Principerna för ett NP-Fullständighetsbevis

Ett fullständighetsbevis för ett problem inkluderar två delar. Den första är att problemet ska tillhöra NP, och den andra är att problemet ska vara NP-Svårt.

NP-tillhörighet

För att bevisa NP-tillhörighet så ska ett vittne kunna bevisa att en given lösning samt probleminstans är korrekt på polynomisk tid.

NP-Svårt

Ett möjligt sätt att bevisa att ett okänt problem är NP-svårt är att reducera ett känt NP-svårt problem till det okända problemet med en Karpreduktion som kan göras på polynomisk tid.

1 Artistproblemet

Artistproblemet som Beslutsproblem

Artistproblemet kan uttryckas som ett beslutsproblem genom att införa en variabel k som argument tillsammans med listan av hotell samt listan av artister som inte tål varandra. Nu formulerar man om problemet **artistproblemet**(H, A, k) som *"är det möjligt att boka totalt k stycken eller fler artister på hotellen H när inget av paren av artister A inte får bor på samma hotell?"*.

NP-fullständighet

För att bevisa Artistproblemet är NP-fullständig kan man bevisa att det tillhör NP genom att bevisa att det går att verifiera i polynomisk tid, och att det är NP-svårt genom att reducera Oberoende mängd problemet till Artistproblemet med en karpreduktion.

Artistproblemet tillhör NP

En lösning för artistproblemet är en lista för varje hotell med artister som är bokade på respektive hotell, en lista kan vara tom. Givet en sådan lista kan man gå igenom varje lista och säkerställa att det inte finns två eller mera artister bokade samma hotell som inte tål varandra. Detta kan göras för varje hotell på polynomisk tid. Givetvis ska det också kollas att alla hotell inte överskrider sin kapacitet samt att alla hotell och skådespelare är från den originella problem instansen. Detta går också att göra på polynomisk tid vilket betyder att Artistproblemet tillhör NP.

Artistproblemet är NP-Svårt

För att bevisa att Artistproblemet är NP-Svårt ska *oberoende mängd problemet* reduceras till artistproblemet. Givet oberoende mängd problemet med två argument G och k där G är en graf och k är antalet hörn som förväntas vara oberoende. Detta kan göras genom att betrakta hörnen i G som artister i artistproblemet samt kanterna mellan två hörn som en relation mellan två artister som inte tål varandra. Om det finns ett hörn som inte har några ensamma hörn bör de tas bort innan reduktionen och detta antal ska subtraheras från k. I detta fall behövs endast ett hotell som är lika stort som antalet totala artister och och k argumentet för Artistproblemet är samma som i oberoende mängd problemet minus antal ensamma hörn som togs bort.

Algorithm 1 Oberoende Mängd till Artistproblemet

Input: $G = \langle V, E \rangle$, k

1: $n \leftarrow$ antalet hörn från V som inte har grannar
2: $Hotels \leftarrow [|V| - n]$ // skapa lista med ett hotell som är lika stort som antalet hörn som har minst en granne
3: **return** Artistproblemet($Hotels$, E, $k - n$)

Beskrivning

Tidskomplexitet

Detta kan göras på polynomisk tid eftersom den enda kostamma beräkningen är att hitta alla ensamma hörn. Detta kan göras genom att för varje hörn kolla om det finns någon kant i E som går till eller från hörnet. För att göra detta bör man gå igenom alla kanter en gång för varje hörn alltså är komplexiteten $\mathcal{O}(|V||E|)$ vilket är polynomiskt.

