VERJETNOSTNI MODEL RAČUNANJA

INFORMATICA 1/89

Deskriptorji: RAČUNANJE, VERJETNOSTNI MODEL, ALGORITMI HEVRISTIČNI

Janez Žerovnik
Inštitut za matematiko, fiziko in mehaniko, Ljubljana

POVZETEK: V preglednem članku predstavimo verjetnostni model računanja in podamo definicijo nekaterih razredov časovne zahtevnosti verjetnostnih algoritmov. V začetnih razdelkih vpeljemo klasični (deterministični) model računanja in ponovimo znane definicije iz teorije časovne zahtevnosti algoritmov.

ABSTRACT: A Probabilistic Model of Computation

In article a survey of a probabilistic model of computation is given. Some classes of probabilistic algorithms are defined. Preliminary sections give definition of classical (deterministic) model of computation and introduction to the theory of time complexity of computation.

1. UVOD

V članku bomo vpeljali nekatere pojme in definicije iz teorije časovne zahtevnosti algoritmov. V slovenščini s tega področja poznamo dve deli [Pisa83, PePi82] in prevod [AhUl86], tuja literatura pa je precej bolj bogata. Samo problemu trgovskega potnika na primer je posvečena cela knjiga [LLRS85]. Že klasična je knjiga Gareya in Johnsona [GaJo79], mi pa bomo zaradi novejših definicij nekaterih razredov slučajnih algoritmov sledili knjigi [Melh84], oprli pa se bomo še na [HoUl79] in [Gill77].

V zadnjem desetletju in pol so razred NP–polnih odločitvenih problemov veliko raziskovali. Vprašanje, ali je razred P enak razredu NP je verjetno eden najpomembnejših odprtih problemov teoretičnega računalništva. Teoretično računalništvo tu imenujemo vedo, ki jo nekateri imenujejo tudi informatika (Informatique), kibernetika uporabna matematika (Prikladnaja matematika) ali 'algoritmika' (algorithmics) [Knut81]. Če bi za katerikoli NP–poln problem uspeli pokazati, da je v P, bi sledilo P=NP.

Ker je veliko praktičnih problemov NP–polnih, si v praksi pomagamo do delnih rešitev na različne načine. Aproksimativni algoritmi na primer so taki algoritmi, ki v polinomskem času najdejo rešitev, ki je 'dovolj blizu' optimalne rešitve. Deterministični hevristični * algoritmi imajo pogosto lepo lastnost, da najdejo dobre rešitve na neki podmnožici problemske domene. Običajno imajo taki algoritmi 'Ahilove pete': obstajajo primeri, na katerih se algoritem obnaša zelo slabo. Avtorji algoritmov običajno že sami navajajo primere, ki so neugodni za algoritem [WePo67,DuBr81]. Pri slučajnih hevurističnih algoritmih nekatere korake naredimo slučajno. S tem se včasih izognemo 'Ahilovim petam' na račun nekaj večje časovne zahtevnosti.

Eden od intuitivno najlažje razumljivih NP-polnih problemov je problem barvanja točk grafa. Ker je sestavek nekoliko predelano poglavje avtorjeve magistrske naloge 'Verjetnostni algoritem za barvanje točk grafa', je tudi tu najpogosteje obravnavani primer ravno problem barvanja točk grafa. *Graf* je kombinatorični objekt, ki ga sestavlja ta množica točk V in množica E, v kateri so pari točk. Elementom množice E pravimo *povezave*. Točki, ki sta povezani s povezavo, sta *sosednji*. *Barvanje* (točk grafa) je prireditev barv točkam grafa. Za barve običajno vzamemo kar naravna števila. Barvanje je *dobro*, če je vsak par sosednjih točk pobarvan z različnima barvama. Formalno to zapišemo na primer takole: $b : V \to N$ je dobro barvanje $\iff (u, v) \in E \implies b(u) \neq b(v)$. Odločitveni problem barvanja grafa lahko zdaj zastavimo takole: Ali za dani graf G in množico barv $\{1, \ldots, k\}$ obstaja dobro barvanje? Če tako barvanje obstaja, rečemo, da je G *k-pobarvljiv*. Pogosto je zanimiva optimizacijska varianta problema: Poišči minimalni k, da je G še k-pobarvljiv! Ta minimalni k imenujemo *kromatično število* grafa G. V nadaljevanju bomo predpostavljali, da bralec pozna osnovne pojme teorije grafov, kot so vpeljani na primer v [BaPi85] ali [CvMi77].

Problem barvanja grafa je zanimiv tudi iz zgodovinskih razlogov. Problem štirih barv je bil skoraj sto let velika uganka, ob kateri se je razvijala teorija grafov. Problem si je prvi zastavil leta 1852 londonski študent Francis Guthrie, ko je barval zemljevid angleških grofij. Domneval je, da je mogoče vsak zemljevid pobarvati s štirimi barvami, seveda tako, da ozemlja, ki mejijo, niso pobarvana z isto barvo. Pri tem ozemlja, ki se dotikajo samo v končno mnogo točkah ne štejemo za sosednja. Problem običajno zastavimo v nekoliko drugačni obliki. Namesto ozemelj barvamo točke (lahko si mislimo, da so to glavna mesta), povezani pa sta točki, ki sta v sosednjih državah. Problem se potem glasi: ali je mogoče točke vsakega planarnega grafa pobarvati s štirimi barvami (tako da pobarvamo poljubni povezani točki z različnima barvama)? Francisov

* Splošno privzete definicije hevurističnega algoritma ni [Tros83]. Po Verbinčevem Slovarju tujk je hevristika: 'nauk o metodah raziskovanja (z uporabljanjem še nedokazanih metod, hipotez, itd.)'. Za našo rabo naj hevristični algoritem pomeni postopek, ki ga uporabljamo, čeprav nimamo dokaza, da so njegove rešitve vedno točne (optimalne). Vemo pa, da na primer velja, da je točen na neki podmnožici problemske domene ali da točno rešuje nek 'podoben' problem ali kakšen podoben delni rezultat.

brat Frederic je problem predstavil profesorju Augustu de Morganu. Širše znan je problem postal leta 1878, ko je Arthur Cayley na srečanju Londonske matematične družbe (London Math. Society) vprašal, ali je problem že rešen.

Prvi se je reševanja problema resno lotil A.B.Kempe, ki je leta 1879 objavil dokaz, da štiri barve zadoščajo. Deset let kasneje je P.J.Heawood odkril napako v Kempejevem dokazu, ki pa so jo mnogi imeli bolj za tehnično pomanjkljivost kot pa za resno napako. Ko pa so leta minevala in nikomur ni uspelo popraviti dokaza, je postalo jasno, da problem ni od muh. Leta 1976 sta Kenneth Appel in Wolfgang Haken objavila, da sta dokazala izrek o štirih barvah. Ideja dokaza je v bistvu enaka Kempejevi, le število posebnih primerov, ki jih je bilo potrebno pregledati, je precej večje (okoli 1500 namesto 5!). Obsežnost je tudi glavna hiba dokaza, saj je za pregled (okoli 600 strani) dolgega dokaza potrebno ogromno časa, tudi če za nekatera rutinska opravila uporabimo računalnik, kot sta to storila avtorja dokaza [WoWi78,ApHa86].

2. Primer NP–polnega problema

Za motivacijo si v tem razdelku še pred formalnimi definicijami oglejmo problem klike (skupka). Zastavimo ga takole: Dan je (neusmerjen) graf $G = (V, E)$ in naravno število k. Ali v G obstaja podgraf, ki je izomorfen polnemu grafu K_k? Obstaja enostaven, toda neučinkovit algoritem za reševanje tega problema. Pregledamo vse podgrafe v G, ki so inducirani z množicami točk kardinalnosti k. Za vsak tak podgraf pa preverimo, če je poln. Med n točkami lahko izberemo k točk na $\binom{n}{k}$ načinov. V primeru $k = n/2$ torej naš algoritem preveri $\binom{n}{n/2} \geq 2^n/(n+1)$ podmnožic. Preverjanje, ali je graf poln, je enostavno in hitro: za vsako od $n(n-1)/2$ povezav pogledamo, ali je v induciranem grafu. Naš naivni algoritem torej zahteva čas, ki je eksponentna funkcija števila točk danega grafa. Preden trdimo, da je to neučinkovito, povejmo, kaj mislimo z velikostjo problema. Predpostavljali bomo, da so kombinatorični objekti (grafi, množice, cela števila,...) podani v 'normalni obliki' s končno abecedo. Tako bo na primer graf z n točkami zakodiran v obliki zaporedja n^2 bitov, v katerem so zložene vrstice matrike sosednosti. Cela števila zapišemo v binarnem zapisu in množice z zaporedjem elementov v nekakšnem redu. Primer problema klike na grafu z n točkami je potem zaporedje bitov dolžine $m := n^2 + \log(n/2) + 1$, če spet zaradi enostavnosti vzamemo primer $k = n/2$. Naš naivni algoritem torej razpoznava jezik CLIQUE $= \{wv; w, v \in \{0,1\}^* \text{ in } |w| = n^2$ za neki n in graf G z matriko sosednosti w ima kliko velikosti k, kjer je v binarni zapis števila $k\}$ v času $O(2\sqrt{m})$, kjer je m velikost problema (dolžina vhoda). Algoritma, ki bi bil bistveno boljši od opisanega, ne poznamo. Ker je problem klike NP–poln, je zelo malo verjetno, da tak algoritem sploh obstaja. To bi namreč pomenilo, da obstajajo učinkoviti (polinomski, kot bomo kasneje definirali) algoritmi za vse NP–polne probleme, na primer za problem trgovskega potnika in problem izpolnjivosti. Splošno je privzeta hipoteza 'P≠NP', da takih algoritmov ni. Dokaz ali protidokaz te hipoteze je verjetno najbolj znan odprt problem teoretičnega računalništva.

