DD1352 - Mästarprov 2

Mathilda Strandberg von Schantz

November 2016

1 Ultrasmart maraton

För att visa att Ultrasmart Maraton-problemet (det är vad problemet att vinna ett specialpris på Ultrasmart Maraton kallas hädanefter) är NP-fullständigt är det två saker som behöver visas: 1. att problemet är i NP 2. att problemet är NP-svårt. Att ett problem ligger i NP innebär att en ja-instans kan verifieras på polynomisk tid. En lösning till Ultrasmart Maraton skulle vara en permutation av noder i en graf G. För att verifiera att den är korrekt kollas att permutationen innehåller alla noder i G precis en gång, att det finns en kant mellan alla angränsande noder i permutationen, samt att den sammanlagda kantvikten av dessa kanter är minst hälften så stor som den sammanlagda vikten av alla kanter i G:

```
UltrasmartMaraton-verifiering
skapa boolesk lista L på n element där n är antalet noder i G
for varje hörn u i permutationen do
    if L(u) = 0 then set L(u)=1
    else
        return false
    end
end
for varje element x i L do
    if x=0 then return false
end
pathSum = totSum = 0
for hörn u och v som angränsar varandra i permutationen do
    if kant e = (u,v) inte existerar i G:s kantmängd then  return false
    else
        pathSum += e.weight
    end
end
for kant e i G:s kantmängd do
    totSum += e.weight
end
if pathSum < totSum/2 then return false
return true
```

Denna kontroll är O($|e|\cdot|n|$) eftersom man behöver gå igenom alla angränsande noder i permutationen och se om en kant mellan dem finns i G:s kantmängd, vilket i värsta fall tar linjär tid i antal kanter i G gånger antal par av noder i permutationen. Alla andra operationer i verifieringen tar linjär tid i antal kanter i G eller linjär tid i antal noder i G. Alltså är kontrollen polynomisk och problemet ligger i NP.

Att ett problem är NP-svårt innebär att det är minst lika svårt som alla problem som ligger i NP. Eller, mer formellt, så innebär det att ett problem X är NP-svårt om det för varje problem Y som ligger i NP finns en polynomisk reduktion från Y till X. Man kan visa att problemet är NP-svårt genom att visa att det är minst lika svårt som ett NP-fullständigt problem, för i så fall har man bevisat att problemet är minst lika svårt som alla problem i NP eftersom det NP-fullständiga problemet ju är det. I detta fall är det NP-fullständiga problemet Hamiltonsk stig. Vi ska alltså reducera Hamiltonsk stig till Ultrasmart Maraton-problemet. Vi säger att om det finns en svart låda som kan lösa Ultrasmart Maraton problemet så kan vi använda den svarta lådan för att lösa Hamiltonsk stig-problemet, och därmed visa att Ultrasmart Maraton problemet är "kraftfull" nog för att lösa Hamiltonsk stig. Reduktionen kan skrivas som

Hamiltonsk stig
Data: Oriktad graf G. G består av en sammanhängande komponent, inga ensamma hörn och har högst en kant mellan två hörn.
Result: Ett 'ja' eller 'nej' på frågan om det finns en Hamiltonsk stig i grafen, det vill säga om det finns en stig i grafen G som besöker varje hörn precis en gång

```
G' ← G
minDegree = INFINITY
skapar tom hörnmängd V"
for kant e i G' do
    e.weight = 1
end
for hörn u i G' do
    // d(u) = gradtalet för hörn u
    if d(u) < minDegree then
        minDegree = d(u)
        töm V"
        V" ← u
    end
end
u = V".getNode
hörn v = copy(u)
skapa kant e = (u,v)
set e.weight = |n|^3
return Ultrasmart Maraton(G')
```

