Razredi kompleksnosti

Tomaž Dobravec, Algoritmi in podatkovne strukture 2

Odločitveni in optimizacijski problemi

- Ločimo med pojmoma problem in naloga (primer: problem=urejanje po velikosti, naloga= tabela števil)

V grobem ločimo dva tipa problemov:

- **Odločitveni problemi**; odgovor na nalogo odločitvenega problema je DA ali NE;

- **Optimizacijski problemi**: odgovor na nalogo optimizacijskega problema je na primer število, matrika, graf, ..., ki najbolje reši dano nalogo;

Optimizacija: imamo **kriterijsko** funkcijo f; med vsemi možnimi (**dopustnimi**) rešitvami (D) dane naloge iščemo tako, ki **optimizira** (min/max) funkcijo f.

Povezava med odločitvenimi in optimizacijskimi problemi

- **Med odločitvenimi in optimizacijskimi problemi** pogosto obstaja **povezava**: optimizacijski problem lahko rešim z večkratnim reševanjem ustreznega odločitvenega problema.

- Optimizacijski problem je na nek način ”težji” od pripadajočega odločitvenega problema.

- Če je odločitveni problem težek (ga ne znam hitro rešiti), bo težka tudi optimizacijska verzija.
- Obratno: če je optimizacijski problem lahek, je lahek tudi pripadajoči odločitveni problem.

Nekaj primerov optimizacijskih problemov in njihovih odločitvenih variant

- Barvanje grafov,
- obstoj Hamiltonovega cikla,
- postavitev šahovskih kraljic,
- ...

Primeri optimizacijskih problemov in njihovih odločitvenih variant

Zahtevnost problema

Kako zahteven je nek problem?

Primer: urejanje števil.

Poznamo različne algoritme: urejanje z mehurčki - $O(n^2)$, hitro urejanje - $O(n \log n)$, radix sort – $O(n)$, ...

Kakšno zahtevnost ima problem urejanja števil: kvadratično, n log n, linearno, ... ?

Definicija: problem je rešljiv v času `t(n)`, če obstaja algoritem, ki vsako nalogo tega problema velikosti `n` reši v času kvečjemu `t(n)`.

Zahtevnost problema je definirana z obstojom algoritma, ki ta problem rešuje!

Opomba:

- ✧ Nek problem lahko proglasimo za zelo zahteven samo zato, ker ne poznamo dobrega algoritma.

- ✧ Za nekatere probleme se da dokazati spodnjo mejo – ne samo, da mi ne poznamo boljšega algoritma, pokazati se da, da ta sploh ne obstaja.

Razredi kompleksnosti

Definicija:

- **razred `P`** ... razred odločitvenih problemov, za katere obstajajo algoritmi, ki vse naloge velikosti n rešijo v polinomskem času `p(n)`.
- **razred `PSPACE`** ... razred odločitvenih problemov, za katere obstajajo algoritmi, ki pri reševanju nalog velikosti `n` porabijo polinomsko mnogo prostora.
- **razred `EXPTIME`** ... razred odločitvenih problemov, za katere obstajajo algoritmi, ki vse naloge velikosti `n` rešijo v eksponentnem času `exp(n)`.

- Velja:
  - $P \subset EXPTIME$
  - $P \subset PSPACE$

Delitev problemov po zahtevnosti

- polinomske probleme (probleme iz `P`) štejemo za **lahke;**
- eksponentne probleme (probleme iz `EXPTIME\P`) štejemo za **težke;**

RAZRED NP

- Definirajmo še razred problemov, za katere sicer ne vemo, ali so lahki ali težki, vemo pa, da bi znali, če bi nam kdo ponudil rešitev, hitro preveriti, ali je rešitev prava.

Definicija: Razred **NP** (nedeterministično polinomski) je razred odločitvenih problemov, za katere znamo v polinomskem času preveriti pravilnost rešitve.

  - dobimo nalogo in rešitev in v polinomskem času rečemo DA, če je rešitev prava in NE, če ni
  - velja: $P \subseteq NP$

Primer problema, za katerega **na poznamo polinomskega algoritma**: problem Hamiltonovega cikla. Rešitve ne znamo poiskati, če pa nam jo nekdo ponudi, pa z lahkoto preverimo, da gre res za Hamiltonov cikel. Zato: $HAM \in NP$.

Poleg HAM obstaja veliko problemov, ki so v **NP** in za katere **ne poznamo polinomskega algoritma**:

- trgovski potnik (TSP),
- razbitje množice (Partition problem),
- klika (poln podgraf v grafu),
- vozliščno pokritje grafa,
- ...

Prevedbe

- **Prevedba**: nalogo problema A lahko rešim tako, da jo ”prevedem” na nalogo problema B, rešim nalogo problema B in rešitev prevedem nazaj v rešitev problema A.

Primer:

- Problem A: poišči najmanjši element zaporedja števil.
- Problem B: uredi zaporedje

Če prevedba ne vzame veliko časa, potem velja: če problem A prevedemo na problem B, potem je problem B težji (ali vsaj tako težak) kot problem A.

Opomba: lažji problem lahko prevedemo na težjega, obratno ne gre!

NP-polni problemi

- Polinomska prevedba: prevedba, za katero potrebujemo polinomsko mnogo časa (glede na velikost naloge).

Najtežje probleme iz razreda NP imenujemo NP-polni problemi:

Definicija: Problem P je **NP-poln**, če lahko vsak problem $P' \in NP$ polinomsko prevedemo na P.