Še malo se zadržimo pri problemu klike. Podmnožic kardinalnosti k med n točkami je $\binom{n}{k}$, torej obstaja zelo veliko kandidatov za kliko. Po drugi strani je zelo lahko preveriti, ali je dani kandidat klika ali ne. Edina znana pot pa je pregledati vse kandidate.

Če bi imeli na razpolago nedeterministični ukaz CHOICE, bi lahko problem rešili učinkoviteje. Ukaz bi bil na primer oblike

CHOICE $Naslov_1$, $Naslov_2$

Z nedeterminizmom je mišljeno, da izbira nadaljevanja nima nobene vnaprej določene verjetnosti. Ukaz CHOICE pač pomeni, da se algoritem lahko nadaljuje na naslovu $Naslov_1$ ali pa na naslovu $Naslov_2$. Algoritme z ukazom CHOICE imenujemo nedeterministične. Tak algoritem razpozna vhodno zaporedje, če obstaja vsaj ena izvedba algoritma, ki razpozna vhod. Časovna zahtevnost nedeterminističnega algoritma je čas najkrajše izvedbe, ki je razpoznala vhod.

Za primer poglejmo, kako bi problem klike rešili z nedeterminističnim algoritmom. Nedeterminizem bomo uporabili pri generiranju kandidatov za kliko. Algoritem je sestavljen iz treh faz. Najprej pregledamo vhodno zaporedje bitov in preverimo, ali je vprašanje dobro postavljeno. Če vhod ni oblike wv in $|w|$ ni kvadrat naravnega števila, ga zavrnemo, sicer pa izračunamo $n = \sqrt{|w|}$ in k. Z $|w|$ smo označili dolžino zaporedja znakov (v tem primeru znakov iz abecede $\{0,1\}$) w. Časovno zahtevnost prve faze lahko ocenimo z $O(n^4)$. V drugi fazi nedeterministično izberemo podmnožico V kardinalnosti k, tako da zgeneriramo zaporedje n bitov z natanko k enicami. To naredimo takole:

```
for i := 1 to n
do CHOICE m1, m2
   m1 : A[i] := 0;
   m2 : A[i] := 1; k := k − 1;
od
if k ≠ 0 then zavrni;
```

Uspešna izvedba gornjega dela algoritma generira eno od $\binom{n}{k}$ podmnožic V. Potrebni čas je očitno reda $O(n)$. V tretji fazi preverimo, ali je izbrana podmnožica klika. Tudi to lahko storimo hitro, vsaj $O(n^4)$. Opisani algoritem potrebuje čas $O(n^4) = O(m^2)$, kjer je m dolžina vhodnega zaporedja. Podani algoritem res razpoznava jezik CLIQUE: Če G ne vsebuje klike velikosti k, potem v drugi fazi zgenerirana podmnožica gotovo ni klika, torej izvedbe, ki bi sprejela vhod, ni. Če pa graf G ima kliko velikosti k, potem v eni od mogočih izvedb algoritem zgenerira prav to kliko. Ta izvedba seveda sprejme vhod.

Videli smo, da lahko problem klike rešimo v polinomskem času z nedeterminističnim algoritmom. Razred problemov, ki so rešljivi v polinomskem času z nedeterminističnim algoritmom označimo z NP. S P pa označimo razred problemov, rešljivih v polinomskem času z determinističnim algoritmom. Med NP problemi posebej obstajajo problemi, na katere je mogoče prevesti (v polinomskem času) vsak drug NP problem. To so NP–polni problemi. Cook je v članku [Cook71] je pokazal, da je problem izpolnjivosti (SAT) NP–poln. S polinomskim prevajanjem pa je bilo od tedaj za veliko problemov dokazano, da so tudi NP–polni, med drugimi tudi problem klike in problem barvanja točk grafa [Karp72]. Obsežni seznami NP–polnih problemov so v knjigah, na primer v [GaJo79].

Za strogo obravnavo opisanih problemov potrebujemo formalno definiran model računanja, ki ga podajamo v naslednjem razdelku.

3. Turingov stroj

V tem razdelku bomo definirali model računanja s pomočjo katerega bomo lahko strogo definirali časovno zahtevnost algoritmov in razdelili probleme glede na težavnost. Model računanja, ki ga običajno izberemo, je Turingov stroj (TS), saj kljub enostavnosti ni nič šibkejši od katerega koli doslej zgrajenega 'superračunalnika'. Vsi znani modeli so namreč polinomsko prevedljivi na model TS, kar pomeni, da so v smislu teorije, ki nas tu zanima, ekvivalentni. Če drži Churcheva teza, pa to velja za vse mogoče modele računanja! Naša definicija se rahlo razlikuje od tistih v [Pisa83] in [PePi82], vendar nas to ne moti zaradi znane ekvivalentnosti modelov računanja [PePi82, GaJo79, Melh84, AhHU76, HoUl86].

Turingov stroj ima dve enoti: kontrolno in spominsko. Spomin je v desno stran neomejen trak, ki je razdeljen na kvadratke, v katerega je mogoče zapisati en znak končne abecede, ki jo uporablja konkretni Turingov stroj. Kontrolna enota ima končno mnogo stanj in lahko naenkrat dosega en znak (kvadratek) na traku. Turingov stroj programiramo s tabelo, v kateri za

vsako stanje in vsak znak abecede (v trenutno vidnem kvadratku) določimo množico možnih operacij. Če je množica prazna, se TS ustavi. Če je Turingov stroj deterministično ima množica možnih operacij v vsakem primeru največ en element, obnašanje deterministočnega Turingovega stroja je s tabelo in vhodnim podatkom torej natanko določeno. TS v eni operaciji lahko naredi troje: lahko zapiše znak na trenutno vidni kvadratek traku, lahko premakne 'oko' v levo ali desno in lahko preide v novo stanje.

TS poženemo tako, da zapišemo vhodno zaporedje znakov na prvih n kvadratkov, postavimo 'oko' na prvi (skrajnje levi) kvadratek traku in postavimo TS v (vedno isto) začetno stanje. Vsi drugi kvadratki na traku so v začetku prazni, vsebujejo torej posebni znak abecede B. TS izvaja operacije, dokler ne naleti na par (znak, stanje) za katerega v tabeli nima nobene operacije in se ustavi. Rezultat računanja je neprazni del traku. Na ta način lahko TS uporabimo za računanje funkcij. Pogosteje pa rabimo TS za razpoznavanje jezikov. TS razpoznava jezik, če izračuna njegovo karakteristično funkcijo. Da se izognemo (nepotrebnemu) brisanju traku pa se dogovorimo, da odlikujemo nekaj stanj in rečemo: če se Turingov stroj ustavi v katerem od teh stanj, je razpoznal vhod za element jezika. Obe definiciji sta ekvivalentni, druga pa je primernejša za obravnavo nedeterminizma.

1.DEFINICIJA: Turingov stroj (TS) je sedmerka $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kjer je

Q končna množica stanj

Γ tračna abeceda

$B \in \Gamma$ prazen simbol

$\Sigma, \Sigma \subseteq \Gamma \setminus \{B\}$ množica vhodnih simbolov

δ prehodna funkcija $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$, ki ni nujno definirana na vsej množici $Q \times \Gamma$

$q_0 \in Q$ začetno stanje

$F \subseteq Q$ množica končnih stanj

Zaradi nedvoumnosti privzamemo, da sta Γ in Q disjunktni množici simbolov. Trenutno stanje Turingovega stroja opisuje *konfiguracija*, to je niz oblike $\alpha_1 q \alpha_2 \in \Gamma^* \times Q \times \Gamma^*$, kjer je q trenutno stanje TS, $\alpha_1\alpha_2$ je zapis na traku, TS pa vidi prvi znak niza α_2. Če je $\alpha_2 = \varepsilon$ prazen niz, potem TS po dogovoru vidi prazen znak B.

