Algoritam sa polinomskim vremenom za jedan takav problem se može iskorititi za rješavanje cijele klase problema. Da bismo to bolje razumijeli, analizirajmo sljedeći problem.

Hamiltonov put u usmjerenom grafu G je usmjereni put koji ide kroz svaki čvor samo jedanput. Posmatramo problem u kojem testiramo da li usmjereni graf sadrži Hamiltonov put koji spaja da određena čvora, kao što je prikazano na sljedećoj slici.



Neka je

$$\text{HAMPATH}=\{(G,s,t)|\ G \text{ je usmjereni graf koji ima Hamiltonov put od } s \text{ do } t\}$$

Veoma lako možemo dobiti algoritam sa eksponencijalnim vremenom za HAMPATH problem modifikujući brute force algoritam za PATH problem u Sekciji 1.2. Potrebno je samo provjeriti da li je potencijalni put i Hamiltonov put. Još uvijek niko ne zna da li je HAMPATH problem rješiv u polinomskom vremenu.

HAMPATH problem nema osobinu takozvane polinomske verifikacije koja je važna da bi se razumjela kompleksnost. Iako ne poznajemo brzi način (polinomsko vrijeme) da odredimo da li graf sadrži Hamiltonov put, ako bi se takav put nekako i otkrio, mogli bismo lako nekoga da uvjerimo da postoji jednostavno prezentujući ga. Drugim riječima, verifikacija postojanja Hamiltonovog puta je mnogo lakša od određivanja njegovog postojanja.

Drugi problem koji se može polinomski verifikovati je problem složenosti. Prisjetimo se da je prirodni broj složen ako je proizvod dva cijela broja koji su veći od 1. Neka je

$$\text{COMPOSITES}=\{x|x = pq \text{ za cijele brojeve } p,q > 1\}.$$

Veoma brzo možemo provjeriti da je broj složen jer sve što trebamo je djelilac tog broja. Nedavno je otkriven algoritam sa polinomskim vremenom koji testira da li je broj složen ali je znatno složeniji od prethodne metode za provjeru složenosti.

Neki problemi možda neće biti moguće polinomski provjeriti. ako anprimjer uzmemo komplemnet od problema HAMPATH. Ako bismo nekako mogli i da odredimo da graf nema Hamiltonov put ne bismo znali način da nekome drugom pokažemo nepostojanje bez korištenja istog eksponencijalnog algoritma za određivanje postojanja. Formalna definicija slijedi.

Definicija 1.3.1. *Verifikator za jezik A je algoritam V gdje je*

$$A=\{w|V \text{ prihvata } (w,c) \text{ za neki string } c\}.$$

Mi mjerimo vrijeme verifikatora samo u zavisnosti od dužine stringa w. Dakle verifikator sa polinomskim vremenom se pokreće u polinomskom vremenu od dužine stringa w. Jezik je polinomski provjerljiv (verifikovan) ako ima verifikatora sa polinomskim vremenom.

Verifikator koristi dodatnu informaciju c da bi verifikovao da je string w član jezika A. Ova informacija se naziva certifikat ili dokaz članstva u jeziku A. Uočimo da za polinomske verifikatore, certifikat ima polinomsku dužinu (u dužini stringa w) jer je to sve čemu verifikator može pristupiti u svom vremenskom ograničenju. Primijenimo ovu definiciju na probleme HAMPATH i COMPOSITES.

Za HAMPATH problem, certifikat za $(G,s,t) \in HAMPATH$ jednostavno je Hamiltonov put od s do t. Za COMPOSITES problem, certifikat za složeni broj x jednostavno je jedan od djelilaca. U oba slučaja, verifikator može u polinomskom vremenu provjeriti da je ulaz u jeziku kada je dat njegov certifikat.