- Vsi zgoraj našteti problemi (klika, vozliščno pokritje, trgovski potnik, razbitje množice) in še mnogi drugi so NP-polni problemi.
- Ker vsak NP problem lahko prevedemo na vsak NP-poln problem, velja: **če bi znali polinomsko rešiti katerikoli NP-poln problem, potem bi znali polinomsko rešiti VSE NP probleme!**

Posledica?

P = NP ?

Eno najbolj znanih še odprtih vprašanj teoretičnega računalništva se glasi: P = NP ?

Z drugimi besedami (spodnja tri vprašanja so ekvivalentna zgornjemu):

- ✧ Ali obstaja kakšen problem, ki je v NP in ni v P?
- ✧ Ali obstaja polinomski algoritem za katerikoli NP-poln problem?
- ✧ Ali obstaja dokaz, da za nek NP-poln problem polinomski algoritem ne obstaja?

Splošno prepričanje: *P≠NP*, vendar tega še nihče ni dokazal.

- ✧ Obstajajo tudi NP problemi, za katere se ne ve, ali so NP-polni ali ne. Recimo: celoštevilska faktorizacija. Nihče še ni našel polinomskega algoritma, zato se domneva, da problem ni v P. Gotovo je v NP (pravilnost rešitve zlahka preverim), ne ve pa se, ali je NP-poln.

- ✧ **V praksi:** če posumimo, da je problem težak (zaradi intuicije ali ker zaman iščemo polinomski algoritem), potem poiščemo prevedbo enega izmed znanih NP-polnih problemov na naš problem; če prevedbo najdemo, vemo, da je problem res težak in da (razen če je P=NP) zanj ne obstaja hiter algoritem.

- ✧ Karpov seznam 21 NP-polnih problemov (iz leta 1972!)

Reševanje NP-polnih problemov

- NP-polni problemi so torej problemi, ki so zelo težki (zanje ne poznamo polinomskih algoritmov); za njih so večinoma znani algoritmi z eksponentno časovno zahtevnostjo (in kot taki uporabni samo za majhne vhodne podatke).
  - Primer: klika

- Kako se torej lotevamo reševanja takih problemov?

- Nekaj pristopov za reševanje NP-polnih problemov.
  - lokalna optimizacija,
  - aproksimacijski algoritmi,
  - verjetnostni algoritmi,
  - hevristike in preiskovanja, ...

Razredi kompleksnosti za optimizacijske probleme

- Ko govorimo o razredih P, NP, PSPACE, ... imamo v mislih odločitvene probleme

- Ko govorimo o optimizacijskih problemih, govorimo o drugih razredih in sicer (zelo površno):

  - **PO** ... razred lahkih (polinomskih) optimizacijskih problemov
  - **NPO** ... razred optimizacijskih problemov, za katere lahko hitro preverimo pravilnost dane (uganjene) rešitve

  - najtežje NPO probleme imenujemo **NP-težki** (*angl. NP-hard*) problemi

- Če za optimizacijski problem vemo, da je NP-težak, potem zanj ni polinomskega algoritma
  - ali obratno: če bi za NP-težak problem našli polinomski algoritem, potem bi dokazali PO=NPO in tudi P=NP;

- Zveza med NP-polnimi in NP-težkimi problemi:

  Naj bo P_{odl} odločitvena varianta problema P, potem velja: P_{odl} je NP-poln $\Rightarrow$ P je NP-težak

Zahtevnost problema NI ENAKO zahtevnost algoritma

- Razred kompleksnosti za nek problem določimo na podlagi obstoja algoritma. Na primer: problem je v P, če obstaja algoritem, ki vse naloge tega problema reši v polinomskem času;
- Obratno ne velja! Na primer: če obstaja algoritem, ki vse naloge reši v eksponentnem času, to še ne pomeni, da problem ni v P; morda obstaja boljši (polinomski) algoritem, ki ga mi ne poznamo;
- Iz kakovosti algoritma ne smemo sklepati na zahtevnost problema; lahko pa postavimo zgornjo mejo.
- Za probleme obstaja več algoritmov, ki imajo lahko zelo različno časovno zahtevnost.

Primer: Problem največjega podzaporedja in več algoritmov za reševanje tega problema.

Problem največjega podzaporedja v zaporedju celih števil išče največje (t.j. z največjo vsoto) neprekinjeno podzaporedje.

Algoritmi za reševanje:	Časovna zahtevnost
1) Groba sila	$O(n^3)$
2) Izboljšana groba sila	$O(n^2)$
3) Algoritem tipa deli-in-vladaj	$O(n \log n)$
4) Kadanov algoritem	$O(n)$

Problem največjega podzaporedja - zaključek

- Videli smo, da za problem največjega podzaporedje obstaja več algoritmov z različno časovno zahtevnostjo: $O(n^3)$, $O(n^2)$, $O(n \log n)$ in $O(n)$.
- Kaj lahko rečemo za problem? Najmanj to, da je v razredu P.
- Ne moremo pa reči, da je linearni algoritem najboljši algoritem, ki obstaja za ta problem.
- Obstaja še boljši? Najbrž ne, a to bi bilo treba dokazati!

Je rekurzija vedno ugodna?

- Na primeru največjega podzaporedja smo videli, da je rekurzivna verzija algoritma veliko boljša od grobe sile. Pa to velja za vse probleme?
- NE, rekurzija ni nujno boljša od grobe sile!
- Primer: Hanoiski stolpiči.