Korrekthet

För att göra ett korrekthetsbevis ska en ja-instans i Oberoende Mängd problemet bli en ja-instans i Artistproblemet efter reduktionen samt en ja-instans av Artistproblemet som kommer från en reducerade Oberoende Mängd instans också vara en ja-instans innan reduktionen. Lite löst kan man säga att om hörnen $v_1, v_2, \cdots v_n$ där $s \leq n$ är oberoende skulle också artisterna som representeras av hörnen inte ogilla varandra och de så skulle kunna bo på samma hotell, vilket betyder att $n \geq s$ artister kan bokas. På samma sätt om artisterna $a_1, a_2 \cdots a_n$ kan bokas $s \leq n$ då skulle det finnas n stycken artister som inte ogillar varanda, alltså de har inte kanter mellan sig också bildar då en oberoende mängd.

2 Grendosproblemet

Vad består en lösning av

Efter som problemet är ett beslutsproblem så är svaret man får *true* eller $false$. *true* returneras om och ändas om det är möjligt att koppla ström till alla notställ i grafen (utom källan) med användning av uppsättningen grendosor. Grendosorna for endast kopplas mellan hörn som har kanter mellan sig och koppling måste börja vid källan. Om det inte är möjligt så returneras $false$. Själva lösning till detta skulle i så fall vara en ritning på hur kopplingarna av grendosor skulle se ut. Detta kan representeras som en riktad graf där kanterna är sladdar och hörnen är grendosornas förgrening. Alla kanter är riktade i strömmens riktning (från grendosornas propp till förgrening) och varje hörn har ett nummer associerat med sig som representerar antalet grenar i förgreningen.

Grendosproblemet tillhör NP

Om man betraktar en lösning som beskrivet tidigare så kan detta göras genom att först kolla att varje hörn i grafen har ett förgreningsvärde som är lika med eller större än antalet kanter som går från hörnet plus 1. Alltså att det finns ett uttag för varje grendosa som skall vara kopplad till hörnet, samt att själva notstället i det hörnet kan försörjas med ström. Ett undantag gäller för källan som alltid har förgreningsvärde 1 och endast har en kant från sig. Grafen måste vara icke-cyklisk, den måste vara sammanhängande, alla hörn från probleminstansen måste också finnas i lösningen, och alla förgreningsvärden i lösningen måste gå att ta från förgreningsvärdena i probleminstansen utan återläggning. Grafens rot måste också vara i källan.

Allt detta kan verifieras på polynomisk tid med hjälp av Dijkstras algoritm och andra grundläggande algoritmer. Alltså tillhör Grendosproblemet NP.

Grendosproblemet är NP-Svårt

Ett möjligt sätt att bevisa att ett grendosproblemet är NP-svårt är att reducera ett känt NP-svårt problem till grendosproblemet med en Karpreduktion som kan göras på polynomisk tid. Jag har valt att använda det kända problemet *Hamiltonsk Cykel* (HS) från föreläsning 25. Jag valde detta problem för att det är ett grafproblem och jag gillar cyklar.

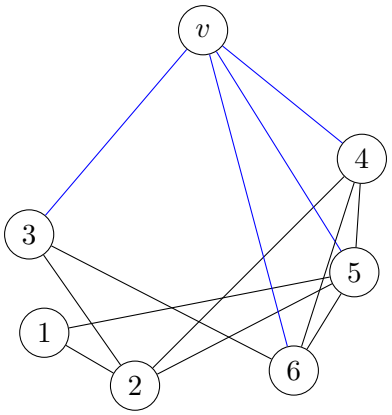
Algorithm 2 Hamiltonsk Cykel till Grendosproblemet

