Korrekthetsresonemang för reduktion av rollbesättningsproblemet

Rollbesättningsproblemet ligger i NP

I rollbesättningsproblemet försöker vi att tilldela skådespelarna en roll i grafen givet tre begränsningar. Det är samma som att fråga om grafen går att färga med X antal färger där färgerna representerar antal skådespelare i vår instans. Problemet kan reduceras till graffärgning

$$\textit{graffärgning} \leq_p \textit{Rollbesättning}$$

För att visa problemet ligger i NP kan vi omformulera optimeringsproblemet till ett valproblem där given en instans I och ett förslag S, kan vi konstruera en funktion Verifier(S,I) som på polynomisk tid verifiera att S är en lösning till instans I.

Valproblem: kan rollerna besättas med högst k skådespelare så att p1 och p2 är alltid med men inte i samma scener?

$$verifier(\mathrm{S},\mathrm{I}) = \begin{cases} true \rightarrow s \in I \\ false \rightarrow s \notin I \end{cases}$$

Verifier: given lösning S, och instans I, vi kallar Verifier(S,I) = 1. Verifier går igenom grafen se till att samma skådespelare inte ligger i närliggande noder, se till att p1 och p2 är inte i närligande noder, se til att noderna finns i det givna probleminstans. Detta förfarande kan göras i polynomisk tid -> Rollbesättning ligger i NP.

Polynomisk verifikation

Vi kan verifiera i polynomisk tid att en given lösning S till en instans I verkligen är korrekt, Verifier(S,I) -> 1. Därför ligger rollbesättning i NP

SAT representation av problemet

Ett annat sätt att representera problemet, som i många avseende är enklare eller mer intuitivt än graffärgning är faktiskt att använda sig av SAT.

Givet ett antal klausuler $c_1, c_2, ..., c_k$ och ett antal variabler $x_1, x_2, ..., x_n$. Vi kan upp bygga upp ett system $c_1 \wedge c_2 \wedge ... \wedge c_k = (x_1 \vee x_2 \vee .... \vee x_n) \wedge (x_1 \vee x_2 \vee .... \vee x_n) \wedge ... \wedge (x_1 \vee x_2 \vee .... \vee x_n)$. Där x_i representerar skådespelare i och närliggande klausuler representerar scener. Vi ska tilldela en 1 till en given variabel för varje klausuler sådan att $c_i \sim c_j -> x_1 \in c_i \leftrightarrow x_2 \notin c_i$. Det vill säga att p1 och p2 inte förekommer i samma scener. Därav följer att närliggande klausuler inte ska ha samma skådespelare tilldelade.

Reduktions korrekthet

Basfall

Den givna instansen är en trivial ja-instans. Om vi reducerar från graffärningen vet vi att vi kommer alltid att få en ja-instans om antal färger än större eller lika med antal noder. I det fallet returnerar vi den minsta castingen.

```
System.out.println("3");
System.out.println("2");
System.out.println("3");
System.out.println("1 1");
System.out.println("1 2");
System.out.println("1 3");
System.out.println("2 1 3");
System.out.println("2 2 3");
```

Generella fall

I det är fallet kan vi inte dra några slutsatser om problemets natur. Därför måste vi tar hänsyn till följande :

- p1 p2
- isolerade hörn
- en cykel reduceras till 3 beståndsdelar

1. P1 och p2 ska vara med men inte i samma scener
   För att kunna satisfierar det problemet lägger vi till basfallet i vår reduktion. Och sätter följande variabel
   ```
   int antal_roller = antal_horn + 3; // Scenes
   int antal_scenner = antal_kanter + 2 + antal_horn; // Actors
   int actors = antal_farg + 2;
   ```

2. Isolerade hörn
   För att ta an av det problemet itererar vi igenom hela grafen och se till att alla noder (Roller) är kopplade till scen 3
3. En cirkel av tre kanter reduceras alltid till 3 enskilda beståndsdelar
   Vi vet från problemet att en cirkel alltid kan reduceras till sina beståndsdelar. Därför tar vi varje närliggande noder i par från indata och adderar 3 eftersom vi har lagt till tre noder som inte fanns med förut.

Tidskomplexitet

Bästa fall

O(1)

Genomsnittligt fall

O(VM + E)

//(V(M+2) + E)

BILAGA

```java
package com.company;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

/**
 * Created by ben on 2017-11-15.
 */
public class reduce
{
   public static void main (String [] args) throws IOException
   {
      String NEW_LINE_SEPARATOR = "\n";
      //   File text = new File("data.txt");
     Scanner sc = new Scanner(System.in);
      int antal_horn = Integer.valueOf(sc.nextLine()); // v
     int antal_kanter = Integer.valueOf(sc.nextLine()); // e
     int antal_farg = Integer.valueOf(sc.nextLine()); // m

     // If the colours are more or equal than vertices, the answer is yes
     // Output minimal 'yes' casting graph
     if (antal_farg >= antal_horn) {
         generate();

      } else {
         // Roles
       int antal_roller = antal_horn + 3;
         // Scenes
       int antal_scenner = antal_kanter + 2 + antal_horn;
         // Actors
       int actors = antal_farg + 2;

         System.out.println(antal_roller);
         System.out.println(antal_scenner);
         System.out.println(actors);
         // Minimal roles
       System.out.println("1 1");
         System.out.println("1 2");
         System.out.println("1 3");


         for (int i = 4; i <= antal_roller; i++) {
            for (int j = 1; j <= actors; j++) {
               if (j == 1) {
                  System.out.print(actors + " " + j);
               } else {
                  System.out.print(" " + j);
               }
            }
            System.out.println(" ");
         }


         // Minimal scenes
       System.out.println("2 1 3" );
```

```
            System.out.println("2 2 3");

            // Ensure no isolated vertices (connect all vertices to scene 3)
          for (int i = 1; i <= antal_horn; i++) {
              System.out.println("2 3 " + (i + 3));
            }

            for (int i = 0; i < antal_scenner && sc.hasNext() ; i++) {
              String line = sc.nextLine();
              String [] values = splitter(line);
              System.out.println("2 " + (Integer.valueOf(values[0]) + 3) + " " + (Integer.valueOf(values[1]) + 3));
            }
        }
    }

    static void generate() {
        String NEW_LINE_SEPARATOR = "\n";
        System.out.println("3");
        System.out.println("2");
        System.out.println("3");
        System.out.println("1 1");
        System.out.println("1 2");
        System.out.println("1 3");
        System.out.println("2 1 3");
        System.out.println("2 2 3");

    }
    static String [] splitter(String lis)
    {
        String [] tra = lis.split(" ");
        return tra;
    }
}
```