Definirajmo *korak* TS. Bodi $X_1X_2 \ldots X_{i-1}qX_i \ldots X_n$ konfiguracija. Naj bo $\delta(q, X_i) = (p, Y, L)$, kjer v primeru $i - 1 = n$ vzamemo $X_i = B$. Če je $i = 1$, potem naslednja konfiguracija ni definirana, saj stroj ne sme 'pasti' čez levi rob traku. Če je $i > 1$ potem zapišemo

$$X_1X_2 \ldots X_{i-1}qX_i \ldots X_n \vdash X_1X_2 \ldots X_{i-2}pX_{i-1}YX_{i+1} \ldots X_n$$

Če se nam na koncu zapisa konfiguracije pojavijo prazni simboli, jih po dogovoru zbrišemo.

Podobno definiramo korak, če je $\delta(q, X_i) = (p, Y, R)$.

$$X_1X_2 \ldots X_{i-1}qX_i \ldots X_n \vdash X_1X_2 \ldots X_{i-1}YpX_{i+1} \ldots X_n$$

V primeru $i - 1 = n$ je niz $X_1 \ldots X_n$ je zapis konfiguracije po koraku daljši od zapisa prejšnje konfiguracije.

Konfiguracija $C = X_1X_2 \ldots X_{i-1}qX_i \ldots X_n$ je *ustavitvena*, če velja $\delta(q, X_i) = \emptyset$. Konfiguracija je *sprejemajoča*, če je $q \in F$. Konfiguracija je *začetna* pri podatku $x = X_1, X_2, ..., X_n$, če je $q = q_0$, in $i = 1$.

Izračun pri podatku $x \in \Gamma^*$ je zaporedje konfiguracij $C_0, C_1, ..., C_k$, za katere velja $C_i \vdash C_{i+1}$ za $0 \leq i < k$. Izračun *se ustavi*, če je končna konfiguracija C_k ustavitvena. Izračun je *sprejemajoč*, če je njegova končna konfiguracija C_k sprejemajoča. Brez škode za splošnost lahko privzamemo, da se vsak sprejemajoč račun ustavi.

2.DEFINICIJA: *Nedeterministični Turingov stroj* (NTS) definiramo enako kot TS, le prehodna funkcija δ ima več mogočih vrednosti. Točneje $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \{R,L\}}$.

V posebnem primeru, ko število elementov v sliki δ ni nikoli večje od 1, dobimo prej definirani (deterministični) TS.

3.DEFINICIJA: Bodi $M = (Z, \Gamma, \delta, F)$ TS:

a) $L(M) = \{x \in \Gamma^*;$ obstaja sprejemajoč izračun *TS M z vhodom x*$\}$ je jezik, ki ga razpoznava M

b) Bodi M deterministično in *totalen*, torej naj se M ustavi pri poljubnem vhodnem nizu $x \in \Gamma^*$. Potem M računa funkcijo $f_M : \Gamma^* \rightarrow \Gamma^*$, če je za vsak $x \in \Gamma^*$ vsebina traku ob ustavitvi enaka $f_M(x)$.

c)(Časovna zahtevnost T_M TS M)

$T_M(n) = \max_{x \in \Gamma^*, |x|=n} \{k;$ *k je dolžina izračuna, ki se ustavi pri vhodu* $x\}$

4.DEFINICIJA: (Razreda P in NP)

P = $\{L; L \subseteq \Gamma^*$ *za neko končno abecedo* Γ *in obstaja deterministični TS M in polinom p, da je* $L = L(M)$ *in* $T_M(n) \leq p(n)$ *za vsak* $n\}$

NP = $\{L; L \subseteq \Gamma^*$ *za neko končno abecedo* Γ *in obstaja nedeterministični TS M in polinom p, da je* $L = L(M)$ *in* $T_M(n) \leq p(n)$ *za vse* $n\}$

V nekaterih virih [GaJo79,AhHU76,Melh84] je časovna zahtevnost NTS definirana nekoliko drugače:

5.DEFINICIJA:

$T_M(n) = \max_{x \in L(M), |x|=n} \min\{k;$ *k je dolžina sprejemajočega izračuna pri vhodu* $x\}$

če je M nedeterministično, in

$T_M(n) = \max_{x \in \Gamma^*, |x|=n} \{k;$ *k je dolžina izračuna, ki se ustavi pri vhodu* $x\}$

če je M deterministično.

$T_M(n) = \infty$, če obstaja $x \in \Gamma^*$, $|x| = n$ tak, da se M ne ustavi pri vhodu x.

Ker je stara definicija $T_M(N)$ v primeru NTS strožja od nove, bi se lahko zgodilo, da bi se razred NP zdaj zajemal manj jezikov. Z NP_1 za hip označimo razred jezikov, ki je definiran tako, da v definiciji 4. uporabimo novo definicijo $T_M(N)$. Očitno je

$$NP_1 \supseteq NP$$

Na srečo pa velja tudi obratno, torej:

6.IZREK: $NP_1 = NP$

DOKAZ: Dokazati moramo $NP_1 \subseteq NP$.

Naj bo $L \in NP_1$. Torej obstaja NTS M s polinomsko časovno zahtevnostjo v smislu definicije 5, ki razpoznava L. Konstruirali bomo NTS N, ki bo razpoznaval isti jezik in bo imel polinomsko časovno zahtevnost v smislu definicije 3. Bodi $p(n) = T_M(n)$ časovna zahtevnost NTS M, kjer je p neki polinom.

TS N naj deluje takole:

- simulira delovanje NTS M in šteje korake
- ko doseže število korakov $p(n)$, se N ustavi
- če je v tem času M razpoznal vhod, ga razpozna tudi N, v vseh drugih primerih pa N odgovori NE.

N očitno razpoznava natanko L, saj po predpostavki za vsak element iz L obstaja sprejemajoč izračun z ne več kot $p(n)$ koraki.

Ker simulacija gre v polinomskem času, je časovna zahtevnost NTS N

očitno polinomska v smislu definicije 5.

Q.E.D.

Ker je vsak deterministični TS hkrati nedeterministični TS je očitno

$$P \subseteq NP$$

Prišli smo do znanega odprtega problema: ali morda velja P=NP? Prevladuje splošno prepričanje (ki seveda ne dokazuje ničesar), da P=NP ne velja. Tukaj navajamo dva 'zdravorazumska' argumenta:

a) Obstajajo problemi (takoimenovani NP–polni, definicija pride kasneje), na primer problem izpolnjivosti (SAT), za katerega velja

$$P = NP \iff SAT \in P$$

Problemov s to lastnostjo je še veliko, prihajajo pa z različnih področij. Iz teorije grafov omenimo problem klike in problem barvanja točk grafa. Razvpiti problem trgovskega potnika je samo eden izmed težkih (beri NP–polnih) transportnih problemov v operacijskih raziskavah. Mnogo dela mnogih raziskovalcev je bilo brez uspeha, ko so iskali učinkovite (polinomske) algoritme za te probleme. Še več,

b) za mnoge algoritme, ki so jih predlagali, se je izkazalo, da imajo več kot polinomsko časovno zahtevnost.

Za konec razdelka navedimo še izrek, ki primerja deterministično in nedeterministično časovno zahtevnost. Enostavna simulacija da za eksponentni red večjo časovno zahtevnost pri simulaciji nedeterminističnega TS z determinističnim. Boljše simulacije ne poznamo. Če velja $P \neq NP$ potem je to tudi najboljše, kar lahko dosežemo.

7.DEFINICIJA: Funkcija $T : N \to N$ je *časovna funkcija*, če obstaja deterministični TS, ki se ustavi po natanko $T(n)$ korakih pri vsakem vhodu dolžine n.

Nekatere običajne funkcije so tudi časovne funkcije, na primer $T(n) = n$, $T(N) = n\lfloor \log n \rfloor$, $T(n) = n^2$, $T(n) = 2^n$.

8.IZREK: (Simulacija nedeterminističnega TS z determinističnim) Bodi $T(n)$ časovna funkcija, $L \subseteq \Gamma^*$ jezik in N nedeterministični TS, ki sprejema L v času $T_N(n) = O(T(n))$. Potem obstaja deterministični TS M z $L(M) = L$ in $T_M(n) = O(c^{T(n)})$ za neko konstanto c.

