Design and Analysis of Algorithms
Exam / example solutions
18.10.2012

Mikko Koivisto
[signature]
Instructor

General advice

A. Know what you can do, and what perhaps not.

B. You are expected to solve some things routinely.

C. Think 10 min, write 10 min — for each problem.

D. Write main things first, may add details later.

E. Do as much as you can, nobody is perfect.

F. You typically get points even if your solution is not perfect.

Specific comments per problem

1. Routine. Basics. Know the master method!

2. Routine. Very similar to an example in CLRS.

3. Routine, up to a point. Exer. 34.5-5 of CLRS.

4. Non-routine, yet elementary.

5. Routine. Taken from CLRS, p. 1124.

6. Non-routine, yet a variant of Exer. 34.4-6 of CLRS.

1. The master method applies:

9/9

(a) $n = \Theta(n^{\log_3 3}) \Rightarrow T(n) = \Theta(n \log n)$.

(b) $\log_2 n = O(n^{\log_3 2 - 1/8}) \Rightarrow T(n) = \Theta(n^{\log_3 2})$.

---

2. Optimal substructure: $(\ell < k)$

$(*)$ $i_1 \cdots i_k$ opt. for $\{1, \ldots, n\} \Rightarrow i_1 \cdots i_\ell$ opt. for $\{1, \ldots, i_{\ell+1}\}$.

9/9

Let
$$f(j) := \min \left\{ \sum_{s=0}^{\ell} c(i_s+1, i_{s+1}) : \begin{array}{l} i_1 \cdots i_\ell \in \{1, \ldots, j-1\} \\ \text{increasing} \\ i_0 = 0,\ i_{\ell+1} = j \end{array} \right\}.$$

We have $f(0) = 0$ and, by $(*)$,
$$f(j) = \min \{ f(i) + c(i+1, j) : 0 \leq i < j \}.$$

Compute $f(j)$ for $j = 1, 2, \ldots, n$ by dynamic programming:

```
introduce arrays f[0..n] and p[0..n].
f[0] ← 0
for j ← 1 to n
        f[j] ← ∞
        for i ← 0 to j-1
                if f[i] + c(i+1, j) < f[j]
                        f[j] ← f[i] + c(i+1, j)
                        p[j] ← i
while p[n] > 0
        print p[n]
        n ← p[n]
```

Running time clearly $O(n^2)$.

3. Clearly in NP because for a given $A \subseteq S$ one can verify in poly-time whether or not $\sum_{x\in A} x = \sum_{x\in S\setminus A} x$.

Reduction: Let (S, t) be an instance of SUBSET-SUM. Let $S' := S \cup \{u\}$, where $u := \sum_{x\in S} x - 2t$.

9/9

Clearly S' can be constructed in poly time.

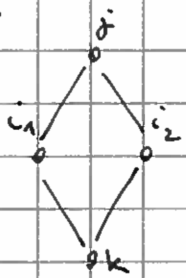
Equivalence:

$$\sum_{x\in A} x = t \iff \sum_{x\in A\cup\{u\}} x = t+u \text{ and } \sum_{x\in S\setminus A} x = \sum_{x\in S} x - t$$

$$\iff \sum_{x\in A\cup\{u\}} x = \sum_{x\in S} x - t = \sum_{x\in S'\setminus(A\cup\{u\})} x .$$

Thus $\exists A\subseteq S$ s.t. $\sum_{x\in A} x = t$ iff $\exists A'\subseteq S'$ s.t. $\sum_{x\in A'} x = \sum_{x\in S'\setminus A'} x$.

4. Correctness: The alg. returns YES iff for some (j, k) it executes line 4 at least 2 times, equivalently, there exist i_1 and i_2 such that $j i_1 k$ and $j i_2 k$ are two simple paths, equivalently, there is a simple cycle $j i_1 k i_2 j$ of four edges.



9/9

Running time: Lines 4 – 7 can be executed at most 2 times for any pair (j, k). Thus, the running time is bounded by the number of pairs (j, k), which is $O(n^2)$. This dominates the complexity of lines 2 – 7, as the pairs (j, k) can be visited efficiently for any i, due to the adj. list repres. Line 1 clearly takes $O(n^2)$ time.

5. 9/9

<u>Algorithm:</u> for each variable X_i, let $X_i \leftarrow \text{Random}\{0,1\}$.
(clearly poly-time)

<u>Analysis:</u> For each clause C of ϕ define

$$Y_C = \begin{cases} 0 & \text{if } C \text{ is not satisfied} \\ 1 & \text{if } C \text{ is satisfied.} \end{cases}$$

Observe $Y_C = 0$ iff C contains three distinct variables and the corresponding literals evaluate to 0. This happens with pr. at most $(1/2)^3 = 1/8$ (pr. is 0 if conflicting literals). Thus $\mathbb{P}\{Y_C = 1\} \geq 1 - 1/8 = 7/8$.
The expected number of satisfied clauses is

$$\mathbb{E}\Big(\sum_C Y_C\Big) = \sum_C \mathbb{E}(Y_C) = \sum_C \mathbb{P}\{Y_C = 1\} \geq 7/8 \cdot \#\text{of clauses}.$$

---

6. 9/9

<u>Notation:</u> $G-s$ is the graph obtained from G by removing s and all edges having s as end point.

<u>Observation:</u>

G contains a ham. path starting from s <u>iff</u>
$G-s$ contains — " — from t for some neighbor t of s in G.

<u>Algorithm:</u>

1 $H \leftarrow \emptyset$.
2 While G not empty.
3 Find a neighbor t of s in G such that $G-s$ contains a ham. path starting from t.
4 Add (s,t) to H.
5 $G \leftarrow G-s$; $s \leftarrow t$
6 return H.

Correct by the observation. Step 3 uses alg. $\mathcal{A}$.

Step 3 calls $\mathcal{A}$ at most $(j-1)$ times for a graph on j vertices.

$\therefore$ takes time $O\Big(\sum_{j=1}^{n} (j-1)\beta^j\Big) = O(n \cdot \beta^n)$, as $\sum_{j=1}^{n} \beta^j = \frac{\beta^{n+1}-1}{\beta-1}$

<u>Faster:</u> In step 3, instead of linear search, use binary search for t: Iteratively, discard a half of the neighbors, keeping the safe half, until a valid t is singled out. $\Rightarrow O(\beta^n \log n)$ time.