Vi ska visa att det finns en giltig Hamiltonsk stig i G' där summan av kantvikterna i den Hamiltonska stigen är minst lika stor som halva summan av alla kantvikter i G' om och endast om det finns en Hamiltonsk stig i G. Vi säger att n = antal noder i G. Vi kallar den kopierade noden i G' för COPY. Vi säger att det finns en lösning till Ultrasmart Maraton-problemet som består av en permutation av noder $u_1, u_2, .., COPY, ..., u_n$ i G'. Samma permutation av noder kommer vara en lösning i G förutom att COPY inte kommer vara med. Det blir samma i motsatt riktning. Nu säger vi att vi har en permutation av noder som utgör en lösning i G. Eftersom G' är G förutom att G' har kantvikter och en extra nod så kommer vi kunna gå samma väg i G' som vi gjort i G, förutom att vi är tvungna till att gå på kanten med kantvikt $|n|^3$ för att få specialpriset också, eftersom det är det enda sättet vi når upp till hälften av den totala kantsumman. Lösningen G = $u_1, u_2, ..., u_n$ kommer alltså bli samma lösning i G' fast lösningen i G' även kommer innehålla noden COPY någonstans i permutationen. I lösningen ovan kopieras noden med lägst gradtal (alternativt en av noderna med lägst gradtal) i grafen. Det behöver inte nödvändigtvis vara så, men det valdes för att undvika att behöva kopiera massor av kanter.

Reduktionen är polynomisk i antal kanter samt noder i G. Först går vi igenom alla kanter och sätter deras vikt till 1, något som sker i O($|e|$). Sedan så går vi igenom alla noder och tittar deras gradtal, vilket sker i O($|n|$). Den totala tidskomplexiteten för reduktionen blir alltså O($|e| + |n|$), vilket är polynomiskt.

2 Schemaproblemet

Vi ska visa att besvarandet av frågan om det går att boka in n lektioner i k klassrum så att de ej slutar senare än timelimit är NP-fullständigt, och vi ska visa det genom att reducera Graffärgning. Till att börja med ska vi visa att problemet ligger i NP (för annars kan det inte vara NP-fullständigt). För att göra det visar vi att en verifiering av en ja-instans av problemet kan göras på polynomisk tid. En lösning till Schemaproblemet skulle vara en bokning av lektioner i klassrum. Det skulle kunna representeras som k stycken scheman där det finns en start- och sluttid angivna för varje lektion bokad i klassrummet, samt en identifierare för samtliga lektioner som är bokade. För varje schema skulle man behöva titta på varje lektion bokad i klassrummet och titta på lektionens eventuella villkor för att se att de är satisfierade. Vi säger att vi tittar på en lektion bokad i ett klassrum, och att lektionen har starttid s1 och sluttid s2. Man börjar med att titta om lektionen har ett villkor av typen "Måste bokas i klassrum x" och säkerställer att det faktiskt är schemat för klassrum x lektionen är bokad i. Man går sedan över till villkor av typen "Kan inte bokas samtidigt som lektion y" och tittar i andra scheman och säkerställer att lektion y inte är bokad samtidigt i något annat klassrum. För att vara mer exakt tittar man att y inte 1. har en starttid inom intervallet [s1, s2] eller 2. har en sluttid inom intervallet [s1, s2] eller 3. har en starttid $<$ s1 och sluttid $>$ s2. Slutligen tittar man på villkor av typen "Måste bokas senare än lektion z". Då kollar man på lektionerna som har sluttid $<$ s2 i alla klassrum och säkerställer att lektion z är bland dessa klassrum.

```
Schema-verifiering
for varje schema current ← 1..k do
    for varje lektion L bokad i aktuella schemat do
        s1 ← L.starttid
        s2 ← L.sluttid
        // kolla villkor typ 1
        if L.has(villkor 'klassrum x') then
            if current ≠ x then return false
        end
        // kolla villkor typ 2
        if L.has(villkor 'inte samtidigt som lektion y') then
            outerloop: for varje schema ← 1..k do
                for varje lektion L2 bokad i aktuella schemat do
                    if L2.starttid ≥ s2 then continue
                    if L2 = y then
                        if (L2.starttid ≤ s1 AND L2.sluttid ≥ s2) OR (s1
                          ≤ L2.starttid ≤ s2 OR s1 ≤ L2.sluttid ≤ s2) then
                          return false
                        else
                            break outerloop
                        end
                    end
                end
            end
        end
        // kolla villkor typ 3
        if L.has(villkor 'senare än lektion z') then
            outerloop2: for varje schema ← 1..k do
                for varje lektion L2 bokad i aktuella schemat do
                    if L2 = z and (L2.starttid ≥ s1 OR s1 ≤ L2.sluttid ≤
                      s2) then return false
                    if L2 = z and L2.sluttid ≤ s1 then break outerloop2
                end
            end
        end
    end
end
return true
```

Nu ser den där verifieringen ganska saftig ut men den är i alla fall polynomisk och går i $O(n^2)$ tid. Först går man igenom schemat för alla klassrum och för varje schema går man igenom alla lektioner bokade i det klassrummet (vilket egentligen är att gå igenom alla lektioner = O(n)) och för varje lektion måste man i värsta fall titta att villkor av typ 2 eller 3 är satisfierade, vilket i sin tur

i värsta fall leder till att man behöver gå igenom alla andra lektioner = O(n^2). Alltså ligger problemet i NP. Det som kvarstår är att visa att problemet också är NP-svårt.

