Mästarprov 2 - DD1352 ADK

Peter Boström
pbos@kth.se

2010-05-10

Contents

1 Tredelning **1**
1.1 Utförande 1
1.2 Pseudokod 1

2 Hissåkning i graf **2**
2.1 Utförande 2
2.2 Reduktion 2
2.3 Illustrativa figurer för hand 3

3 Konstruktion av hissgående **4**
3.1 Utförande optimeringsproblemet 4
3.2 Utförande konstruktionsproblemet 4

1 Tredelning

Visa att Tredelning är NP-fullständigt genom att reducera från det NP-fullständiga problemet mängdpartitionering.

För att visa NP-fullständighet måste två saker bevisas:

1. Att problemet tillhör NP.
2. Att problemet går att reducera till från något NP-fullständigt problem i polynomisk tid.

1.1 Utförande

Ifall verifikation av tredelning går i polynomisk tid tillhör tredelning NP. Det görs genom att räkna att summan av varje partition är lika och sedan kolla att varje tal förekommer rätt antal gånger. Trivial räkning då varje nytt tal läggs till i en lista med tillhörande räknare ger en komplexitet på $O(n^2) \in PTIME$. Eftersom en lösning går att verifiera inom polynomisk tid ligger problemet i NP.

Reduktionen görs från mängdpartitioneringsproblemet där en mängd positiva tal ska delas upp i två partitioner med samma summa. Det vill säga indata för tvådelningsproblemet ska omformuleras så att det går att lösa med hjälp av tredelningsalgoritmen.

Om tvådelningsproblemet går att lösa kan indatamängden delas upp i två partitioner med samma summa. Båda dessa kommer att innehålla halva av den totala summan. För att översätta problemet till tredelningsproblemet så introducerar vi ett nytt element till indatamängden som är lika stort som den halva summan av indatamängden. Denna läggs till mängden för körning av tredelningsproblemet.

Tredelningsproblemet tvingas att lägga det nya elementet till en egen partition eftersom problemet, om lösbart, kommer att ge partitioner av den storleken. Detta betyder att det som tredelningsproblemet har kvar att lösa är ifall alla element utom det tillagda går att dela upp i två mängder med samma summa. Kvarstående är alltså exakt originalproblemet. Går originalproblemet att dela upp i två delar, så kommer tredelningen även att fungera. Samma gäller åt andra hållet. Ifall probleminstansen var lösbar, och vi frånser från partitionen med det tillagda elementet så återstår enbart två partitioner med samma summa bestående av indatamängden till originalproblemet.

Eftersom mängdpartitioneringsproblemet är NP-fullständigt har vi även bevisat tvåan, att problemet är reducerbart från något NP-fullständigt problem. Reduktionen går i polynomisk (linjär t.om.) tid. Därför är även problemet tredelning är NP-fullständigt.

Poängen är att visa att tredelning kan lösa tvådelning, med hjälp av en s.k. karp-reduktion, en speciell reduktion som går i polynomisk tid. Förutsatt att $P \neq NP$ så går problemet mängdpartitionering som bekant inte att lösa inom polynomisk tid. Vad det visar är att tredelning omöjligt kan gå att lösa i polynomisk tid heller, då man annars hade kunnat lösa mängdpartitionering att både översätta det till tredelning och lösa det med tredelningsalgoritmen, allt inom polynomisk tid. Vilket blir motsägelsefullt, eftersom problemet mängdpartitionering omöjligt går att lösa inom polynomisk tid.

1.2 Pseudokod

```
1  function Mängdpartitionering(N) =
2      s = sum(N)
3      if s % 2 == 1:  # om summan är udda går talen inte att dela upp i
           två lika stora partitioner, då alla tal är heltal.
4          P = {1} # icke lösbar probleminstans (nej-instans)
5      else:
6          P = N + {s / 2} # om problemet är tvådelningsbart så måste det
               vara tredelningsbart med ett nytt element av halva storleken
7
8      return Tredelning(P)
```

2 Hissåkning i graf

Visa att problemet hissåkning i graf är NP-fullständigt.

2.1 Utförande

Först ska NP-tillhörighet visas, detta görs genom att visa att verifiering av lösning i polynomisk tid. För att verifiera att en ja-instans är korrekt måste vi få reda på den totala kostnaden för lösningen. Detta görs lämpligen med en bredden först-sökning som simultant utgår från alla noder som ska besökas, dvs. att alla dessa noder placeras som "startnoder" i bfs-kön och har "avstånd noll". När vi når ett ickebesökt mål som en (eller flera) personer skulle till så läggs helt enkelt n*T(d) till "totalkostnaden", där n = antalet personer som skulle till den noden, och d är avståndet till den. Detta görs tills alla noder som personer ville åka till är besökta. I värsta fall är exakt hela grafen besökt. Detta ger $O(|V| + |E|)$, eftersom det räcker med en BFS $=> O(n^2)$ vilket är polynomiskt. Sedan måste det även kollas om antalet noder i ja-instansen överrensstämmer med antalet tillåtna, samt att totalsumman underskrider gränsen. Detta är trivialt. Alltså ligger problemet i NP.

