Delvis kortfattade lösningsförslag för tentamen i Datastrukturer (DAT036) från 2013-08-23

Nils Anders Danielsson

1. Notera att `t` kommer att innehålla som mest 10 olika värden. Tidskomplexiteten blir $\Theta(n)$.

2. Följande implementation är linjär eftersom konstant arbete utförs för varje element (plus konstant övrigt arbete):

```
// Skapar ett träd med samma struktur och innehåll som t.
// Arrayen t ses som ett komplett binärt träd, med noderna
// i nivåordning, och en eventuell rot på position 0.
public Tree(A[] t) {
    if (t == null) {
        return;
    } else {
        root = fromArray(t, 0);
    }
}

private Node fromArray(A[] t, int pos) {
    if (pos >= t.length) {
        return null;
    } else {
        return new Node(t[pos],
                        fromArray(t, pos * 2 + 1),
                        fromArray(t, pos * 2 + 2));
    }
}
```

3. Se uppgift 3 från duggan som gavs 2012-11-21. (Lösningen behöver modifieras något eftersom vi bara ska avgöra om listorna innehåller samma element, och inte om de är permutationer av varandra.)

4. Om `n` $= 2$, `k` $= 1$ och `xs` $= \{1, 2\}$ så blir resultatet 1, inte 2. Algoritmen kan fås att fungera genom att byta ut maxheapen mot en minheap, och `delete-max` mot `delete-min`.

5. Anta att grafen är $G = (V, E)$. (Jag har valt att låta kantmängden E bestå av tripplar av formen (u, v, w), där $u, v \in V$ och $w \in \mathbb{N}$. En sådan trippel står för en kant från u till v med vikten w.) Vi kan konstruera en ny graf $G' = (V', E')$ där

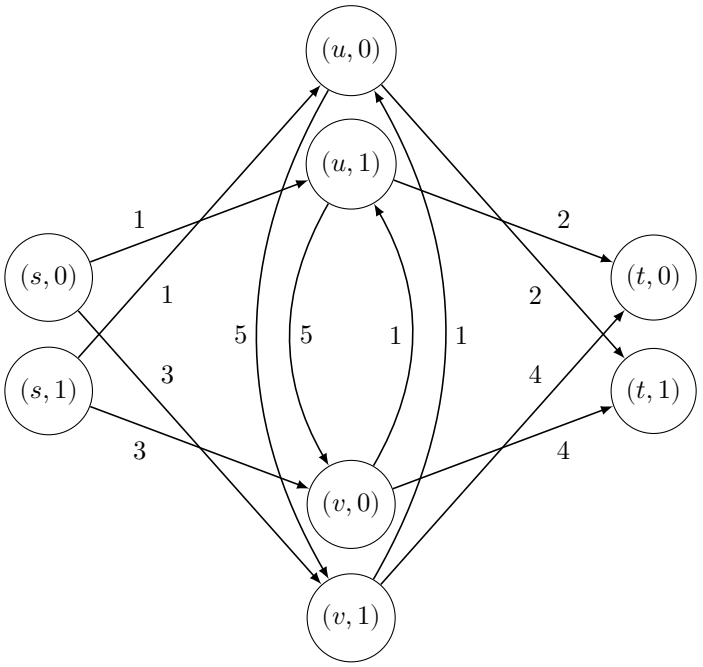
$$V' = \{ (v, 0) \mid v \in V \} \cup \{ (v, 1) \mid v \in V \}$$

och

$$\begin{aligned} E' = {} & \{ ((u, 0), (v, 1), w) \mid (u, v, w) \in E \} \cup \\ & \{ ((u, 1), (v, 0), w) \mid (u, v, w) \in E \}. \end{aligned}$$

Den nya grafen G' innehåller två noder för varje nod v i G: den "jämna" noden $(v, 0)$ och den "udda" noden $(v, 1)$. För varje kant från u till v i G finns det också två kanter i G' (med samma vikt som kanten i G): en kant från den jämna noden $(u, 0)$ till den udda noden $(v, 1)$, och en kant från den udda noden $(u, 1)$ till den jämna noden $(v, 0)$.

Exempel: Om G är grafen i uppgiftsspecifikationen så blir G' följande graf:



Notera att för varje väg

$$v_0 \xrightarrow{w_1} v_1 \xrightarrow{w_2} \ldots \xrightarrow{w_n} v_n$$

i G så finns det en motsvarande väg

$$(v_0, 0) \xrightarrow{w_1} (v_1, 1) \xrightarrow{w_2} \ldots \xrightarrow{w_n} (v_n, p)$$

i G', där p är 0 om n är jämn och 1 om n är udda. Vidare, för varje väg

$$(v_0, p_0) \xrightarrow{w_1} (v_1, p_1) \xrightarrow{w_2} ... \xrightarrow{w_n} (v_n, p_n)$$

i G' så finns det en väg

$$v_0 \xrightarrow{w_1} v_1 \xrightarrow{w_2} ... \xrightarrow{w_n} v_n$$

i G. Vi kan således avgöra om det finns en väg med ett udda antal steg från s till t i G genom att avgöra om det finns en väg från $(s, 0)$ till $(t, 1)$ i G', och i så fall har den kortaste (lättaste) vägen från $(s, 0)$ till $(t, 1)$ i G' samma vikt som den kortaste vägen med ett udda antal steg från s till t i G.

Om grannlistor används så kan algoritmen implementeras med följande tidskomplexitet:

- Konstruera G': $O(|V| + |E|)$.
- Avgöra om det finns en väg från $(s, 0)$ till $(t, 1)$ i G', och i så fall beräkna den minsta möjliga vikten för en sådan väg (med Dijkstras algoritm): $O(|V'| + |E'| \log |V'|) = O(|V| + |E| \log |V|)$.

Totalt: $O(|V| + |E| \log |V|)$, vilket nog får anses vara effektivt.

6. En Javaimplementation där mängderna representeras av bitarrayer:

```
public class Set {
    private boolean[] bits;

    public Set(int n) {
        bits = new boolean[n];
        for (int p = 0; p < n; p++) {
            bits[p] = false;
        }
    }

    public void insert(int i) {
        bits[i] = true;
    }

    public boolean member(int i) {
        return bits[i];
    }
```

```
    public void union(Set s) {
        for (int p = 0; p < bits.length; p++) {
            bits[p] = bits[p] || s.bits[p];
        }

        s.bits = null;
    }
}
```

Det är lätt att se att operationernas värstafallstidskomplexiteter uppfyller kravet för trea. En amorterad analys med (en positiv konstant gånger) summan av alla bitarrayernas längder som potential visar att vi också uppfyller kravet för fyra. (Notera att den amorterade analysen inte skulle fungera om raden `s.bits = null;` togs bort.)