DOKAZ: Za vsak $x \in L$ obstaja izračun dolžine $\leq T_N(|x|)$, ki sprejme x. Bodi $k = \max\{card(\delta(z,a)); z \text{ stanje TS } N \text{ in } a \in \Gamma\}$. Potem ima TS N na vsakem koraku največ k različnih mogočih operacij. Torej je največ $k^{T_N(|x|)}$ različnih izračunov, ki se ustavijo, dolžine $\leq T_N(|x|)$ pri vhodu x. Spomnimo se, da je $x \in L$ natanko tedaj, ko je vsaj eden od izračunov, ki se ustavijo, sprejemajoč. Uporabimo to pri simulaciji! Izberimo tak m, da bo $T_N(n) \leq mT(n)$ za vsak n. Poglejmo cela števila od 0 do $k^{mT(n)}$ pri osnovi k. Vsaka od mogočih izvedb TS N je zakodirana kot eno od teh števil na naslednji način: i-ta cifra v k-adičnem zapisu števila določa operacijo na i-tem koraku izvedbe. (Nekatera števila ne predstavljajo možne izvedbe, vendar nas to ne moti.) Enostavno je videti, da je vsako konkretno izvedbo TS N potrebno največ $p(mT(|x|))$ korakov, za neki polinom p. (Res, dodatno moramo na vsakem koraku poiskati naslednjo cifro zakodirane izvedbe, za to se moramo največ dvakrat zapeljati čez trak.) Torej celotna simulacija zahteva čas $p(mT(|x|))k^{mT(|x|)} = O(c^{T(|x|)})$ za neko konstanto c.

Q.E.D.

4. Problemi, jeziki in optimizacijski problemi

Razreda P in NP smo definirali za jezike. Praktični problemi pa običajno niso formulirani v obliki problema razpoznavanja nekega jezika. V tem razdelku bomo pokazali zvezo med optimizacijskimi problemi in njihovimi odločitvenimi inačicami.

Za primer si poglejmo odločitveni problem barvanja točk grafa in ustrezni optimizacijski problem.

Problem: Optimizacijski problem barvanja točk grafa

Vhod: Graf G

Izhod: Dobro barvanje grafa G z $\chi(G)$ barvami

Optimizacijski problem barvanja točk grafa očitno ni jezik, pač pa zahteva izračun neke transformacije iz matrike $n \times n$ bitov v zaporedje n števil $\in \{1, 2, ..., \chi(G)\}$ (ki označujejo barve). Pripadajoči odločitveni problem je

Problem: Odločitveni problem pobarvljivosti grafa (k-COL)

Vhod: Graf G in celo število k

Vprašanje: Ali obstaja dobro barvanje grafa G z največ k barvami?

Odločitvenemu problemu je mogoče na naraven način prirediti jezik. To je množica vseh primerov problema (G, k) ali točneje zapisano wv, kjer je w koda grafa G in v koda števila k, za katere je odgovor na dano vprašanje pozitiven.

$L_{k-COL} = \{(G,k); G \text{ je } k\text{–pobarvljiv }\}$

Oziroma

$L_{k-COL} = \{\ wv;\ w$ je matrika povezav nekega grafa G in v je binarni zapis nekega celega števila k in graf G je k–pobarvljiv $\}$

Mimogrede smo privzeli, da je kodiranje *naših problemov* 'naravno'. Kot smo omenili že v uvodu poglavja, se dogovorimo, kako bomo kodirali kombinatorične objekte: naravna števila bomo zapisali binarno, splošne grafe pa z matriko sosednosti. Bistvena je zahteva, da kodirna shema ne sme umetno znižati zahtevnosti problema. Če bi na primer naravna števila zapisali unarno ($n = \underbrace{111\ldots1}_{n}$), bi namesto vhoda dolžine $O(\log n)$ imeli vhod dolžine $O(n)$. Tako bi eksponentni algoritem navidez postal polinomski! Podrobneje so zahteve 'naravnega' kodiranje opisane na primer v [PePl82].

Odločitvenemu problemu smo priredili jezik, torej se lahko vprašamo, ali je problem v P. V nadaljevanju bomo pokazali, da bi v tem primeru optimizacijski problem lahko rešili v polinomskem času. Torej lahko tudi za optimizacijski problem rečemo, da je polinomski, če je njegov pripadajoči odločitveni problem v razredu P.

Za začetek poglejmo karakteristično lastnost razreda NP.

9.IZREK:

NP $= \{L; L \subseteq \Sigma^*$ *za neko končno abecedo* Σ *in obstaja polinom* p *in polinomsko izračunljiv predikat* $Q(x,y) \subseteq \Sigma^* \times \Sigma^*$ *, da za vsak* $x \in \Sigma^*$ *velja:*

$$x \in L \iff \exists y \in \Sigma^* : |y| \leq p(|x|) \text{ in } Q(x,y)\}$$

Niz y imenujemo *certifikat*.

DOKAZ: a) Bodi p polinom in Q polinomsko izračunljiv predikat. Torej obstaja deterministični TS M, ki izračuna $Q(x,y)$ v času $q(x,y)$ za neki polinom q. Naslednji nedeterministični algoritem sprejme L.

(1) Ob vhodu x nedeterministično zgeneriraj zaporedje $y \in \Sigma^*, |y| \leq p(|x|)$

(2) Razpoznaj x, če je res $Q(x,y)$ (izračunano z M)

Nedeterministični algoritem v delu (1) je podoben algoritmu, s katerim smo v prvem razdelku reševali problem klike. Kot smo se prepričali že tam, je algoritem polinomski na nedeterminističnem TS, število korakov pa je omejeno s $p(|x|)$. Časovna zahtevnost drugem delu algoritma je omejena s polinomom q, ki je odvisen od dolžine vhoda $|x|$ in dolžine $|y|$, ki pa je spet omejena s polinomom $p(|x|)$. Torej je $L \in$ NP.

b) Bodi $L \in$ NP $L \subseteq \Sigma^*$. N naj bo nedeterministični TS, ki razpoznava L v času, omejenem s polinomom p. Kot v dokazu prevedljivosti nedeterminističnega TS na deterministični TS (Izrek 8) naj bo k največje število različnih operacij, ki jih na enem koraku lahko izbere N. V dokazu omenjenega

izreka smo ugotovili, da lahko z zaporedjem cifer v k-adičnem številskem zapisu natanko določimo (zakodiramo) vsako od možnih izvedb NTS pri vhodu x. Brez škode za splošnost vzemimo, da je $|\Sigma| \geq k$. Definirajmo

$Q(x,y)$ = 'y je koda sprejemajočega izračuna NTS N pri vhodu x'

Q je (po definiciji razreda NP) izračunljiv v polinomskem času in $L = \{x; \exists y |y| \leq p(|x|)\ in\ Q(x,y)\}$

Q.E.D.

Izrek daje možnost alternativne definicije razreda NP, ki ima vsaj eno prednost. S tako definicijo se izognemo razlaganju fenomena nedeterminizma, tu vse delo opravi en eksistenčni kvantifikator!

10.DEFINICIJA: *Minimizacijski* problem je podan s polinomsko izračunljivim predikatom $Q \subseteq \Sigma^* \times \Sigma^*$ in polinomsko izračunljivo *namensko* funkcijo $c : \Sigma^* \times \Sigma^* \longrightarrow N$. Če velja $Q(x,y)$, potem je y *dopustna* rešitev problema pri vhodu x s 'stroškom' $c(x,y)$. Če velja $Q(x,y)$ in $c(x,y) \leq c(x,y')$ za vsak y' za katerega velja $Q(x,y')$, potem je y *optimalna* rešitev za x. Minimizacijski problem je *polinomsko omejen*, če obstaja polinom p, tako da iz $Q(x,y)$ sledi $|y| \leq p(|x|)$.

V tem delu bomo obravnavali samo polinomsko omejene probleme. Podobno kot minimizacijske bi lahko definirali maksimizacijske optimizacijske probleme. V primeru optimizacijskega problema barvanja točk grafa je $Q(x,y)$ izpolnjen, če je x koda grafa, y pa koda dobrega barvanja. $c(x,y)$ je število barv, uporabljeno v barvanju, ki ga kodira y.

11.DEFINICIJA: Bodi (Q,c) polinomsko omejen minimizacijski problem. Definirajmo štiri verzije problema:

a) **Problem:** (Q,c)-odločitveni problem
Vhod: $x \in \Sigma^*$, celo število C
Vprašanje: Ali obstaja y, da velja $Q(x,y)$ in $c(x,y) \leq C$?

b) **Problem:** (Q,c)-optimizacijski problem
Vhod: $x \in \Sigma^*$
Izhod: Optimalna rešitev $optsol(x) \in \Sigma^*$ za x

c) **Problem:** (Q,c)-problem optimalne vrednosti
Vhod: $x \in \Sigma^*$
Izhod: $optval(x) = c(x, optsol(x))$

d) **Problem:** (Q,c)-problem C-rešitve
Vhod: $x \in \Sigma^*$, celo število C
Izhod: $y = witness(C)$, da velja $Q(x,y)$ in $c(x,y) \leq C$, če tak y obstaja.