Den reduktion som redovisas utgår från hörntäckningsproblemet. Hörntäckningsproblemet är NP-fullständigt även för anslutna grafer, dvs. grafer som består av en komponent. Detta beror på att hörntäckningen i en komponent inte påverkar hörnteckningen i någon annan komponent, och problemen kan lösas enskilt och sedan kombineras. Reduktionen kommer alltså utgå från en NP-fullständig variant av hörntäckningsproblemet som endast tar anslutna grafer för att förenkla reduktionen och bli av med hantering av specialfall.

2.2 Reduktion

Först skapas en ny nod och en kant mellan denna och godtycklig nod i originalgrafen. Detta är v_0 där hissen startar ifrån och sedan inte får stanna. Varje kant i originalgrafen byts sedan ut mot två nya kanter som går mellan noderna. Varje ny kant får också en ny nod på sig, □, som ligger ett steg från originalkantens hörn. Till båda av dessa nya noder, □, skapas en ny person med mål att besöka varsin av dessa noder. Varje originalkant tilldelas också en "kvot" av arbete: 2*T(1) som är tillåtet används. Detta arbete är nämligen det arbete som krävs ifall hissen stannar på någon av de noder som angränsar till originalkanten, vilket innebär i hörntäckningsproblemet att kanten är täckt av den nod som hissen stannade på.

För att summera så kommer varje kant att tillföra två nya personer och ett tillåtet maxarbete på $2 * T(1)$. Antalet stopp som hissen får göra är samma som antalet hörn som får tillsättas i hörntäckningsproblemet. Ifall det finns en hörntäckning i originalgrafen med k antal hörn så kommer totalkostnaden K att bli maximalt $2|E| * T(1)$. Detta eftersom det endast är att stanna i varje hörn där hörntäckningen hade placerat en nod. Eftersom en ny nod introduceras som startnod för problemet, med tillhörande kant, så blir det även tillåtet att stanna i vilken nod som helst i originalgrafen. Viktigt att visa är att det inte uppkommer några ogiltiga ja-instanser vid reduktionen.

Viktigt att notera är att $2 * T(1) = 2 > T(2) = 2 * log(3)$. Istället för varje originalkant införs två nya noder som två nya personer har som mål. Ifall hissen stannar på någon av de noder som omger kanten behöver båda personer ta ett steg var, vilket ger $2 * T(1)$ i totalt arbete. Antag att hissen istället stannar på en av de noder som personerna ska av på. Då måste en person gå två steg, medan den andra personen inte behöver utföra något arbete. Totalt blir detta $T(2)$ arbete, vilket är sämre. För att tjäna mot arbetskvoten måste alltså hissen göra två stopp i "området". "Området" är de två noderna kring en originalkant, de två nyinförda noderna och de fyra kanterna mellan dem.

Om hissen inte stannar i ett "område" kommer det innebära att de två personer som skulle dit måste ta minst två steg. Arbetet blir alltså *minst* $2 * T(2)$ för dessa två personer. Det går alltså inte att "tjäna arbete gentemot kvoten" genom att en kant inte blir täckt av en nod. Enda sättet att tjäna in arbete mot kvoten är att använda ytterligare stopp när alla områden redan stannats i. För att kunna göra hissåkningen billigare än $2|E| * T(1)$ måste hissen alltså tillåtas stanna i fler noder än vad som krävs för den minimala hörntäckningen. Alltså kommer hissåkningen endast att vara genomförbar när en hörntäckning i originalgrafen är det.

En ja-instans av ena problemet medför alltså en ja-instans av det andra problemet. Reduktionen är alltså giltig åt båda hållen. Att ersätta varje kant med två nya noder, fyra kanter och två nya personer

kan gå i linjär tid. Den är definitivt polynomisk. Eftersom Karp-reduktionen går i polynomisk tid, är korrekt, och hörntäckning är NP-fullständigt, är även Hissåkningsproblemet NP-fullständigt.

2.3 Illustrativa figurer för hand

Här ska jag ha lite tid att rita ersättningen av kanten med nya kanter + noder, hoppas jag hinner rita dessa innan mästarprovet ska lämnas in. Annars förlåt. :)

3 Konstruktion av hissgående

Att göra en konstruktionsreduktion av hissgåendeproblemet. Reduktionen görs i två steg: Konstruera en polynomisk turingreduktion av optimeringsproblemet till beslutsproblemet till reduktionsproblemet med logaritmiskt anrop till beslutsproblemet och sedan från konstruktionsproblemet till beslutsproblemet.

Anropet till beslutsproblemet ser ut som följande: B(G=<V,E>, P, k, K) där G = grafen med tillhörande v_0, P = personernas stoppmål, k = antal tillåtna stopp, och K = målvärdet = maximalt tillåtet arbete.