Definicija 1.3.2. *NP klasa je klasa jezika koji imaju verifikatore sa polinomskim vremenom.*

NP klasa je bitna jer sadrži mnoge probleme koji su nam interesantni. Iz prethodne diskusije možemo zaključiti da su oba problema, HAMPATH i COMPOSITES problemi, iz NP klase. Također, COMPOSITES problem je iz P klase koji je podskup od NP, ali dokazivanje ovog jačeg rezultata je mnogo teže. Izraz NP dolazi od *nedeterminističko polinomsko vrijeme* i izveden je iz alternativne karakterizacije koristeći nedeterminističko polinomsko vrijeme Turingovih mašina. Problemi u klasi NP se nekad nazivaju NP-problemi.

Nedetrministička Turingova mašina (NTM) koja odlučuje HAMPATH problem u nedeterminističkom polinomskom vremenu je data sa:

N_1= "Na ulazu (G,s,t), gdje je G usmjereni graf sa čvorovima s i t

1. napisati listu od m brojeva $p_1, p_2, \ldots, p_m$ pri čemu je m broj čvorova grafa G. Svaki broj u ovoj listi je nedeterministički određen da bude između 1 i m.
2. Provjeri ponavljanja u listi. Ako je neko pronađeno, *odbij*.
3. Provjeri d li je $s = p_1$ i $t = p_m$. Ako bilo koji nije ispunjen, *odbij*.
4. Za svaki i između 1 i $m-1$, provjeri da li je (p_i, p_{i+1}) ivica od G. Ako neki nije, *odbij*. U suprotnom, svi su testovi prošli, *privati*. "

Da bi analizirali ovaj algoritam i verifikujemo da se izvršava u nedeterminističkom

polinomskomm vremenu, analiziramo svaku fazu posebno. U prvoj fazi, nedeterministička selekcija se jasno pokreće u polinomskom vremenu. U drugoj i trećoj fazi, svaki dio je jednaostabv za provjeriti, pa se zajedno izvršavaju u polinomskom vremenu. Na kraju, jasno je da se četvrta faza izvršava u polinomskom vremenu. Dakle, ovaj algoritam se izvršava u nederminističkom polinomskom vremenu.

Teorem 1.3.1. *Neki jezik je u NP klasi akko je odlučiv u nekoj Turingovoj mašini sa nedeterminističkim polinomskim vremenom.*

Dokaz ove teoreme se isto može naći u [2].

Klasu kompleksnosti sa nedeterminističkim vremenom NTIME($t(n)$) definišemo analogno klasi kompleksnosti sa determinsitičkim vremenom TIME ($t(n)$).

Definicija 1.3.3. *NTIME(t(n))=$\{L|L$ je jezik koji je odlučen od nedeterminističke Turingove mašine u $O(t(n))$ vremenu.$\}$*

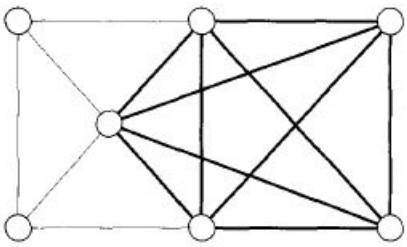
Posljedica 1.3.2. *Vrijedi*

$$NP = \bigcup_k NTIME(n^k).$$

Klasa NP je neosjetljiva na izbor razumnog nedeterminističkog modela izračunavanja jer su svi takvi modeli polinomski ekvivalentni. Kada opisujemo i analiziramo nedeterminističke polinomske vremenske algoritame, slijedimo prethodne konvencije za determinističke polinomne vremenske algoritme. Svaka faza nedeterminističkog polinomskog vremenskog algoritma mora imati očitu provedbua u nedeterminističkom polinomskom vremenu na razumnom nedeterminističkom izračunljivom modelu. Mi analiziramo algoritam da pokažemo da svaka grana koristi najviše polinomski mnogo faza.

1.3.1 Primjeri problema NP klase

Klika u neusmjerenom grafu je podgraf u kojem su svaka dva čvora povezani ivicom. K-klika je klika koja sadrži k čvorova. Na sljedećoj slici je data 5-klika



Problem klika jeste problem da se odredi da li graf sadrži kliku određene veličine. Neka je

$$\text{CLIQUE}=\{(G,k)|\ G \text{ je neusmjeren graf sa } k\text{-klika.}\}$$

Teorem 1.3.3. $CLIQUE \in NP$.