Za primer barvanja točk grafa smo prvi dve verziji problema že videli. Problem iskanja optimalne vrednosti v primeru barvanja točk grafa pomeni izračun kromatičnega števila $\chi(G)$, problem C-rešitve pa formuliramo takole: pri danem C poišči dobro C-barvanje, če (seveda) obstaja.

Primerjajmo časovno zahtevnost štirih verzij problema. Očitno problem razpoznavanja ni težji od problema C-rešitve in problema optimalne vrednosti, ta dva pa nista težja od optimizacijskega problema. Ali bolj nazorno:

problem razpoznavanja $\propto$ problem C-rešitve $\propto$ optimizacijski problem
problem razpoznavanja $\propto$ problem optimalne vrednosti $\propto$ optimizacijski problem

Natančneje formuliramo pravkar povedano z naslednjimi trditvami:

12.LEMA: Bodi (Q,c) polinomsko omejen optimizacijski problem. Potem velja:

a) Če je funkcija $optsol : \Sigma^* \longrightarrow \Sigma^*$ izračunljiva v polinomskem času, potem isto velja za funkciji $optval : \Sigma^* \longrightarrow \Sigma^*$ in $witness : \Sigma^* \longrightarrow \Sigma^*$.

b) Če je vsaj ena od funkcij *witness* in *optval* izračunljiva v polinomskem času, potem je

$$L_{(Q,C)} := \{(x,C); x \in \Sigma^*, C \in N in\ optval \leq C\} \in P$$

DOKAZ: Očitno iz prej povedanega.

Manj očitno je, da velja tudi obrat zadnje leme. V splošnem znamo dokazati nekoliko šibkejšo obratno trditev. Primer, ki ga v splošnem ne znamo dokazati, pa bomo dokazali za poseben primer barvanja točk grafa.

13.LEMA: Bodi (Q,c) polinomsko omejen optimizacijski problem.

a) Če sta funkciji *witness* in *optval* izračunljivi v polinomskem času, potem isto velja za *optsol*.

b) Če je problem razpoznavanja v razredu P in za neki polinom q velja $optval(x) \leq 2^{q(|x|)}$ za vse x, potem je funkcija *optval* izračunljiva v polinomskem času.

DOKAZ:

a) Trivialno, saj za vsak x velja $optsol(x) = witness(x, optval(x))$.

b) Za izračun optimalne rešitve optval bomo uporabili binarno iskanje med vrednostmi $\{1, ..., 2^{q(|x|)}\}$. Poglejmo naslednji algoritem:

```
low := 0;
high := 1;
while x nima rešitve ≤ high do high := 2 * high;
while high − low ≥ 1
    do middle := trunc((high + low)/2)
    if x ima rešitev ≤ middle
      then high := middle
      else low := middle;
    od
optval(x) := low;
```

Ker je problem po predpostavki polinomsko omejen, vemo, da je rešitev omejena z $2^{q(|x|)}$, kjer je q neki polinom. Prva **while** zanka torej v polinomskem številu korakov določi zgornjo mejo rešitve.

Hitro vidimo, da je $low \leq optval(x) \leq high$ invarianta do zanke. Torej gornji algoritem pravilno izračuna vrednost $optval(x)$ v $log_2(2^{q(|x|)}) = q(|x|)$ ponovitvah zanke. Če v drugi **while** zanki uporabimo polinomski algoritem za razpoznavanje, ki po predpostavki obstaja, smo torej res našli polinomski algoritem za izračun optimalne vrednosti $optval(x)$.

Q.E.D.

Primerjavo med problemom C-vrednosti in problemom razpoznavanja naredimo za konkretni primer barvanja. V knjigi [Melh84] najdemo dokaza za problem trgovskega potnika in za izpolnjivost, pa tudi trditev, da podobni dokazi, v katerih uporabimo tehniko redukcije problema na manjši problem iste vrste, obstajajo tudi za mnoge druge NP-polne probleme.

14.LEMA: Če je odločitveni problem C-barvanja v P potem obstaja algoritem, ki v polinomskem času poišče C-barvanje.

DOKAZ: Idejo iz [PePi82] zapišimo v obliki algoritma:

```
if not C − COL(G) then (*barvanja ni*)
else
for c := C downto 2 do begin
    izberi poljubno točko u ∈ V(G');
    pobarvaj u z barvo c;
    for all točke v ∈ V(G'), ki niso sosede u-ja do
        if not c − COL(V(G'), E(G') ∪ (u, v)) then begin
          (*točki sta neodvisni!*)
          pobarvaj v z barvo c;
          identificiraj u in v v G';
        end;
    G' := G' \{u}
end
preostale točke v G' pobarvaj z barvo 1
```

Identifikacijo dveh točk grafa očitno lahko naredimo v linearnem času (potrebno je namesto dveh vrstic (stolpcev) enega pobrisati, v drugega pa vpisati konjunkcijo vrstic (stolpcev)), zato je gornji algoritem polinomski, seveda pri predpostavki, da je algoritem, ki odloči, ali je graf c-pobarvljiv, polinomski.

Q.E.D

5. Polinomsko prevajanje in NP–polni problemi

Prevajanje problemov je uporabna tehnika za ugotavljanje 'težavnosti' problemov. (Primer smo videli v prejšnjem razdelku.)

15.DEFINICIJA: a) Naj bosta Σ in Γ končni abecedi. Preslikava $f : \Sigma^* \to \Gamma^*$ je *polinomska transformacija*, če lahko f izračunamo v polinomskem času na determinističnem TS.

b) Bodita $L_1 \subseteq \Sigma^*$ in $L_2 \subseteq \Gamma^*$ jezika. L_1 je *polinomsko prevedljiv* na L_2, oznaka $L_1 \propto L_2$, če obstaja polinomska transformacija f, tako da je

$$x \in L_1 \iff f(x) \in L_2$$

za vse $x \in \Sigma^*$.

c) Jezik L je NP–poln, če velja

1) $L \in$ NP

2) $L' \propto L_1$ za vse $L' \in$ NP

Pomembnost definicije pojasni naslednji

16.IZREK: Bodi L_0 NP–poln. Potem

a) $L_0 \in$ P $\iff$ P = NP

b) če je $L_0 \propto L_1$ in $L_1 \in$ NP, potem je L_1 NP–poln

DOKAZ: a) Če je P=NP potem je gotovo $L_0 \in$ P. Dokazati moramo še drugo smer ekvivalence. Bodi $L_0 \in$ P in $L \in$ NP poljuben. Ker je $L_0 \in$ P, obstaja deterministični TS M, ki sprejme L_0 v času omejenem z nekim polinomom p. Ker je L_0 NP–poln in $L \in$ P, velja $L \propto L_0$. Torej obstaja polinomsko izračunljiva preslikava f, tako da je $L = f^{-1}(L_0)$. Bodi N deterministični TS, ki izračunava f v času, omejenem s polinomom q. Iz TS M in N konstruirajmo nov TS A, ki bo razpoznaval L takole:

(1) Izračunaj $f(x)$ v času $q(|x|)$ s TS N.

(2) Premakni glavo ('oko') na prvi simbol $f(x)$ v času $|f(x)|$

(3) Razpoznaj $f(x)$ uporabljajoč M v času $p(|f(x)|)$. Če je $f(x) \in L_0$, potem sprejmi x, sicer ga zavrni.

Gornji algoritem očitno razpoznava L. Potreben čas je $q(|x|) + |f(x)| + p(|f(x)|) \leq r(|x|)$ za neki polinom r. Zadnja neenakost je posledica dejstva, da je $|f(x)| \leq |x| + q(|x|)$, saj lahko TS v enem koraku porabi največ en nov kvadratek traku.

b) Pokazati moramo, da je $L \propto L_1$ za vsak $L \in$ NP. Vzemimo poljuben $L \in$ NP. Ker je L_0 NP–poln, obstaja polinomska transformacija f, da je $L = f^{-1}(L_0)$. Ker je $L_0 \propto L_1$ obstaja tudi polinomska transformacija g, tako da je $L_0 = g^{-1}(L_1)$. Označimo $h = g \circ f$. Potem je $L = f^{-1}(L_0) = f^{-1}(g^{-1}(L_1)) = (g \circ f)^{-1}(L_1) = h^{-1}(L_1)$. S podobnimi argumenti kot v točki a) pokažemo, da je h polinomska transformacija.

Q.E.D.

NP–polni problemi so torej najtežji med problemi razreda NP. Če bi bil eden izmed njih rešljiv v polinomskem času, potem bi bili vsi rešljivi v polinomskem času in veljalo bi P=NP. Po drugi strani pa velja: iz N≠NP sledi, da noben NP–poln problem ni rešljiv v polinomskem času.

Točka b) nam da enostavno tehniko za dokaz NP–polnosti danega problema. Če želimo pokazati, da je L NP–poln, moramo dokazati $L \in$ NP in $L_0 \propto L$ za nek problem L_0, za katerega že vemo, da je NP–poln.