3.1 Utförande optimeringsproblemet

Optimeringsproblemet löses genom att binärsöka B(G=<V,E>, P, k, K) för varierande K för att hitta minimala K:et för vilket hissåkningen är genomförbar. Detta ger ett logaritmiskt antal anrop (binärsökning). En trivial övre gräns kan hittas genom att säga att varje person placeras helt fel och måste passera alla kanter för att nå målet.

Detta ger ett maxarbete på $|P| * T(|E|)$ då varje person måste gå $|E|$ steg. Vi avrundar arbetet uppåt för att få ett heltalsvärde som sedan kan binärsökas med. Vilket ger totalt $log(|P| * T(|E|))$ anrop av beslutsproblemet enligt specifikation, då $T(x)$ inte växer exponentiellt. Notera att $|P| = n, |E| \leq m^2$,

Vi kan kalla optimeringsproblemet OPT(B). Det har tidskomplexitet $O(log(n * T(m^2)) = O(log(n * m^2 * log(m^2 + 1))) = O(log(n * m^2 * log(m^2)))$ förutsatt att anropen till beslutsproblemen inte räknas med. Reduktionen har alltså denna komplexitet, för helheten gäller annorlunda.

3.2 Utförande konstruktionsproblemet

I konstruktionsproblemet använder vi oss av optimeringsproblemet först för att få fram optimala originalgränsen. Detta räknas ut en gång via optimeringsproblemet som beskrivits tidigare.

$$KOSTNAD = OPT(B)$$

Om $KOSTNAD = 0$ vet vi att hissen kan stanna på allas målvåningar. Vi returnerar helt enkelt alla P:s destinationer som lösning utan dubletter. I senare steg vet vi alltså att det "kostar" att behöva flytta en hiss. Ifall det inte hade kostat så hade hissen kunnat flyttas till ett annat ställe som hade givit en billigare åktur. Vi får ingen billigare åktur än $OPT(B)$, då den är optimal.

Från varje nod i originalgrafen testar vi att lägga till k+1 nya kanter som ansluts till noden och varsin nyskapad nod. Vi lägger även till en person som ska gå till denna varje nyskapad nod och tillåter ett extra arbete på $(k+1) * T(1)$. Ifall beslutsproblemet fortfarande är lösbart med KOSTNAD+(k+1)*T(1) arbete betyder detta att ett stopp i en optimal lösning kan hamna i den nod från originalgrafen vi kollar på. Det kan inte vara lösbart med mindre än KOSTNAD+(k+1)*T(1) eftersom inget extra stopp kan göras i den nyskapade noden. (k+1 står eftersom jag kom på ett motfall där ifall man bara lägger till en ny nod så kan det leda till en fixering med ickeminimalt arbete. Denna hände om det finns en plats där det står en person och hissen i detta fall ibland skulle kunna stanna precis på nyskapade noden och inte minska förbrukat arbete. Detta blir omöjligt med k+1, då det inte finns k+1 noder på personers slutdestinationer, ty det hade givit KOSTNAD=0. Polynomiskt, alltså ok. Möjligtvis inte optimalt. Detta tvingar en fixering av originalnoden och inte dess nyskapade grannar.)

I det fallet, lägg till originalnoden i en lösningsmängd och låt de nyskapade noderna vara kvar. Låt även den nytillagda kostnadsökningen på (k+1)*T(1) ligga kvar (dvs. KOSTNAD += (k+1)*T(1)). Vad detta innebär är att noden fixeras till en lösning. Alla efterföljande anrop kommer att kräva att ett stopp görs på den nod som fixerades, då kostnaden annars hade ökat för den tillagda noden som nu ligger kvar.

Annars tag bort de nyskapade noderna och kanterna från grafen, och uppdatera inte kostnad, utan låt den stå kvar. (Detta sker som om denna iteration av loopen aldrig hade hänt.)

När alla noder slutligen har gåtts igenom så kommer alla stopp som gick att göra (i den ordningen) att vara fixerade, detta innebär att k st. stopp kommer att vara fixerade och att deras stoppunkter kommer att ligga i "lösningsmängden". Returnera lösningsmängden och vi är färdiga.

Tidskomplexiteten för att plocka bort alla dublettnoder ur P är $O(n)$ med t.ex. räknesortering, ev. $O(n + m)$ för att skapa en m st. stor array och gå igenom den. (Det påverkar dock inte slutgiltiga tidskomplexiteten.) Tidskomplexiteten för att gå igenom alla m st. noder är linjär, $O(k * m)$, då $O(k)$ arbete utförs varje gång (liksom tidigare frånser man från beslutsproblemet). Optimeringsproblemet är

logaritmiskt och därför mindre än dessa steg av algoritmen och kan därför bortses. Totalt ger de steg som räknas en tidskomplexitet på $O(n + k * m)$. Ska komplexiteten enbart anges i n och m är $n \leq m$, vilket medför $O(n + m^2)$, men den är inte en lika tight jämförelse om inte $k = m$.