Klika je certifikat. Verifikator V za $CLIQUE$ je:

$V=$ "Na ulazu $((G,k),c)$:

1. Testiraj da li je c skup od k čvorova u grafu G.
2. Testiraj da li G sadrži sve ivice koje povezuju čvorove u c.
3. Ako oboje prolazi, *prihvati*. U suprotnom, *odbij*."

Ovaj teorem smo mogli dokazati i dajući eksplicitno Turingovu mašinu sa nedeterminističkim polinomijalnim vremenom koja odlučuje $CLIQUE$:

$V=$ "Na ulazu $((G,k),c)$:

1. nedeterministički odaberi podskup c od k čvorova u G.
2. Testiraj da li G sadrži sve ivice koje povezuju čvorove u c.
3. Ako da, *prihvati*. U suprotnom, *odbij*."

Posmatrajmo sada $SUBSET-SUM$ problem u kojem imamo kolekciju brojeva $x_1, x_2, \ldots, x_n$ i ciljani broj t. Želimo da odredimo da li data kolekcija sadrži (pod)kolekciju brojeva čija je suma broj t. dakle,

$$\text{SUBSET-SUM}=\{(S,t)|\ S=\{x_1,x_2,\ldots,x_n\} \text{ gdje za neko } \{y_1,y_2,\ldots,y_n\}\subseteq\{x_1,x_2,\ldots,x_n\} \text{ vrijedi } \textstyle\sum y_i = t\}.$$

Naprimjer, $(\{4,11,16,21,27\},25) \in SUBSET-SUM$ jer je $4+21=25$.

Teorem 1.3.4. $SUBSET-SUM \in NP$.

Podskup je certifikat. Verifikator V za $SUBSET-SUM$ je:

$V=$ "Na ulazu $((S,t),c)$:

1. Testiraj da li je c kolekcija brojeva čija je suma t.
2. Testiraj da li S sadrži sve brojeve iz c.
3. Ako oboje prolazi, *prihvati*. U suprotnom, *odbij*."

Ovaj teorem smo mogli dokazati i dajući eksplicitno Turingovu mašinu sa nedeterminističkim polinomijalnim vremenom koja odlučuje $SUBSET-SUM$:

$V=$ "Na ulazu $((S,t),c)$:

1. Nedeterministički odaberi podskup c brojeva u S.
2. Testiraj da li je c kolekcija brojeva čija je suma broj t

3. Ako test prođe, *prihvati*. U suprotnom, *odbij*."

Primijetimo da komplementi od ovih problema, a to su $\overline{CLIQUE}$ i $\overline{SUBSET-SUM}$, nisu očigledno članovi od NP. Verifikacija da nešto nije prisutno čini se težom nego verifikacija da nešto jeste prisutno. Zbog toga, definišemo novu klasu kompleksnosti $coNP$, koja sadrži jezike koji su komplementi od jezika u NP. Ne znamo da li se $coNP$ razlikuje od NP.

1.4 P=NP?

Kao što smo rekli, NP je klasa jezika koji su rješivi u polinomskom vremenu na nedeterminističkoj Turingovoj mašini, ili ekvivalentno, klasa jezika za koje se može verifikovati u polinomskom vremenu (brzo). P klasa je klasa jezika gdje se članstvo može odlučiti u polinomskom vremenu (brzo). Predstavili smo dva jezika $HAMPATH$ i $CLIQUE$ koji su u klasi NP ali nam nije poznato jesu li u klasi P. Moć polinomske verifikacije je čini se veća od polinomske odlučivosti. Iako je teško to i zamisliti, P i NP bi mogle biti jednake. Ne možemo još dokazati postojanje jezika u NP koji nije u P. Pitanje da li je $P = NP$ je jedno od najvećih do sada neriješenih pitanja u teorijskoj kompjuterskoj nauci. Ako bi ove klas ebile jednake, svaki problem koji se može polinomski verifikovati, mogao bi se polinomski i odlučiti. Mnogi vjeruju da ove dvije klase nisu jednake, Razlog je taj što su mnogi pokušali da nađu algoritme sa polinomskim vremenom za probleme u NP, ali bez uspjeha. Naučnici su također pokušali dokazati da ove dvije kalse nisu jednake, ali to bi značilo da treba dokazati da ne postoji brzi algoritam koji bi zamijenio brute-force algoritam.