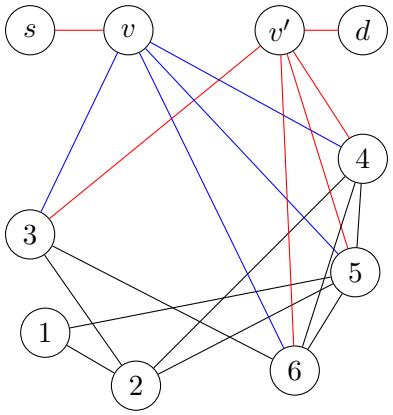
Input: $G = \langle V, E \rangle$

$v \leftarrow$ ett godtyckligt hörn från V
$N \leftarrow$ grannar till v
$v' \leftarrow$ nytt hörn
$source \leftarrow$ nytt hörn
$drain \leftarrow$ nytt hörn
$e' \leftarrow \emptyset$ // tom mängd av kanter
for $n \in N$ **do** // för varje hörn n som är granne till v
    $e' \leftarrow e' \cup (n, v')$ // skapa kant mellan v' och n
end for
$V' \leftarrow V \cup \{v', source, drain\}$ // lägg till hörnen v', $source$, och $drain$ i V'
$e' \leftarrow \{(source, v), (v', drain)\}$ // skapa kant från $source$ till v samt v' till $drain$
$E' \leftarrow E \cup e'$
$G' \leftarrow \langle V', E' \rangle$
$Grendosor \leftarrow [2, \overset{|V'|}{\cdots}, 2]$ // Lista med tvåor av längd $V'.length$
return Grendosproblemet(G', $source$, $Grendosor$)

Beskrivning

Vad algoritmen gör är att ta ett godtyckligt hörn v från grafen G och kopiera det på ett sätt så att ett nytt hörn v' har kanter till exakt samma hörn from v har kanter till. Det skapas också två nya hörn, *source* och *drain*, *source* har en kant som går till v och v' har en kant som går till *drain*. Dessa nya hörn samt nya kanter är därefter infogade i grafen G som sedan ges till *Grendosproblemet* med *source* som källa tillsammans med en lista av av grendosor som alla har 2 uttag, denna lista har samma längd som antalet hörn i den nya grafen G'. Grendosproblemet är ett beslutsproblem och förväntas returnera *true* eller $false$ enligt uppgiftsbeskrivningen.





Tidskomplexitet

Att ta ett godtyckligt hörn i grafen kan ske på konstant tid. Antalet grannar till detta hörn är begränsat till antalet kanter i grafen $|E|$, så *for* loopen i grafen har också samma begränsning. Skapandet av listan med grendosor sker på $|V|$ tid. Alltså är den totala komplexiteten $\mathcal{O}(|E| + |V|)$ vilket är polynomisk.

Korrekthet

För ett korrekthetsbevis av en karpreduktion är det möjligt att bevisa att en ja-instans i Hamiltonsk Cykel (HS) är en ja-instans i Grendosproblemet (GDP) samt att en ja instans i Grendosproblemet (GDP) är en ja-instans i Hamiltonsk Cykel (HS).

$$HS \Rightarrow GDP \wedge GDP \Rightarrow HS$$

$$HS \Leftrightarrow GDP$$

Hamiltonsk Cykel till Grendosproblemet

Lemma 2.1. *Om man ger en graf G' som innehåller en Hamiltonsk Stig som börjar i källan till Grendosproblemet och man har tillräckligt många grendosor med förgrening större än 1. Då är Grendosproblemet lösbart.*

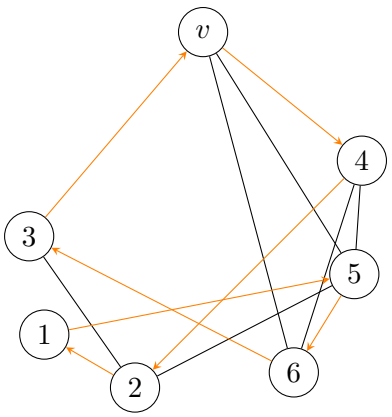
Proof. En Hamiltonsk Stig skulle besöka varje notställ exakt en gång. Om denna stig då börjar i källan kan man då koppla grendosorna enligt stigen för att nå varje notställ. Om alla grendosor som används har en förgrening större än 1 så kan ena förgreningen kopplas till notstället och den andra till det nästa notstället i den Hamiltonska stigen. På detta sätt uppfylls alla krav för en lösning av Grendosproblemet. □