Prvi dokaz NP–polnosti nekega problema je naredil Cook v svojem klasičnem članku [Cook71]. Dokazal je, da je problem izpolnjivosti NP–poln. Dokaz tu opuščamo, bralec ga lahko najde v knjigah, na primer v [GaJo79] ali [Mejh84], v slovenščini pa v [PePi82]. Ko tak dokaz imamo, lahko relativno enostavno pokažemo NP–polnost drugih problemov, kot je prvi storil Karp [Karp72]. Seznam NP–polnih problemov se od tedaj širi, zajetne sezname lahko bralec najde v omenjenih knjigah.

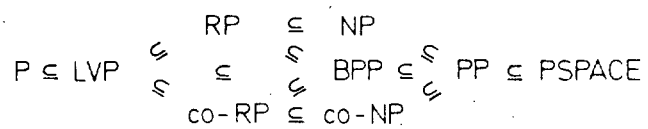
Za problem *k-COL* (odločitveni problem barvanja točk grafa) je dokazal NP–polnost že Karp [Karp72]. Dokaz je posreden, vmes pokaže NP–polnost problema 3-izpolnjivosti (*3-SAT*). Dokaz tu opuščamo, bralec ga lahko v slovenščini najde v [Gode83]. Pravkar povedano zapišimo v obliki izrekov:

17.IZREK: Problem izpolnjivosti je NP–poln.

18.IZREK: *SAT* $\propto$ *3-SAT* $\propto$ *k-COL*

6. Verjetnostni model računanja

V tem razdelku bomo definirali še nekaj novih razredov časovne zahtevnosti algoritmov. Namesto načrta si poglejmo naslednji diagram:



Slika 1. Nekateri razredi jezikov

kjer je co-C:={$\Gamma^* - L; L \in$ C *in* Γ *končna abeceda*} za vsak razred jezikov C.

Nove razrede problemov bomo definirali s pomočjo verjetnostnega Turingovega stroja. Verjetnostni Turingov stroj je TS z možnostjo slučajnih odločitev. Tukaj samo omenimo, da s tem računska moč stroja ni nič večja. Mogoče pa je, da se lahko kak problem reši hitreje ali na manjšem prostoru, kot z determinističnim Turingovim strojem.

19.DEFINICIJA: *Verjetnostni Turingov stroj* (VTS) je DTS s posebnimi stanji, v katerih sta mogoči dve nadaljevanji, ki sta enako verjetni.

Taka definicija VTS je ugodna zaradi enostavnosti. Ker so vse veje enako verjetne, dobimo enostavno porazdelitev možnih rezultatov. Model, v katerem bi dovolili neuravnotežene verjetnosti nadaljevanj, ni nič močnejši od tukaj definiranega [Schø86].

V splošnem VTS računa slučajno funkcijo: za vsak vhod x izračuna VTS rezultat y z verjetnostjo $Pr\{M(x) = y\}$.

20.DEFINICIJA: Delna funkcija, ki jo izračunava VTS M je definirana z

$$\phi_M(x) = \begin{cases} y & ,\text{če } Pr\{M(x) = y\} > \frac{1}{2} \\ nedefinirano & ,\text{če takega } y \text{ ni} \end{cases}$$

21.DEFINICIJA: *Verjetnost napake* VTS M je

$$e_M(x) = \begin{cases} Pr\{M(x) \neq \phi_M(x)\} & ,\text{če je } \phi_M(x) \text{ definirano} \\ nedefinirano & ,\text{sicer} \end{cases}$$

22.DEFINICIJA: VTS M izračunava ϕ_M z omejeno verjetnostjo napake, če obstaja konstanta $\varepsilon < \frac{1}{2}$, tako da je $e_M(x) \leq \varepsilon$ za vsak x iz domene ϕ_M.

23.DEFINICIJA: *(Blumova) časovna zahtevnost* VTS M je

$$T_M(x) = \begin{cases} najmanjši\ n,\ da\ je \\ Pr\{M(x) = \phi_M(x)\} > \frac{1}{2} \\ v\ manj\ kot\ n\ korakih & ,\text{če je } \phi_M(x) \text{ definirano} \\ \infty & ,\text{sicer} \end{cases}$$

in

$$T_M(n) = \max_{|x| \leq n} T_M(n)$$

Zadnja definicija je analogna definiciji časovne zahtevnosti NTS kot dolžine najkrajšega sprejemajočega izračuna (definicija 5.).

24.DEFINICIJA: VTS je *polinomsko omejen*, če obstaja polinom p, tako da se vsak možen izračun vhodov dolžine n ustavi po največ $p(n)$ korakih.

25.DEFINICIJA: VTS razpoznava jezik, če izračuna njegovo karakteristično funkcijo.

Ali, drugače zapisano: $x \in L \Leftrightarrow Pr\{M(x) = 1\} > \frac{1}{2}$ in $x \notin L \Leftrightarrow Pr\{M(x) = 0\} > \frac{1}{2}$

26.DEFINICIJA:

PP = 'razred jezikov, ki jih razpoznavajo polinomsko omejeni VTS'

BPP = 'razred jezikov, ki jih razpoznavajo polinomsko omejeni VTS z omejeno verjetnostjo napake'

ZPP (ali LVP) = 'razred jezikov, ki jih razpoznavajo VTS s polinomsko povprečno časovno zahtevnostjo in z verjetnostjo napake 0'

27.IZREK: LVP $\subseteq$ BPP $\subseteq$ PP $\subseteq$ PSPACE

S PSPACE smo označili razred jezikov, ki jih razpoznavajo TS s polinomsko omejeno dolžino uporabljenega traku. Izkaže se, da je vseeno, če vzamemo v definiciji deterministični ali nedeterministični TS [Savi74].

DOKAZ: Relacija BPP $\subseteq$ PP je očitna iz definicije. PP $\subseteq$ PSPACE je posledica izreka o deterministični simulaciji verjetnostnega TS, ki ga tu nismo navedli. Dokaz bi šel podobno kot dokaz izreka 8.

Pokažimo še LVP $\subseteq$BPP. Naj bo L jezik, ki ga razpoznava VTS M z verjetnostjo napake 0 in povprečnim časom omejenim s polinomom $p(n)$. Bodi $c > 2$ poljubna konstanta. Z M' označimo VTS, ki simulira delovanje VTS M do največ $cp(n)$ korakov. Če se M do tedaj ne bi ustavil, M' odgovori karkoli. Ker M naredi več kot $cp(n)$ korakov z verjetnostjo manj kot $1/c$, je verjetnost napake polinomsko omejenega stroja M' največ $1/c < 1/2$.

Q.E.D.

28.IZREK: P $\subseteq$ LVP $\subseteq$ NP $\subseteq$ PP

DOKAZ: Ker vsak polinomsko omejeni TS računa z verjetnostjo napake 0, je očitno P$\subseteq$ LVP.

Bodi L $\in$LVP in M VTS, ki razpoznava L z verjetnostjo napake 0 in polinomsko omejenim povprečnim časom. Če na M gledamo kot na nedeterministični TS vidimo, da M razpozna L v polinomskem času, saj mora biti za vsak vhod vsaj en izračun s polinomsko omejenim številom korakov. Torej $L \in$ NP in LVP $\subseteq$ NP.

Pokažimo še NP $\subseteq$ PP. Brez škode za splošnost lahko privzamemo, da L razpoznava polinomsko omejen NTS, ki ima v vsakem stanju največ dva mogoča koraka. Če na M gledamo kot na verjetnostni stroj, potem je L množica nizov, za katere obstaja sprejemajoč izračun. Torej $x \in L \Leftrightarrow \Pr\{M(x)\ sprejme\} > 0$. Za dokaz, da je L v razredu PP konstruiramo stroj M' takole: M' najprej vrže kovanec; če je rezultat grb, potem sprejme vhod, sicer pa simulira delovanje stroja M, tako da vsakič, ko ima na voljo dve nadaljevanji, izbere eno ali drugo z enako verjetnostjo ($\frac{1}{2}$).

Vendar verjetnostni stroj M' še ni povsem dober. Če $x \notin L$ je mogoče, da velja $Pr\{M'(x)\ sprejme\} = 1/2$. Torej je mogoče, da M' ne izračuna karakteristične funkcije L. Definirajmo VTS M'', za katerega bo veljalo $Pr\{M''(x)\ sprejme\} < 1/2$ za vse x-e, ki niso v L.