Najbolji poznati metod za rješavanje jezika u NP dterministički koristi eksponencijalno vrijeme. Drugim riješima možemo dokazati da je

$$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k}).$$

ali ne znamo da li je NP sadržan u manjoj determinističkoj klasi vremenske kompleksnosti.

1.5 NP kompletnost

Do napretka u pitanju odnosa između P i NP dolazi 1970. godine u radu Stephena Cooka i Leonida Levina. Oni su otkrili određene probleme u NP čija je individualna kompleksnost povezana aa kompleksnosti cijele klase. Ako za neki od ovih problema postoji polinomski algoritam, svi problemi u NP bi bili rješivi u polinomskom vremenu. Ovi problemi se nazivaju NP kompletni. NP kompletnost je važna iz teorijskih i iz praktičnih razloga.

Sa teorijske strane, u pokušaju dokazivanja da P i NP nisu jednaki, možemo se fokusirati na NP kompletne probleme. Ako neki problem u NP zahtijeva više od polinomskog vremena, tada NP kompletan također zahtijeva to. U pokušaju dokazivanja da su P i NP jednaki, potrebno je samo naći algoritam sa polinomskim vremenom za NP kompletan problem da bi se cilj postigao.

Sa praktične strane, fenomen NP kompletnosti može izbjeći gubljenje vremena u traženju nepostojećeg polinomskog algoritma da riješi određeni problem. Iako nemamo potrebnu matematiku da bi dokazali da je problem nerješiv u polinomskom vremenu, vjerujemo da je P različit od NP. Dakle, dokazujući da je problem NP kompletan je jak dokaz njegove nepolinomijalnosti.

Prvi NP kompletan problem kojeg predstavljamo ovdje se naziva **problem zadovoljavanja**. Podsjetimo se da promjenljive koje mogu uzeti vrijednosti tačna (1) i netačna (0) se nazivaju Bulovim varijablama, dok je Bulova formula izraz koji uključuje Bulove varijable i operacije ($\wedge$, $\vee$, $\neg$). Naprimjer,

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

je Bulova formula. Bulova formula je zadovoljena ako neko pridruživanje 0 i 1 varijablama formulu čini tačnom. Prethodna formula je tačna ako pridružimo $x = 0$, $y = 1$ i $z = 0$. Također kažemo da pridruživanje zadovoljava formulu ϕ. Problem zadovoljavanja je test da li je Bulova formula zadovoljena. Neka je

$$\text{SAT}=\{(\phi)|\ \phi \text{ je zadovoljena Bulova formula.}\}$$

Sljedeći teorem povezuje kompleksnost SAT problema sa kompleksnosti svih problema u NP.

Teorem 1.5.1 (Cook-Levina teorema)**.** $SAT \in P \Leftrightarrow P = NP$.

U prethodnom izlaganju smo predstavili koncept reduciranja jednog problema na drugi. Kada se problem A reducira na problem B, rješenje problema B se može iskoristiti da bi se riješio problem A. Sada ćemo definisati verziju reduciranja koja uzima efikasnost izračunavanja u obzir. Kada je problem A efikasno reduciran na problem B, efikasno rješenje problema B se može iskoristiti da bi se problem A riješio efikasno.

Definicija 1.5.1. *Funkcija $f : \sum^* \to \sum^*$ je izračunljiva funkcija sa polinomskim vremenom ako postoji Turingova mašina M sa polinomskim vremenom koja se zaustavlja sa samo $f(w)$ na svojoj traci kada je počela sa ulazom w.*

Definicija 1.5.2. *Jezik A je reducibilan polinomskim vremenom na jezik B i pišemo $A \leq_P B$ ako postoji izračunljiva funkcija sa polinomskim vremenom $f : \sum^* \to \sum^*$, gdje za svako w vrijedi*

$$w \in A \Leftrightarrow f(w) \in B.$$

Funkcija f se naziva redukcija sa polinomskim vremenom A na B.