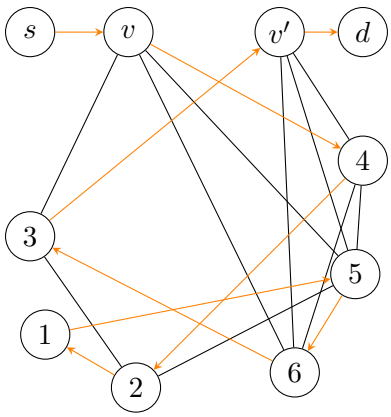
Lemma 2.2. ***Hamiltonsk Cykel till Grendosproblemet*** *kommer förse en Hamiltonsk Stig med tillräckligt många grendosor för att lösa Grendosproblemet.*

Proof. Varje notställ behöver ha sin egna grendosa, alltså behövs det minst lika många grendosor som det finns notställ. Vilket är antalet grendosor som ges på rad 14 i koden. Dessa grendosor har en förgrening som är större än 1. □

Lemma 2.3. *Om man ger en graf G som innehåller en Hamiltonsk Cykel till* ***Hamiltonsk Cykel till Grendosproblemet*** *algoritmen kommer en ny graf G' skapas som har en Hamiltonsk Stig.*

Proof. Om man startat i hörnet v som blir valt av algoritmen gå finns det en cykel som börjar i v, besöker alla hörn i grafen, samt slutar i v. Denna cykel är då trivialt en stig om man följer samma cykel men man slutar precis innan man kommer tillbaka till v. I exemplet nedan till vänster skulle en sådan stig vara $v, 4, 2, 1, 5, 6, 3$. Eftersom det nya hörnet v' har samma grannar som v så betyder det att v' kommer vara granne till det sista hörnet i stigen och kan då läggas till som det sista hörnet, att sedan lägga till hörnet s och d i början respektive slutet av stigen enligt grafen nedan till höger är trivialt. □





Lemma 2.4. *Om man ger en graf med en Hamiltonsk Cykel till* ***Hamiltonsk Cykel till Grendosproblemet*** *algoritmen kommer den ge en ja-instans i Grendosproblemet.*

Proof. Lemma 2.1 visar att om man har en graf med en Hamiltonsk stig och man ger denna graf till Grendosproblemet och sätter början av stigen som källan och ger problem tillräckligt många grendosor med förgrening större än 1 så är problemet lösbart. Lemma 2.2 bevisa att **Hamiltonsk Cykel till Grendosproblemet** får tillräckligt med grendosor av rätt förgrening för att lösa problemet. Lemma 2.3 bevisar att en graf med en Hamiltonsk Cykel som ges till **Hamiltonsk Cykel till Grendosproblemet** kommer skapa en ny graf med en Hamiltonsk stig. Allt detta tillsammans betyder att när algoritmen på rad 15 anropar Grendosproblemet(G', v, $Grendosor$) kommer *true* returneras om indata G har en Hamiltonsk cykel. □

Grendosproblemet till Hamiltonsk Cykel

Lemma 2.5. *En ja-instans i Grendosproblemet som skapas av* ***Hamiltonsk Cykel till Grendosproblemet*** *skulle innehålla en Hamiltonsk Stig som börjar i källan.*

Proof. Eftersom de reducerade instanserna som ges till Grendosproblemet i **Hamiltonsk Cykel till Grendosproblemet** endast har grendosor som har förgreningsgrad två, betyder det att vid varje notställ att enda förgreningen kommer gå till notställets lampa och den andra kommer vara kopplad till exakt ett annat notställ. På detta vis är inte några förgreningar möjliga och uppsättning måste vara en stig. En ja-instans måste besöka varje notställ samt börja i källan. Alltså är varje ja-instans en Hamiltonsk stig av grendosor som börjar i källan. □

Lemma 2.6. *Om det finns en Hamiltonsk stig i G' som skapas av* ***Hamiltonsk Cykel till Grendosproblemet*** *så finns det en Hamiltonsk cykel i den original grafen G som gavs som argument till algoritmen.*