Precis som i förra uppgiften visas detta genom en Karp-reduktion från ett känt NP-fullständigt problem. För detta väljer jag Graffärgningsproblemet, som går ut på att se om en oriktad graf är k-färgbar, det vill säga om det går att färga den i k färger så att inga angränsande noder är likfärgade. Vi säger att noderna i grafen är lektioner och att kanterna mellan dem visar på en konflikt. Eftersom kanterna bör tolkas entydigt så kan vi bara hitta en slags konflikt i dem. Det är svårt att kunna tolka en oriktad kant som "konflikten mellan dessa två lektioner är att den ena måste ske efter den andra", för vilken är den ena och vilken är den andra? Istället bryr vi oss bara om ett av kraven, som är att lektionerna som är förbundna med en kant inte får ske samtidigt. Därtill säger vi att alla lektioner har längden 1. Vad gäller klassrum så bryr vi oss inte om dem nämnvärt, eftersom den komplexiteten är svår att utläsa från grafen. Vi struntar alltså i rumsdimensionen i uppgiften och tar oss istället an tidsdimensionen. Vi säger att det finns lika många klassrum som lektioner, vilket gör rumsdimensionen till ett icke-problem, och säger att k i graffärgningsproblemet = timelimit i schemaproblemet. En reduktion kan då se ut på detta sätt:

Graffärgning
Data: Oriktad graf G samt målnummer m. G består av en sammanhängande komponent, inga ensamma hörn och har högst en kant mellan två hörn.
Result: Ett 'ja' eller 'nej' på frågan om det finns en m-färgning i grafen, det vill säga om det finns ett sätt att färga hörnen med m färger så att inga intilliggande noder har samma färg

```
skapa lista L
for varje hörn u ∈ G do d[u] ← ∞
s ← slumpmässig nod från G
d[s] ← 0
Q ← {s}
while Q ≠ ∅ do
    u ← DEQUEUE(Q)
    lektion x ← u
    x.length = 1
    for varje granne v till u do
        x.addConstraint("Kan inte ske samtidigt som v")
        if d[v] = ∞ then
            d[v] ← d[u] + 1
            ENQUEUE(Q, v)
        end
    end
    L ← x
end
k ← G.numberOfNodes
timelimit ← m
return Schema(k, L, timelimit)
```

Nu hävdar vi att grafen har en k-färgning endast om det finns ett sätt att schemalägga lektionerna i klassrummen så att ingen slutar senare än timelimit. Om det finns en k-färgning av noderna finns det ett sätt att dela in de n noderna i k färggrupper, vilket innebär att det också finns ett sätt att dela in de n lektionerna i k tidsutrymmen, eftersom precis som att alla noder som inte är förbundna kan ha samma färg så kan alla lektioner som representerar de noderna ske samtidigt fast i olika klassrum. Och eftersom k = timelimit och varje lektion har längd 1 så innebär det att ingen av dem kommer sluta senare än k, eftersom precis som att alla noder som har färg 1 kommer representeras som alla lektioner som har (starttid, sluttid) = (0,1) så kommer de noder som har färg k representeras av lektionerna som har (starttid, sluttid) = (k-1, k) så om vi hade en lektion som hade sluttid > k så skulle den behöva representera en nod som är i färggrupp x > k. Och i motsatt riktning så kommer det bara finnas en giltig schemaläggning där ingen lektion slutar senare än timelimit om det finns en k-färgning i grafen. För om det finns en giltig schemaläggning innebär det

att det finns ett sätt att dela in lektionerna i timelimit tidsutrymmen, vilket i sin tur innebär att deras korresponderande noder kan delas in i k = timelimit färggrupper.

Denna reduktion har tidskomplexiteten O($|n| + |e|$), där $|n|$ = antal noder i grafen och $|e|$ = antal kanter i grafen. Detta då algoritmen baseras på bredd-först-sökning och vi bara gör konstanta saker med noderna, som att lägga till villkor, längd o.s.v. Alltså är reduktionen polynomisk, vilket är ett krav för en Karp-reduktion.