Eerste deeltoets Algoritmiek (INFOAL)
05-10-2010
Uitwerkingen

(De opmaak is over de jaren verloren gegaan maar de antwoorden zijn nog correct en gecontroleerd door de docent)

Eerste deeltoets Algoritmiek, 5 oktober 2010

Opgave 1, Gelijke buur.
Gegeven: rij van n kabouters, waarbij n even is en K1 en Kn zijn jongen.
Tb: er staan twee kabouters van gelijk type naast elkaar.
(a) Dit is uit het ongerijmde te bewijzen.
Stel dat op een segment [i..j] GEEN gelijk paar te vinden is.
Dan staan op dat segment J en M kabouters afwisselend, wat impliceert dat
1e alle kabouters met een rangnummer dat EVEN verschilt van i, hetzelfde type hebben als i,
2e en alle kabouters waarvan het rangnummer ONEVEN verschilt van i, het andere type hebben.
Het gegeven zegt dat n-1 ONEVEN is, terwijl Kn hetzelfde type is als K1. Dit is tegenspraak met het afwisselend staan.

Een heel mooie, iets abstractere aanpak is deze. ALS er na kabouter 1 GEEN gelijk paar ZOU voorkomen, dan staan ze in type alternerend, dus DAN heb je: oneven=J, even=M.
Noem een kabouter "GEZOND" als het een oneven jongen of even meisje is, en "ZIEK" als het een even jongen of oneven meisje is. De gegevens impliceren dat K1 gezond is en Kn ziek, generaliseer tot invariant "Ki is gezond, Kj is ziek" en je hebt direct een Binairy Search om een paar te vinden in lgn stappen.

Hoe moet je het niet bewijzen? Er zijn mooie en duidelijke redeneringen die de kwestie terugbrengen tot een rij van n-2 kabouters, bijvoorbeeld deze:
als K2 of K{n-1} een jongen is, heb je daar een gelijk paar;
zijn ze beiden een meisje, dan is de deelrij [2..n-1] van even lengte en heeft gelijk begin en eind.
PROBLEEM met deze redenering is, dat als je deze gaat omzetten naar een constructief algoritme, dit lineaire tijd gaat kosten, bv in de situatie JMJMJMJM MJMJMJMJ

Er zijn ook veel redeneringen die "alleen beginnen" maar niet zijn "door te voeren".
Bijvoorbeeld:
"Kijk naar het midden, als er DIT aan de hand is zit er zeker een paar links ervan, als er DAT aan de hand is zeker een paar rechts.
Ga recursief zo door."
Wat is hier mis? Je algoritme treft bv op het eerste midden conditie DIT aan, gaat naar rechts en treft conditie DAT aan:
1.....m2.....m1.....n
Garantie dus, dat er een paar bestaat RECHTS van m1,

en dat er een paar bestaat LINKS van m2.
Echter, dit hoeft niet hetzelfde paar te zijn en er is
geen garantie op een paar tussen m2 en m1.

(b) Uit de redering volgt dat segment [i..j] beslist een
gelijk paar bevat als
1. j-i is EVEN en Ki en Kj hebben ongelijk type
2. OF j-i is ONEVEN en Ki en Kj hebben gelijk type.

Definieer het predicaat G(i,j) als
(j-i is EVEN en Ki /= Kj) OF (j-i is ONEVEN en Ki=Kj)

Het predicaat G heeft de volgende eigenschappen:
A. Er geldt G(1,n) (volgt uit gegeven).
B. Als G(i,j) en i <=h<=j, dan G(i,h) OF G(h,j).
(Mag je bewijzen, hoeft niet)
C. Uit G(i,i+1) volgt dat de naast elkaar staande
kabouters i en i+1 van gelijk type zijn.

En is daarom precies de INV die we nodig hebben:
INV: G(i,j)

Sommigen kwamen tot de invariant "j-i ONEVEN en Ki=Kj".
Probleem hiermee is dat je, na het berekenen van h tussen
i en j, totaal niet weet of m-i en j-m even/oneven zijn.