Proof. En Hamiltonsk stig i G' måste börja i s eller d (eller tvärt om) eftersom bägge dessa hörn har udda gradtal. s och d är endast kopplade till v respektive v' vilket betyder att början och slutet av den Hamiltonska stig lyder $s, v, \cdots, v', d$. Hörnen s, d, v' fanns inte i original grafen G och kan därför tas bort, och istället för att ta bort kanten som går till v' kan denna knytas till v istället för att då forma om stigen till en cykel. Detta kommer alltid vara möjligt eftersom att v och v' delar samma grannar. □

Lemma 2.7. *Om G' är grafen från en ja instans i Grendosproblemet och G' är kommer från resultatet av en reduktion från G med* ***Hamiltonsk Cykel till Grendosproblemet****, algoritmen. Då innehåller G en Hamiltonsk Cykel.*

Proof. Lemma 2.5 säger att ja-instansen i grafen G' skulle innehålla en Hamiltonsk stig. Lemma 2.6 säger att om det finns en Hamiltonsk Stig i G' så finns det en Hamiltonsk cykel i G. Alltså är alla ja-instanser i Grendosproblemet som kommer från **Hamiltonsk Cykel till Grendosproblemet** också en ja-instans i Hamiltonsk Cykel problemet. □

3 Konstruktion av Artistproblemet

Denna lösning blev inte godkännd

~~Vi antar att vi har en algoritm **CanBookArtists(H, A, k)** som kan lösa Artistproblemet när det är ställt som ett beslutsproblem.~~ Vi vill använda denna beslutsalgoritm för att skapa en konstruktion av lösningen för Artistproblemet genom en Turing reduktion som anropar beslutsalgoritm högst kvadratiskt många gånger.

Turing Reduktion av ett Hotell

Betrakta delproblemet av en instans där det endast finns ett hotell med h stycken platser. Låt m vara största delmängden av artister som går att boka på samma hotell utan att det blir en konflikt. I denna problem uppsättning går det att som mest att boka $min(m, h)$ artister eftersom att det antingen inte finns tillräckligt med plats på hotellet eller att det inte finns tillräckligt många artister som kan bo tillsammans. För att hitta detta värde kan man börja med att anropa **CanBookArtists(h, A, k)** och börja med att $k = 1$ och öka värde av k tills algoritmen säger att det inte är möjligt att boka k stycken artister länge. Jag kommer använda denna princip för att göra konstruktionsalgoritmen för artistproblemet med exakt ett hotell.

Algorithm 3 Artistproblemet Ett Hotell

Input: h, A

1: $max \leftarrow 0$ // antal artister som går att boka
2: **while** CanBookArtists($[h], A, max + 1$) **do**
3: $max \leftarrow max + 1$
4: **end while**
5: // max är nu det maximala antalet artister som går att boka
6: $L \leftarrow$ lista av alla individuella artister från A
7: $RES \leftarrow \emptyset$
8: **for** $i \in 0$ to $L.lenght$ **do**
9: **Invariant** $\forall k < i$ gäller $a_k \in RES \Leftrightarrow a_k$ är nödvändig för en optimal lösning
10: $a_i \leftarrow L[i]$
11: $A \leftarrow A \setminus a_i$ // ta bort alla par av artister som innehåller a_i
12: **if not** CanBookArtists($[h], A, max$) **then**
13: $RES.append(a_i)$
14: $A \leftarrow A \cup a_i$ // lägg tillbaka alla par av artister som innehåller a_i
15: **end if**
16: **end for**
17: **return** RES

Beskrivning

Vad denna algoritm gör är att först räkna ut max antalet artister som går att boka och sparar det i variabeln max. Sedan provar man att ta bort en artist a_i i taget från A och kollar om det fortfarande går att boka max stycken artister. Om det inte längre är möjligt så betyder det att a_i var en nödvändig artist att boka och a_i läggs till i resultatet och alla par som a_i är med i läggs tillbaka i A. När alla artister har blivit provade returneras res som då är listan av artister som är nödvändiga för att boka max stycken artister.

