Algoritmer, datastrukturer och komplexitet

Mästarprov 2

Namn: _____

1. **Hållbart kommunikationsnät**

   Som indata till det generella handelsresandeproblemet, TSP, får vi en fullständig graf G med positiva heltalskantvikter och ett positivt heltal, mål M. Frågan lyder: Finns det en tur runt hörnen som passerar varje hörn i G exakt en gång och som har en sammanlagd kantviktssumma på högst M?

   Som indata till Hållbart kommunikationsnät får vi en n x n matris K, en array A som anger antalet kablar som ska gå från varje nod, samt ett mål M - alltså det kabelantal som inte får överstigas. Betrakta följande reduktion:

Algorithm 1 TSP(G, M)

```
for all hörn u, v i G do
  K[u][v] ← kantvikt(u, v) {Vi fyller en matris med kantvikter}
end for
for all hörn h i G do
  A[h] ← 2 {Vi får inte passera ett hörn mer än en gång}
end for
return HållbartKommunikationsnät(K, A, M)
```

   Vi representerar alltså grafen G som en symmetrisk matris, K, där $K[u][v]$ representerar kantvikterna, dvs kostnaden för dra en kabel mellan hörnen u och v. Vi behöver bara 2 kanter per hörn, eftersom vi ska passera varje hörn en gång. Vi behöver alltså en kant för att komma till hörnet, och en för att besöka ett annat hörn. $A[h]$ anger hörn h:s valens, dvs antalet kablar som ska gå till hörn h. Målet M är detsamma som för handelsresandeproblemet TSP.

   Om det inte finns en hamiltoncykel, dvs att varje stad (hörn) besök en gång, vars totala längd är högst M, så är det inte heller möjligt att bygga ett hållbart kommunikationsnät, vilket är trivialt att inse.

   För att visa att problemet ligger i NP behöver vi visa att en lösning till problemets går att verifiera i polynomiell tid. För att verifiera en lösning till Hållbart kommunikationsnät så behöver vi bekräfta att det i matrisen K ligger rätt antal kablar mellan alla noder, såsom specificerat av arrayen A, summera antalet kablar och kolla att antalet inte är större än målet M. Detta tar linjär tid i grafens storlek. Verifieringen tar alltså polynomiell tid och problemet ligger således i NP.

   Problemet är NP-svårt, ty vi har reducerat det NP-fullständiga handelsresandeproblemet. Eftersom det ligger i NP och är NP-svårt är det också NP-fullständigt.

2. **Energisnålt garage för tåg**

   Beslutsproblemet: Kan vi få plats med tågen i ett garage med sidan K?

   Som indata till Lådpackningsproblemet får vi n prylar med rationella vikter $w_1, ..., w_n$ mellan 0 och 1, och frågan lyder: Vilket är det minsta antal lådor som behövs för att förvara alla n prylar utan att någon låda innehåller mer än 1 kg?

   Som indata till Energisnålt garage för tåg har vi en array A med n stycken positiva heltal $t_1, ..., t_n$ där t_i anger längden på tågsätt nr i, samt ett mål K. Frågan lyder: Går det att avgöra ifall det finns något sätt att parkera tågsätten i ett garage med sidan K?

Algorithm 2 Lådpackning($w_1, .., w_n, K$)

```
faktor ← K
for i ← 1 to n do
    faktor ← faktor∗nämnare(w_i) {Vi vill bara ha heltal som indata}
end for
for i ← 1 to n do
    A[i] = w_i ∗ faktor {Skapa array med tåg från vikterna}
end for
t ← tåg med längd faktor
for i ← K + 1 to faktor do {Fyll ut överflödiga spår}
    A[i] ← t
end for
return EnergisnåltGarage(A, faktor)
```

Vi tar indata från lådpackningsproblemet, och använder prylarna som tåg och lådorna som spår. Vi förlänger varje tåg genom att multiplicera med produkten av alla nämnare eftersom vi behöver heltal som indata till Energisnålt Garage. Eftersom garaget måste kvadratiskt multiplicerar vi också med antalet lådor, K. Vi kallar produkten av alla nämnare * K för $faktor$.

Problemet nu är att tågen är för långa och får alltså kanske inte plats på spåren. Vi ökar därför sidan på garaget till $faktor$. Nu har vi alltså för många spår att ställa tågen på, så vi lägger till $faktor - (K + 1)$ stycken tåg av längd $faktor$, dvs vi fyller igen överflödiga spår. Detta lämnar oss med K stycken spår av längd $faktor$ där tågen ska få plats. Istället för K stycken lådor som rymmer 1 kg med vikterna $w_1, ..., w_n$ har vi alltså K stycken spår som rymmer $faktor$ långa tåg med tågen $t_1 * faktor, ..., t_n * faktor$, och på så sätt har vi behållit proportionerna mellan alla värden.

Problemet ligger i NP eftersom vi lätt verifierar lösningen i polynomiell tid. Problemet är NP-svårt eftersom vi har reducerat ett NP-fullständigt problem. Alltså är problemet NP-fullständigt.

3. **Konstruktion av parkering i energisnålt tåggarage**

   För att ta reda på den minsta garagesidan K, så kan vi börja med att ansätta $K = 1$, och sedan anropa beslutsproblemet och öka K med 1 varje gång vi får en nej-instans. När vi således får en ja-instans vet vi att vi kan bygga ett garage med sidan K så att alla tåg $t_1, ..., t_n$ får plats.

   Problemet är sedan att beskriva vilka tågsätt som ska parkera på samma spår, och detta kan vi göra genom att ”sätta ihop” två tåg för att på så vis se om de kan stå på samma spår. Om vi fortfarande får en ja-instans med samma mål K så vet vi att de kan stå parkerade på samma spår.

Algorithm 3 OptimalParkering(T)

```
M ← 1
while Beslut(T, M) != true do {Kan vi få plats med alla tåg med garagesidan M?}
  M ← M + 1
end while
s ← 1 {Spår s}
for all t in T do {För alla kvarvarande tåg i T}
  Låt L vara en tom lista
  Lägg till t till L
  i ← 2
  while i < T do
    X = t + t_i {Sätt ihop två tåg}
    T = T + {X} − {t} − {t_i} {Ta bort tågen som vi slagit ihop från T och lägg till X}
    if Beslut(T, M) then {Är detta tågsätt en möjlighet i en optimal lösning?}
      Lägg till t_i till L
      t ← X {Vi ökar tågstorleken till X}
    else
      T = T − {X} + {t} + {t_i} {De kan inte stå på samma spår}
    end if
    i ← i + 1
  end while
  P[s] ← L {Lägg till alla tåg som finns i L till spår s}
  s ← s + 1 {Det får inte plats några andra spår på s}
end for
```

Vad som händer är att vi slår ihop tåg, och undersöker om det finns nåt sätt att parkera med de ihopparade tågen. Om det finns betyder det att det finns en optimal lösning där de ihopslagna tågen befinner sig på samma spår. Om det inte finns betyder det att tågen inte kan vara på samma spår i en optimal lösning.

Vi vet att algoritmen fungerar eftersom längden av två tåg parkerade bredvid varandra är lika lång som två ihopslagna tåg. För varje tåg t behöver vi gå igenom alla kvarvarande tåg och undersöka om de kan stå på samma spår, så vi får tidskomplexiteten $O(n^2 * B(n))$