Binair zoeken gebruikt ALTIJD twee variabelen om het
zoekgebied te karakteriseren, en je invariant moet dus ook
een eigenschap van TWEE variabelen geven.

(c) Het eigenlijke ontwerpwerk is al in deel b gedaan.

```
i = 1; j=n;
while (j-i > 1)
  { h = (i+j)/2;
    if (G(i,h)) { j=h; }
    else        { i=h; }
  }
return als paar: i en i+1
```

Neem R = ceil(lg n-1); initieel is j-i <= 2^R en je kunt
inzien dat als j-i <= 2^r, dan h-i en j-h <= 2^{r-1},
wat impliceert dat verschil 1 wordt bereikt na
hoogstens R (en minstens R-1) ronden.
Per ronde bekijk je 1 kabouter (voor het eerst),
namelijk de nieuwe middenkabouter h.

Wat ik nog wel erg veel zag is dat men op de middenpositie
naar een paar gaat zoeken, dus met een test type(h)==type(h+1)
in de lus. Hiertegen heb ik op college gewaarschuwd: het
is zinloos want te duur; je gaat immers TWEE kabouters per
iteratie bekijken en in het slechtste geval doe je dat
nog steeds lgn keer.

Opgave 2, De arme monnik.

(a) Neem als oplossing: de indices van overnachtings-
plaatsen inclusief start en eind.
Zoekruimte: deelrijen van [0,1,...,n,n+1], die
beginnen met 0 en eindigen met n+1
Toelaatbaar: voor twee opeenvolgende indices i,i'
geldt s_i' <= s_i + M.
Doelfunctie (MIN): SOM c_i over i in de deelrij.

(b) De Optimale deelstructuur komt in beeld als je kijkt
naar de laatste overnachting, zeg m, voor het eindpunt.
Als S = (0,...,m,n+1) een optimale indexlijst is,
Dan is het beginstuk S' = (0,..,m) een goedkoopste
indexlijst van herbergen om in Hm te komen.
Bewijs: Om te beginnen: S' is inderdaad een toelaatbare
lijst indices waarmee je in Hm kunt komen. De
toelaatbaarheid van S' (afstand tussen opeenvolgende
herbergen is <= M) volgt direct uit toelaatbaarheid
van S.
Stel dat S" = (0,..,m) ook een lijst herbergen
is waarmee je in Hm kunt komen, en de kosten van S"
zijn lager dan van S'.
Vorm dan S"' = S",n+1.
Ten eerste is S"' een toelaatbare oplossing om in
n+1 te komen. Immers, de afstand tussen opeenvolgende
herbergen binnen S" is <= M, en s_{n+1}-sm <= M omdat
S toelaatbaar was.
Ten tweede is S"' een goedkopere manier om in H{n+1}
te komen. Immers, kosten S zijn S' plus c{n+1} en
kosten S"' zijn S" plus c{n+1}. Dus uit
kosten S" <= kosten S'
volgt
kosten S"' <= kosten S
Dit is in strijd met de aangenomen optimaliteit van S.
Conclusie: een OPTIMALE S met voorlaatste herberg m
bevat een OPTIMALE reis naar herberg m.

De meeste problematische inzendingen bleven te abstract,
zeiden bv dat je kon werken met Z = Z'+a, vervangen door
Z", etc., zonder dat werd duidelijk gemaakt hoe zo'n
deeloplossing eruit ziet.

(c) Deelprobleem: Reis naar herberg j, waar 0 <= j <= n+1.
De kosten om in het startpunt te komen zijn c_0 (dus 0).
Om naar herberg j te komen, moet je kijken naar de laatste
herberg i. Wegens de OSS is p[j] = p[i]+c_j:

```
p[0] = 0;
for (j=1; j <= n+1; j++)
 { // zoek goedkoopste voorherberg i
        g = +infty; i = j-1;
        while (i >= 0 && s[i] >= s[j]-M)
          { if (p[i] < g) g = p[i];
            i--;
          }
   p[j] = g + c[j]
       }
```

Opgave 3, Met de Bus.
(a) Zoekruimte: deelverzameling C van {1,...,n}
Toelaatbaar: SOM_{i in C} ci >= K.
Doelfunctie (MIN): #C.