Tidskomplexitet

Första delen av koden som räknar ut antalet bokbara artister behöver högst h slingor. Att skapa listan av artister kan göras genom att gå igenom paren av artister och lägga till bägge artister i en lista, sedan sortera listan med *merge sort* och iterera listan igen och ta bort alla duplikater på linjär tid, totalt tar detta $|A|log(|A|)$. Att iterera genom alla artister sker proportionellt mot antalet par i A, samma komplexitet gäller när man tar bort alla par av artister. Samma gäller att lägga tillbaka paren av artister. Sammanfattningsvis så är komplexiteten $\mathcal{O}(h + |A|log(|A|) + |A|) = \mathcal{O}(h + |A|log(|A|) = |A|log(|A|)$. Denna komplexitet är polynomisk.

Antalet **CanBookArtists(**$[h]$**,** A**,** max**)** anrop är proportionellt mot både h och A, eftersom den används i bägge loopar. Vi vet också att det inte går att boka mer än exakt alla artister så antalet anrop är begränsat till $|A| + min(|A|, h) = \mathcal{O}(|A|)$

Korrekthet

För att bevisa korrektheten kan en invariant användas. Den första loopen är tillräckligt trivial för att korrektheten inte behöver analyseras, men man kan säga att en optimal lösning inte kan boka mer än max antal artister på hotellet. Den andra loopen har invarianten I som lyder att alla behandlade artister (alltså att deras index är mindre än i) har blivit tilldelade hotellet h om artisten är nödvändig för att boka en optimal lösningen (en lösning som kan boka max stycken artister).

Eftersom i är lika med 0 innan loopen börjar så är det trivialt att I är sann.

På rad 11 så tas artisten a_i bort från problemet och sedan på rad 12 så körs **CanBookArtists(**$[h]$**,** A**,** max**)** med a_i borttagen från A. Om det fortfarande är möjligt att boka max stycken artister även om a_i är borttagen så är a_i inte en nödvändig artist och loopen kommer repeteras med $i = i + 1$ och I håller. Om det visar sig att det inte går att boka max stycken artister utan a_i så är a_i en nödvändig artist. I detta fall så tilldelas hotellet a_i och alla par som a_i tillhör läggs tillbaka i A.

När loopen har kört färdigt så har alla artister behandlas och de har antingen blivit tilldela hotellet h om de är nödvändiga för en optimal lösning eller så är de inte nödvändig och därav inte blivit tilldelad till hotellet. I bägge fallen så betyder det att RES består av en optimal lösning.

Turing Reduktion av flera Hotell

Artistproblemet kan sedan lösas för flera hotell genom flera anrop till Artistproblemet för ett hotell. Man gör detta genom att anropa **ArtistproblemetEttHotell(h, A)** för varje hotell i omvänd storleksordning samt att man tar bort alla artister som har fått en plats på ett hotell.

Lemma 3.1. *Givet en mängd hotell $H = h_1, h_2 \cdots h_n$ och en lista artister A. Om h_i är det största hotellet och det är möjligt att boka upp till k artist på det hotellet. Antalet totala artister som går att boka på alla hotell ändras inte om de k stycken artister bokas på hotell h_i*

Proof. Låt OPT vara en optimal algoritm och $|OPT|$ vara det maximala antalet artister i en optimal lösning. Vid den tidpunkten där inget av hotellen har fått någon artist, det är inte möjligt att tilldela mer än k stycken artister ett hotell åt gången eftersom det är det som går att tilldelas till det största hotellet. Alltså kan den OPT inte prestera bättre än de giriga algoritmen i det stegen. Efter detta steg återstår samma problem där A är samma som men för $H \setminus h_i$ (samma artist kan vara bokad på flera hotell), nu kan OPT fortfarande inte tilldela mer än $|OPT| - k$ så skillnaden mellan steget är högst k alltså är det alltid minst lika lönsamt som OPT att i ett Artistproblem att tilldela det största hotellet så många artster som möjligt.