Bodi $p(n)$ polinom, ki omejuje število korakov izvajanja stroja M. Vsak $x \in L$ dolžine n je sprejet z verjetnostjo vsaj $2^{-p(n)}$, saj gotovo obstaja vsaj en sprejemajoč izračun in je verjetnost vsakega od izračunov dolžine n vsaj $2^{-p(n)}$. Verjetnostni TS M'' deluje takole: Najprej M'' vrže $p(n)+1$ kovanec in sprejme vhod brez nadaljevanja z verjetnostjo $\frac{1}{2} - 2^{-p(n)}$. Potem M'' simulira delovanje M in sprejme vhod, če ga sprejme M. M'' torej zavrne vsak $x \notin L$ z verjetnostjo vsaj $\frac{1}{2} + 2^{-p(n)-1}$ in sprejme vsak $x \in L$ z verjetnostjo vsaj $\frac{1}{2} + 2^{-p(n)-1}$. Torej VTS M'' razpoznava L v polinomskem času, zato NP$\subseteq$PP.

Q.E.D.

29.DEFINICIJA: VPP (ali RP) je razred jezikov, ki jih razpoznajo polinomsko omejeni VTS, ki imajo za vhode, ki niso v jeziku, verjetnost napake 0.

Ali, drugače zapisano:

$$L \in RP \Leftrightarrow L\ razpoznava\ polinomsko\ omejeni\ VTS\ M\ in$$
$$Pr\{M(x)\ sprejme\} = 0\ za\ \forall x \notin L.$$

Opomba: Tu bi dobili isti razred, če bi za $x \in L$ zahtevali $Pr\{M(x) sprejme\} \geq q$ za katerokoli konstanto $0 < q < 1$. Po n ponovitvah algoritma je namreč verjetnost napake enaka $1 - \binom{n}{0}(1-q)^n$.

30.IZREK:

1.) RP $\subseteq$ NP $\cap$ BPP

2.) LVP = RP $\cap$ co-RP

DOKAZ: 1.) Inkluziji RP $\subseteq$ BPP in RP $\subseteq$ NP sta očitni.

2.) Pokazali bomo LVP $\subseteq$ RP, LVP $\subseteq$ co $-$ RP in RP $\cap$ co $-$ RP $\subseteq$ LVP.

Najprej pokažimo LVP $\subseteq$ RP. Če je $L \in$ LVP, lahko L razpoznamo TS M, katerega pričakovani čas izvajanja je omejen z nekim polinomom, imenujmo ga q, rezultati pa so popolnoma zanesljivi. Konstruirajmo TS M' takole: - pri vhodu x simulira delovanje TS M največ $q(|x|)$ korakov. Če bi se M ustavil, preden se ustavi M' (to se zgodi z verjetnostjo vsaj 1/2), potem M' odgovori isto kot M. Sicer M' odgovori NE. Vidimo, da so pritrdilni odgovori M' povsem zanesljivi, negativni odgovori pa so pravilni z verjetnostjo 1/2. Torej je $L \in$ RP in LVP $\subseteq$ RP.

Podoben argument pokaže LVP $\subseteq$ co − RP.

Ostala nam je še inkluzija RP $\cap$ co − RP $\subseteq$ LVP. Bodi $L \in$ RP $\cap$ co − RP. Potem obstajata VTS M_1 in M_2, za katera velja:

- čas izvajanja obeh VTS je omejen s polinomom
- pozitivni odgovori M_1 in negativni odgovori M_2 so povsem zanesljivi
- negativni odgovori M_1 in pozitivni odgovori M_2 so pravilni z verjetnostjo, ki je večja od neke konstante, na primer 1/2.

Pokažimo, kako lahko konstruiramo TS z verjetnostjo napake 0 in s polinomskim povprečnim časom izvajanja. Bodi x poljuben. Simulirajmo delovanje M_1 in delovanje M_2, kar lahko naredimo v polinomskem času. Če je vsaj en odgovor zanesljiv, končamo. Zanimiv je primer, ko oba algoritma dasta odgovor, ki ni povsem zanesljiv. (verjetnost tega dogodka je manjša od 1/4). V tem primeru poskus ponovimo, dokler se ne zgodi prvi primer. Pričakovano število ponovitev poskusa je omejeno $(\sum_{i=1}^{\infty}(1/4^i))$, zato je povprečni potrebni čas omejen s polinomom.

Q.E.D.

Za konec navedimo še nekaj primerov. Začnimo s problemom določitve, ali je neko število praštevilo.

Problem: Praštevila ($PRIMES$)

Vhod: Naravno število n v binarnem zapisu.

Odgovor: Da, če je n praštevilo in ne, če ni.

31.IZREK:

a) $PRIMES \in$ co − NP

b) $PRIMES \in$ co − RP

DOKAZ: a) trivialno

b) Naslednji algoritem (Solovay-Strassen) dokazuje, da je problem $PRIMES$ v razredu co-RP.

izberi $a \in \{1, 2, \ldots, n-1\}$

če $(a, n) \neq 1$ **potem** n ni praštevilo

če $(a/n) \not\equiv a^{\frac{n-1}{2}} (mod\ n)$ **potem** n ni praštevilo

sicer n je (z verjetnostjo $> 1/2$) praštevilo

kjer je (a/n) Legendrov simbol (prim. [Gras75]).

$$(a/n) = \begin{cases} 1 & \exists x : a \equiv x^2 (mod\ n) \\ -1 & a \not\equiv x^2 (mod\ n) \forall x \end{cases}$$

Legendrov simbol znamo učinkovito (v polinomskem času) izračunati z algoritmom, ki je nekoliko podoben Evklidovemu. Temelji na recipročnem zakonu, ki pravi, da je $(q/p) = -(p/q)$, če je $p = q = 3$ in $(q/p) = (p/q)$ sicer. Poleg tega za $q > p$, torej $q = mp + r$, velja $(q/p) = (r/p)$.

Če je p praštevilo, je kongruenca

$$(a/p) \equiv a^{\frac{p-1}{2}} (mod\ p)$$

izpolnjena za vsa števila a, $1 \leq a \leq p-1$. Če pa p ni praštevilo, pa kongruenca velja za največ polovico a-jev [SoSt77].

Q.E.D.

Neodvisno je podoben dokaz našel Rabin [Rabi76].

Kot drugi primer si poglejmo naslednja problema:

Problem: MAJ

Vhod: (KNO) formula stavčnega računa $F(x_1, \ldots, x_n)$

Vprašanje: Ali F velja za večino (za več kot 2^{n-1}) naborov vrednosti za $x_1, \ldots, x_n$?

Problem: $\#SAT$

Vhod: (KNO) formula stavčnega računa $F(x_1, \ldots, x_n)$ in naravno število i

Vprašanje: Ali obstaja vsaj i različnih naborov vrednosti za spremenljivke $x_1, \ldots, x_n$, ki zadoščajo F.

KNO je kratica za konjuktivno normalno obliko.

Brez dokaza navedimo (glej npr. [Gill77])

32.IZREK: MAJ in $\#SAT$ sta PP−polna problema.

7. Reševanje NP−polnih problemov

Doslej najboljša ocena za čas, potreben za reševanje NP- polnega problema je eksponentna funkcija dolžine vhoda. Pri velikih n je stvar videti brezupna. Po drugi strani so mnogi praktični problemi dokazano NP−polni. Kaj storiti? Poglejmo nekaj splošnih pristopov:

a) Posebni primeri: Pogosto se zgodi, da v praksi ne potrebujemo rešiti NP−polnega problema v vsej splošnosti. Podproblemi imajo zaradi dodatnih omejitev včasih polinomske rešitve.

b) Dinamično programiranje in razveji−omeji sta tehniki, ki ju je mogoče uporabiti pri reševanju nekaterih NP−polnih problemov. V obeh primerih s pametno izbiro nadaljevanja precej zmanjšamo potrebno računanje na neperspektivnih rešitvah. Čas obeh metod pa je še vedno eksponenten. Več o omenjenih metodah najdemo na primer v knjigi [Koza86].

c) Z verjetnostno analizo lahko včasih pokažemo, da so resnično 'težki' primeri NP−polnega problema dokaj redki. V takem primeru lahko najdemo algoritem z dobrim pričakovanim časom računanja, čeprav zgornje meje ne moremo omejiti s polinomom. Seveda je potrebno paziti pri izbiri porazdelitve primerov NP−polnega problema. Dokaz, da je izbrana porazdelitev prava, pa utegne biti resen problem [Karp76].