Let erop, de zoekruimte is dus niet "de bussen", je moet hier echt wel aangeven dat je een deelverzameling zoekt en niet een permutatie of een bus.

(b) Het is altijd goed om de grootste bus, zeg m, te nemen, plus een selectie van de overige bussen tot capaciteit M-cm.
Ik formuleer en bewijs de GCP:
Voor elke deelverzameling C met totaalcapaciteit >=K
(waarbij de grootste bus m niet in C zit)
is er een deelverzameling C' met
- totaalcapaciteit >=K,
- grootste bus m erin,
- #C' <= #C.

Bewijs: zij C een verzameling bussen met voldoende capaciteit. Vorm C' door een willekeurige bus l uit C te verwijderen, en m erin te stoppen.
Dan is de capaciteit van C': cap(C) - cl + cm.
Omdat m de grootste bus is, is dit minstens cap(C)
dus C' is toelaatbaar.
De constructie impliceert direct dat m in C' en #C' = #C.

Algoritme:
sorteer de bussen op groot naar klein
cap = 0; i=0;
while (cap <K)
{ i++; cap += c[i];};
Stuur bussen 1 t/m i op pad

Er hoort ook nog een OSS-je bij:
Als C een optimale deelverzameling is die K personen vervoert, en C bevat bus b, dus is te schrijven als C'+b,
dan is C' een optimale verzameling van (alles min b) die K-c_b personen vervoert.
Deze OSS rechtvaardigt dat je, na het pakken van de grootste bus, verder gaat met de tweede grote enz.

Bonus: doe het in lineaire tijd.
Hoe vind je de gewenste verzameling bussen zonder ze op capaciteit te sorteren? Dat is namelijk met O(n lg n) de bottleneck in dit algoritme.
Punt is natuurlijk, dat er niet geeist is dat de verzameling gebruikte bussen in volgorde wordt opgeleverd en dat geeft de mogelijkheid, het probleem aan te pakken met LINEAIRE selectie.

Voor een instantie met n bussen en wenscapaciteit K:
1. Gebruik Selectie om de Mediaan MED te bepalen, en de bussen te splitsen in grotere en kleinere.
2. Sommeer de capaciteit van bussen grotergelijk MED.

3. ALS (die cap > K) { Bepaal selectie uit de grotere }
   ANDERS { Bepaal selectie uit de KLEINERE,
   met wenscapaciteit K - cap }

Stappen 1 en 2 kosten O(n), je probleem wordt met deze werkzaamheden gehalveerd.

Opgave 4, Geamortiseerde Boomkleuring.

Merk op dat de test in de while statement altijd EENMAAL VAKER wordt uitgevoerd dan de BODY. Je besluit immers een while altijd met een keer uitkomst FALSE.

(a) Voor een lange executie begin je in een compleet witte boom, onderin. Je kleurt d knopen en test dus de loop-conditie d+1 keer.
Als je in een zwarte knoop begint is de test meteen FALSE.

(b) Per definitie bedragen de geamortiseerde kosten:
c^ = c + PHInieuw - PHIoud.
Stel dat er t knopen worden gekleurd, dan is c = t+1,
en PHInieuw - PHIoud = -t, zodat c^ = 1.

(c) Voor een reeks van m operaties geldt
Werkelijke kosten = Geamortiseerde kosten
plus PHI(begin) - PHI(eind).
De geamortiseerde kosten zijn exact m omdat elke operatie exact 1 als c^ heeft.
Het aantal witte knopen in het begin is n, het aantal aan het eind is niet-negatief zodat
PHI(begin) - PHI(eind) <= n,
waaruit de grens van n+m volgt.