Mästarprov 1

Pierre Le Fevre, pierrelf@kth.se

DD2350, HT21

Färgsortering med minimalt antal hällningar

Givet:

- 12 färger (A-L)
- 14 rör
- n (Antal färger i ett rör)
- $O(c^n)$

Algoritmidé

För att lösa uppgiften kan vi jämföra alla rör med alla andra, varje steg. Givet är att komplexiteten är $O(c^n)$, alltså är det en rimlig metod. För att se till att lösningen använder sig av bredden-först metoden används en kö. Vid varje iteration kommer alla rör att jämföras med alla andra, och för varje jämförelse kommer ett nytt "stadie" att sparas i kön, om det gick att hälla.

Om det går att hälla eller inte är, som enligt uppgiften, en enkel jämförelse. Röret man ska hälla till ska ha minst en ledig plats i toppen, och båda rör måste ha samma färg i toppen.

Stadiet innehåller vilka byten som behövts för att komma så långt, för att sedan kunna hämta ut de för att skriva ut. När alla jämförelser skett för det stadiet hämtas då första stadiet i kön, så startar processen igen. Den kortaste vägen kommer alltid vara den som kommer först i kön, så man får alltid den bästa lösningen direkt när problemet är löst. Att problemet är löst eller inte vet man genom att alla rör bara har en färg per rör.

För att hämta ut de när problemet är löst kan man dra ut listan av steg ur det stadiet som var aktivt när loopen avslutades. Sedan är det bara skriva ut de kronologiskt, alltså som de ligger i listan.

Ett förslag på datastrukturer

```
Step {
    int from
    int to
}

Pipe {
    char[] pipe
}

State {
    Pipe[] pipes
    Step[] steps
}
```

Pseudokod

```
algorithm färgsortering is
    input: Initial state of color pipes (State state)
    output: Steps to replicate

    Queue<Step> q

    while true
        if solved(state)
            for step in state.steps
                print("Häll från" + step.from  + "till" + step.to)
            return state.steps

        for p1 in state.pipes
            for p2 in state.pipes
                if (not p2.isFull()) and (p1.top() = p2.top())
                    swapTop(p1, p2)
                    if q.contains(state)
                        q.offer(State(state, Step(p1, p2)))

        //Load next state
        state = q.poll()
```

Tidskomplexitet

Enligt uppgiftens specifikation är komplexitet exponentiell, med konstanten *c* som bas, alltså $O(c^n)$. Frågan är alltså vad *c* är. I och med att det finns 14 rör, och vi jämför alla rören varje gång blir komplexiten $O(14^n)$. Skulle antalet rör förändras kommer *c* också att förändras.

Utskriften använder stegen som sparats i stadiets steg-lista. Antalet steg är lika med antalet hällningar som krävs för att röret ska vara sorterat. Det är bara att skriva ut en lista, alltså linjär tid. Antalet hällningar är rimligtvis nära kopplat till antalet färger per rör, alltså *n*.

Korrekthet

För att skapa en korrekthetsanalys måste de två viktigaste delarna av algoritmen analyseras, alltså indata och utdata (PRE respektive POST). Som indata är mycket redan bestämt, men antal färger i varje rör bestäms vid körning. Det kan därför vara viktigt att undersöka vad som händer för "gränsfall" av indata, ex. n=0. För utdata behöver vi kunna verifiera att lösningen faktiskt är löst. Det finns flera krav i uppgiften som bestämmer om ett stadie rör är löst eller inte.

Det är alltså viktigt att ingångsvillkoren, ex. PRE, är uppfyllda vid start, och att utgångsvillkoret POST uppfylls när programmet är klart. Det går även att sätta in mindre checkar under tiden programmet körs (ASSERT). För färgsorteringsalgoritmen är det egentligen inte så många olika operationer som görs, så det skulle vara svårt att implementera en sådan.