Kao i sa običnim preslikavanjem redukcije, redukcija sa polinomskim vremenom A na B omogućava način da se testiranje članstva u A prebaci na testiranje članstva u B, ali je sada prebacivanje urađeno efikasno. Za testiranje da li je $w \in A$, koristimo redukciju f da bi preslikali w u $f(w)$ i testirali da li je $f(w) \in B$.

Ako je poznato da je neki jezik reducibilan sa polinomskim vremenom na drugi jezik za kojeg znamo da ima rješenje sa polinomskim vremenom, dobijamo rješenje sa polinomskim vremenom originalnog jezika, kao što kaže sljedeća teorema.

Teorem 1.5.2. *Ako je $A \leq_P B$ i $B \in P$, tada je $A \in P$.*

Neka je M algoritam sa polinomskim vremenom koji odlučuje B i neka je f redukcija sa polinomskim vremenom iz A u B. Na sljedeći način opisujemo algoritam N sa polinomskim vremenom koji odlučuje A:

N= "Na ulazu w:

1. Izračunati $f(w)$.
2. Pokreni M na ulazu $f(w)$ i stavi za izlaz štagod M daje izlaz."

Imamo da je $w \in A$ kad god je $f(w) \in B$ jer je f redukcija sa A u B. Odatle M prihvata $f(w)$ kad god je $w \in A$. Još više, N se pokreće u polinomskom vremenu jer svaka od dvije faze se pokreće u polinomskom vremenu. Primijetimo da je druga faza u polinomskom vremenu jer je kompozicija dva polinoma opet polinom.

Posmatrajmo sada novi problem koji je poseban slučaj problema zadovoljavanja pri čemu su sve formule u specijalnom obliku:

$$\text{3SAT}=\{(\phi)|\ \phi \text{ je zadovoljena 3knf formula.}\}$$

pri čemu je 3knf konjuktivna normalna forma sa tri člana, kao naprimjer

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_3) \wedge (x_4 \vee x_5 \vee x_6)$$

Primijetimo da u zadovoljenoj knf formuli svaki član mora imati dio kome je pridružena 1.

Sljedeći teorem nam predstavlja redukciju sa polinomskim vremenom sa 3SAT problema na CLIQUE problem.

Teorem 1.5.3. *3SAT je sa polinomskim vremenom reducibilan na CLIQUE .*

Redukcija f iz 3SAT u CLIQUE sa polinomskim vremenom koju ćemo predstaviti, prebacuje formule u grafove. U konstruisanom grafu, klike određene veličine odgovaraju

zadovoljenju formule. Strukture unutar grafa su konstruisane tako da oponašaju ponašanje varijabli i izraza formule.

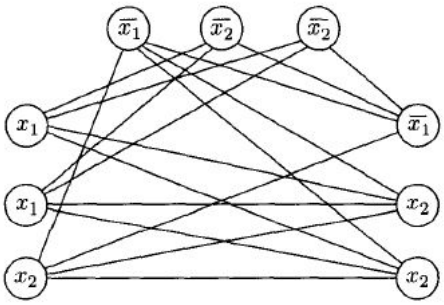
Neka je

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$$

formula sa k članova. Redukcija f generira string (G, k), pri čemu je G neusmjeren graf definisan kako slijedi. Čvorovi u G su organizirani u k grupa po 3 čvora koji se nazivaju trojke $t_1, t_2, \ldots, t_k$. Svaka trojska odgovara članovima u ϕ i svaki čvor u trojci odgovara članu u određenoj trojci. Označimo svaki čvor u G sa određenim članom formule ϕ. Ivice u G povezuju sve čvorove osim dva tipa. Ne povezuju se čvorovi unutar jedne tojske i ne povezuju se čvorovi sa kontradiktornim oznakama (recimo x_1 i $\overline{x_2}$).

Sljedeći graf ilustruje ovu konstrukciju kada je formula data sa

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



Sada ćemo pkazati da je ϕ zadovoljena akko graf G sadrži k kliku.