□

Med hjälp av insikten från lemma 3.1 går det att konstruera en girig algoritm som löser konstruktionsproblemet med giriga anrop till **ArtistproblemetEttHotell(h, A)**.

Algorithm 4 Artistproblemet Flera Hotell

Input: H, A

1: $RES \leftarrow \emptyset$ // Tom lista av listor
2: $H \leftarrow mergeSort(H)$
3: $H \leftarrow reverse(H)$
4: **for** $i \in 0$ to $H.length$ **do**
5: **Invariant** $\forall k < i$ gäller $RES[H[k]]$ blivit tilldelat artister $\wedge$ tilldela artister är inte med i A
6: $h_i \leftarrow H[i]$
7: $res \leftarrow ArtistproblemetEttHotell(h_i, A)$ // hitta artisterna som kan bokas på h_i
8: $RES[h_i] \leftarrow res$ // lägg till artisterna i RES på den plats som motsvarar h_i
9: $A \leftarrow A \setminus res$ // Ta bort alla par av artister som blev innehåller en bokad artist
10: **end for**
11: **return** RES

Beskrivning

Algoritmen hittar den optimala tilldelningen för det största hotellet med hjälp av **ArtistproblemetEttHotell(h, A)** och sparar den i tabellen RES, de artister som har fått en tilldelning tas sedan bort från mängden av artistpar sedan upprepas problemet för det näst största hotellet o.s.v.

Tidskomplexitet

Sortera och reversa H sker på $|H|log(|H|)$ tid, att iterera genom alla hotell sker givetvis på $|H|$ tid, att ta bort artister från A kan ske på linjär tid för högst h_i stycken artister, och som etablerat tidigare så sker funktionen **ArtistproblemetEttHotell(h, A)** på $|A|log(|A|)$ tid. Det gäller också att om $|H| \geq 2|A|$ så är problemet trivialt lösbart genom att ge ett eget hotell till varje artist. Så totalt är komplexitet för algoritmen

$$\mathcal{O}(|H|log(|H|) + |H||A|log(|A|)) = \mathcal{O}(|A|^2 log(|A|))$$

CanBookArtists($[h], A, max$**)** anropas $|A|$ gånger i **ArtistproblemetEttHotell(h, A)** som i sig anropas $|H|$ gånger. Likt tidigare resonemang är $|H|$ begränsat till $2|A|$ så antalet anrop till **CanBookArtists(**$[h], A, max$**)** sker $\mathcal{O}(|A|^2)$ gånger.

Korrekthet

För att bevisa korrektheten av en girig algoritm ska man bevisa att vid varje steg att att algoritmen presterar minst lika bra som en optimal lösning. Detta utfördes is lemma 3.1. För att bevisa korrektheten av koden kan en invariant användas. Invariant lyder att alla hotell som har blivit behandlade har fått en optimal tilldelning av **ArtistproblemetEttHotell(h, A)** och att alla tilldelade artister har tagits bort från A.

Innan loopen börjar så är varianten trivialt sann eftersom inget hotell har blivit tilldelat artister, och inga artister har blivit tilldelat ett hotell.

Varje slinga börjar med att tilldela artister till det största obehandlade hotellet, dessa artister läggs till i resultatet och alla par som innehåller någon av artisterna tas också bort. Efter detta kommer det då fortfarande gälla att de behandlade hotellen har fått artister tilldela enligt rad 8, samt att A

inte innehåller bokade artister enligt rad 9. Efter slingan är färdig kan man då tolka det som att alla hotell har blivit tilldela sina artister så det inte finns några obehandlade hotell kvar.