Sample Exam [1]

Oct. 15 2009
kl 1400 – 1800
byggnad V

Ansvarig:
Devdatt Dubhashi Tel. 073 932 2226 Rum 6479 EDIT

Poäng:	60	
Betygsgränser:	Chalmers	5:48, 4:36, 3:24
	GU	VG:48, G:28
	Doktorander	G:36
Hjälpmedel:	kursboken, anteckningar	

- Recommended: First look through all questions and make sure that you understand them properly. In case of doubt, do not hesitate to ask.
- **Answer all questions in the given space on the question paper (the stapled sheets of paper you are looking at). The question paper will be collected from you after the exam. Only the solutions written in the space provided on the question paper will count for your points.**
- Use extra sheets only for your own rough work and then write the final answers on the question paper.
- Answer concisely and to the point. (English if you can and Swedish if you must!)
- Code strictly forbidden! Motivated pseudocode or plain but clear English/Swedish description is fine.

Lycka till!

[1] 2005 läsperiod 1, INN200 (GU) / TIN090 (CTH).

Problem 1 Finding the m smallest [10] An array T contains n numbers. You want to find the m smallest numbers where m is much smaller than n. Would you

(a) sort T and pick the first m,

(b) call `select(T,i)` for $i = 1, 2, \ldots, m$, or

(c) use some other method?

Justify your answer by comparing the time complexity of the different methods.
Solution: (a) $O(n + logn + m) = O(n \log n)$. (b) $O(nm)$ using linear time SELECT algorithm. Better is to use $SELECT(T, m)$ to find mth smallest element x and then run through the array to output all the elements at most x for a total time of $O(n)$.

Problem 2 Changing Coins [10] Suppose that you have coins available in denominations (i.e. values) of $1, p, p^2, \ldots, p^n$, where $p > 1$ and $n \geq 0$ are integers (for example, $1, 5, 25$ with $p = 5$ and $n = 2$). You have an unlimited number of coins of each type. What algorithm will you use to find the minimum number of coins required to change a given amount? Give a short outline of a proof in justification.
Solution: Greedy algorithm which uses the largest coin possible at each step: for amount A use largest coin c at mos A, then repeat with $A - c$. To see optimality, consider an optimal algorithm and order the coins it uses from largest to least. Consider the first time it uses a different coin from the one used greedy algorithm, p. It uses smaller value coins from this point. Look at the first time these coins sum to a value $s \geq p^i$. We claim $s = p^i$ for otherwise since $s - p^i$ is divisible by the last coin used p^j, we can remove the last coin and still exceed p^i contradicting that this was the first time the sum exceeded p^i. Thus we may replace the coins adding to s by one coin p^i contradicting that we had an optimal algorithm. Thus the optimal algorithm agrees with greedy at each step.

Problem 3 Minimize Idle Time [10] Chalmers super-commputing center has its super computer running from 10 AM to 5 PM. Let's translate time and denote this interval $[0, M]$. At the start of the day it receives n requests where request i wants to reserve the supercomputer from time $0 \leq s_i$ to $f_i \leq M$. You want to schedule the jobs so that the supercomputer time is left idle as little as possible.

(a) Give counter-examples to show that neither of the follwing greedy heuristics gives the optimal solution: longest job first, earliest finishing job first. **Solution**: Pictures drawn in class.

Now we develop a dynamic programming algorithm. Let $OPT(i)$ denote the optimal solution considering only jobs $1 \cdots i$.

(b) First say why we may sort the jobs in order of finishing time and assume that the last finishing job has $f_i = M$. **Solution**: Without loss of generality we may number jobs in oredr of finishing time and we need only consider an interval $[s_i, f_j]$ where s_i is the earliest start time andf_i the latest finish time, because outside this interval, the supercomputer witll certainly be idle.

(c) What is $OPT(1)$? **Solution**: $OPT(1) = 0$.