Za primer omenimo metodo simpleksov za reševanje problema linearnega programiranja, ki ima eksponentno časovno zahtevnost. Metoda simpleksov se v praksi dobro obnese, verjetno zaradi izredne redkosti 'težkih' primerkov problema. Dokazano je namreč, da je metoda v povprečju polinomska [Smal83]. Kljub temu, da je problem linearnega programiranja v razredu P [Khac79], pa ne poznamo polinomskega algoritma, ki bi bil v praksi boljši od metode simpleksov.

d) Aproksimacijski algoritmi lahko včasih dajo zadovoljive rešitve v kratkem času. Seveda ni vseeno, kaj nam pomeni v konkretnem primeru zadovoljiva rešitev. V primeru barvanja točk grafa na primer velja, da bi bil tudi algoritem, ki bi poiskal skoraj optimalno rešitev NP−poln. Garey in Johnson sta namreč pokazala [GaJo76], da velja: če za kakšni konstanti $r < 2$ in d velja, da algoritem A poišče barvanje z $A(G) \leq r\chi(G) + d$ barvami v polinomskem času, potem obstaja algoritem, ki poišče $\chi(G)$-barvanje v polinomskem času.

e) Hevristični algoritmi dajejo dobre rezultate na kakšni podmnožici problemske domene. Pri deterministčnih hevrističnih algoritmih se običajno zgodi, da imajo Ahilove pete: na nekaterih primerih se obnašajo zelo slabo. Če nekatere korake naredimo slučajno, se običajno izognemo Ahilovim petam na račun nekaj večje (pričakovane) časovne zahtevnosti. Omenimo nekaj hevrističnih algoritmov za problem barvanja točk grafa:

- Corneil-Grahamov hevristični algoritem temelji na algoritmu Zykova [CoGr73,Gode83].

- Veliko hevrističnih algoritmov spada v skupino, imenovano postopki zaporednega barvanja [Bata80]. Opišemo jih z naslednjo algoritemsko shemo:

pobarvaj točke grafa s 'prazno' barvo

dokler $\exists$ *točka, pobarvana s prazno barvo* **ponavljaj**

izberi barvo za točko v

Če določimo vrstni red barvanja točk in strategijo izbire barve, dobimo algoritem. Če na primer točke izbiramo po padajočih stopnjah in izbrano

točko vsakič pobarvamo z najnižjo barvo, ki je še prosta (noben sosed še ni pobarvan s to barvo), dobimo Welsh–Powellov postopek. Za ta postopek je mogoče pokazati, da porabi največ $d+1$ barv, kjer je d največja stopnja točke v grafu. Vendar (kar pa ni presenetljivo) obstaja družina grafov, na kateri je razmerje med številom barv, ki jih porabi Welsh- Powellov postopek in med dejansko potrebnim številom barv $\chi(G)$ poljubno veliko [WePo67]. V [Žero88a] je podana konstrukcija take družine grafov.

algoritem (Welsh–Petford), ki ga obravnavamo v [Žero87,Žero88,Žero88a], temelji na protivolilnem modelu delcev z interakcijo iz statistične mehanike (percolation theory). Je primer verjetnostnega hevrističnega algoritma. Gre za iterativno (slučajno) lokalno 'popravljanje' danega barvanja, ki z verjetnostjo 1 v končno mnogo korakih doseže neko dobro barvanje, če smo na začetku izbrali vsaj $\chi(G)$ barv. Če algoritem po vnaprej določenem številu korakov nasilno prekinemo, se nam lahko zgodi, da ne dobimo dobrega barvanja, čeprav to obstaja. Vemo, da je pričakovani čas absorbcije na regularnih grafih reda $O(n^2)$. Poleg tega se algoritem na testiranih grafih obnaša zelo dobro. Zato algoritem z vgrajeno nasilno prekinitvijo (s polinomsko mejo dovoljenega števila korakov) apliciramo na 'vse' grafe in upamo, da bodo rezultati še vedno dobri.

8. Literatura

AhHU76] A.V.Aho, V.E.Hopcroft, J.D.Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley 1976

ApHa86] K.Appel, W.Haken: The Four color Proof Suffices, The Mathematical Intelligencer, Vol. 8, 1986, No. 1, p.10-20

[Bata80] V.Batagelj: Barvanja točk grafov, Seminar za računalniško in numerično matematiko 201, Ljubljana 1980

[BaPi85] D.Bajc, T.Pisanski: Najnujnejše o grafih, Presekova knjižnica 22, DMFA SRS, Ljubljana 1985

*[Cook71] S.A.Cook: The Complexity of Theorem-proving Procedures, Proc. of 3^{rd} Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio, 151-158, (1971)

CoGr73] D.G.Corneil, B.Graham: An algorithm for determining the chromatic number of a graph, SIAM J. Comp. 2 (1973) 4, 311-318

CvMi77] R.Cvetković , M.Milić : Teorija grafova i njene primjene, Naučna knjiga Beograd, 1977

DoWe83] P.Donnelly, D.Welsh: Finite particle systems and infection models, Math. Proc. Camb. Phil. Soc (1983), 94, 167-182

DuBr81] R.D.Dutton, R.C.Brigham: A new graph coloring algorithm, The Computer Journal 24 (1981) 1, 85-86

GaJo76] M.R.Garey, D.S.Johnson: The Complexity of Near-Optimal Graph Coloring, JACM 23 (1976) 1, 43-49

GaJo79] M.R.Garey, D.S.Johnson: Computers and Intractability, W.H.Freeman and Co., (San Francisco) 1979

[Gill77] J.Gill: Computational complexity of probabilistic Turing machines, Siam. J. Comp. 6 (1977) 4, p.675-695

[Gras75] J.Grasselli: Osnove teorije števil, zbirka Sigma, DZS 1975

Gode83] H.Godec: Algoritmi za barvanje grafov, diplomsko delo, FNT Ljubljana 1983

[HoUl79] V.E.Hopcroft, J.D.Ullman: Introduction to automata theory, languages and computation, Addison-Wesley 1979

[HoUl86] V.E.Hopcroft, J.D.Ullman: Uvod v teorijo avtomatov,jezikov in izračunov, (prevod B.Vilfan), Fakulteta za elektrotehniko, Ljubljana 1986

[Karp72] R.M.Karp: Reducibility among combinatorial problems, v Complexity of Computer Computations (ur. Miller, Thatcher), Plenum Press, New York, 85-103, (1972)

[Karp76] R.M.Karp: The Probabilistic Analysis of some Combinatorial Search Algorithms, v Algorithms and complexity (ur. Traub), 1-20, Academic Press 1976

[Khac79] L.G.Khatchjan: LP in polynomial time, Doklady Akad. Nauk SSSR 244 (1979) p.1093-1096

[Knut81] D.E.Knuth: Algorithms in modern mathematics and computer science, Lecture Notes in Comp. Sci. 122, Springer-Verlag, Berlin 1981

[Koza86] J.Kozak: Podatkovne strukture in algoritmi, DMFA SRS, Ljubljana 1986

LLRS85] E.L.Lawler, J.K.Lenstra, A.H.G.Rinnooy Kan, D.B.Shmoys: The Traveling Salesman Problem, Wiley 1985

[Melh84] K.Melhorn: Graph Algorithms and NP-Completeness, Springer-Verlag, 1984

[PePi82] M.Petkovšek, T.Pisanski: Izbrana poglavja iz računalništva, I. del, DMFA SRS, Ljubljana 1982

[Pisa83] T.Pisanski: Izračunljivost in rešljivost, DMFA SRS, Ljubljana 1983

[Rabi76] M.O.Rabin: Probabilistic Algorithms, v Algorithms and Complexity (ur. Traub), 21-39, Academic Press 1976

*[Savi74] W.J.Savitch: Relationship between nondeterministic and deterministic tape complexities, Journal for Computer and System Sciences (1974), p.177-192

*[Smal83] Smale: On the average number of steps of the simplex method of linear programming, Math. Prog. 27 (1983) p.241-262

[SoSt77] R.Solovay, V.Strassen: A Fast Monte-Carlo Test for Primality, SIAM J. Comp., Vol. 6, No. 1, March 1977

[Schø86] U.Schøning: Complexity and Structure, Lecture Notes in Comp. Sci. 211, Springer-Verlag 1986

[Tros83] V.N.Trostnikov: Što su konstruktivni procesi u matematici, Moderna matematika, Školska knjiga Zagreb 1983

WePe83] D.J.A.Welsh, D.M.Petford: A Randomised Attack to an NP-Complete Problem, Univ. of Oxford (preprint), 1983

WePo67] D.J.A.Welsh, M.B.Powell: An upper bound for the chromatic number of a graph and its application to timetabling problems, The Computer Journal 10 (1967), 85-86

WoWi78] D.R.Woodall, R.J.Wilson: The Appel-Haken Proof of the Four-Colour Theorem, v Selected Topics in Graph Theory I, (ur. L.W.Beineke, R.J.Wilson), Academic Press 1978

[Žero87] J.Žerovnik: Poskusi s slučajnim hevrističnim algoritmom, Zbornik XI. simpozija iz informatike Jahorina 1987, 294-(1-10)

[Žero88] J.Žerovnik: Randomised Heuristical Algorithm for Graph Colouring, Proceedings of Eighth Yugoslav Seminar on Graph Theory, Novi Sad 1987, (sprejeto)

Žero88a] J.Žerovnik: Verjetnostni algoritem za barvanje grafa, magistersko delo, FNT Ljubljana 1988

Z * smo označili posredne reference.