Pretpostavimo da je formula ϕ odgovarajućeg tipa i da je zadovoljena. Dakle, barem jedan iskaz je tačan u svakom članu. U svakoj trojci grafa G, izaberimo jedan čvor koji odgovara tačnoj varijabli u članu formule. Ako je u određenom članu formule više od jednog iskaza tačno, biramo samo jednog. Upravo izabrani čvorovi formiraju k kliku. Broj izabranih čvorova je k jer smo izabrali po jedan u k trojki. Svaki par čvorova je povezan ivicom jer se čvorovi povezuju na ranije opisani način. Ne mogu biti izabrani iz iste trojske jer smo izabrali samo jedan član u trojci. Ne mogu biti kontradiktorni, jer povezujemo čvorove koji su oba tačna u datom pridruživanju. Prema tome, graf G sadrži k kliku.

Obrnuto, pretpostavimo da graf G sadrži k kliku. Nijedan od čvorova te klike nije iz iste trojke jer čvorovi iz iste trojske nisu povezani ivicama. Prema tome, svaka od k trojki sadrži tačno jedan od čvorova u k kliki. Sada dodjeljujemo tačnost varijablama u formuli ϕ tako da svaki iskaz označava čvor u kliki. To je moguće uraditi jer dva čvora koja označavaju kontradiktorsnost nisu povezana i zato oba ne mogu biti u istoj kliki. Ovo pridruživanje varijablama zadovoljava formulu ϕ jer svaka trojka zadrži čvor klike i zato svaki član sadrži sikaz kome je pridružena tačnost. Prema tome, formula ϕ je zadovoljena.

Definicija 1.5.3. *Jezik B je NP-kompletan ako zadovoljava dva uslova:*

1. $B \in NP$,

2. *svaki* $A \in NP$ *je reducibilan polinomskim vremenom na* B.

Teorem 1.5.4. *Ako je* B *NP-kompletan i* $B \in P$*, tada je* $P = NP$.

Dokaz ove teoreme direktno slijedi iz definicije polinomske reducibilnosti.

Teorem 1.5.5. *Ako je* B *NP-kompletan i* $B \leq_P C$ *za* $C \in NP$*, tada je* C *NP-kompletan.*

Već znamo da je C u NP pa trebamo pokazati da je svaki A iz NP polinomski reducibilan na C. Budući da je B NP-kompletan, svaki jezik u NP je polinomski reducibilan na B i B je zatuzvrat polinomski reducibilan na C. Polinomske reducibilnosti se slažu, dakle ako je A polinomski reducibilan na B i B polinomski reducibilan na C, tada je A polinomski reducibilan na C. Prema tome, svaki jezik u NP je polinomski reducibilan na C.

1.5.1 Cook-Levin teorem

Jednom kada dobijemo NP-kompletan problem, možemo dobiti druge probleme polinomskom redukcijom tog problema. Ipak, da bi dobili prvi NP-kompletan problem je jako teško. To ćemo uraditi dokazom sljedeće teoreme koja je drugačija formulacija Teoreme 1.5.1.

Teorem 1.5.6. *SAT je NP-kompletan.*

Dokaz ove teoreme se isto može naći u [2].

Posljedica 1.5.7. *3SAT je NP-kompletan.*

Dokaz ove posljedice se isto može naći u [2].

1.5.2 Još NP-kompletnih problema

Posljedica 1.5.8. *CLIQUE je NP-kompletan.*

Dokaz ove teoreme se isto može naći u [2].

Teorem 1.5.9. *VERTEX-COVER je NP-kompletan.*

Dokaz ove teoreme se isto može naći u [2].

Teorem 1.5.10. *HAMPATH je NP-kompletan.*

Dokaz ove teoreme se isto može naći u [2].

Teorem 1.5.11. *UHAMPATH je NP-kompletan.*

Dokaz ove teoreme se isto može naći u [2].

Teorem 1.5.12. *SUBSET-SUM je NP-kompletan.*

Dokaz ove teoreme se isto može naći u [2].

Bibliografija

[1] C Moore and S Mertens. *The Nature of Computation.* Oxford University Press, 2011.

[2] M Sipser. *Introduction to the Theory of Computation.* Thompson Learning, Inc., 2006.