(d) Give a recurrence for $OPT(i)$ based on what we do with job i. **Solution**:

$$OPT(i) = \min \Big(OPT(i-1) + f_i - f_{i-1}, OPT(p(i)) + s_i - f_{p(i)} \Big) .$$

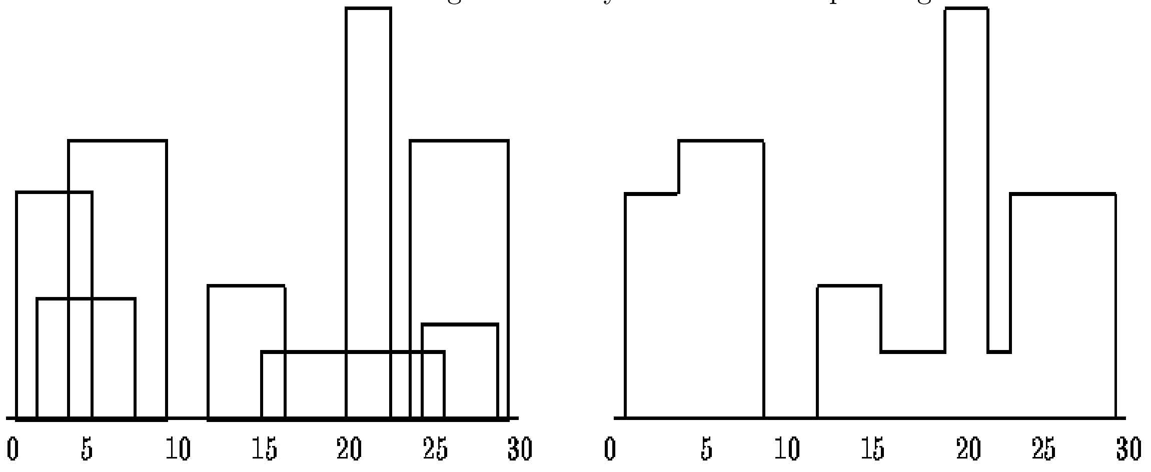
(e) Implement the recurrence efficiently in pseudocode and analyse its time and space complexity. **Solution**

```
M[1] = 0
for i= 2 to n do
  A = M[i-1] + f[i] - f[i-1] , B =  M[p[i]] + s[i] - f[p[i]]
  if A > B then M[i] = B else M[i]= A.
return M[n]
```

(f) How would you modify your algorithm to also produce a list of the jobs scheduled? **Solution** Keep track of which choice is made in the recursion:

```
M[1] = 0, b[1] = 1
for i= 2 to n do
  A = M[i-1] + f[i] - f[i-1] , B =  M[p[i]] + s[i] - f[p[i]]
  if A > B then M[i] = B , b[i] = 0 else M[i]= A. b[i] = 1
i= n
while i > n do if b[i] = 1 then output i, i = p[i]
                                else i = i -1.
```

Problem 4 Manhattan Skyline [10] With the advent of high speed graphics workstations, CAD (computer-aided design) and other areas (CAM, VLSI design) have made increasingly effective use of computers. One of the problems with drawing images is the elimination of hidden lines – lines obscured by other parts of a drawing.
You are to design a program to assist an architect in drawing the skyline of Manhattan given the locations of the buildings in the city. As we know, in Manhattan, all buildings are rectangular in shape and they share a common bottom (the city is very flat). The city is also viewed as two-dimensional. There are n buildings specified by three arrays $L[1 \dots n], R[1 \dots n]$ and $H[1 \dots n]$. Building i has left side at coordinate $L[i]$, right coordinate at $R[i]$ and height $H[i]$. In the figure below, buildings are shown on the left with $L = (12, 2, 14, 19, 3, 1, 24, 23)$ $R = (16, 7, 25, 22, 9, 5, 28, 29)$ and $H = (7, 6, 3, 18, 13, 11, 4, 13)$. The skyline, shown on the right, is represented by the two arrays: $(1, 3, 9, 12, 16, 19, 22, 23, 29)$ and $(11, 13, 0, 7, 3, 18, 3, 13, 0)$ where the first array gives the x coordinate and the second the height of the skyline at the corresponding co-ordinate.



(a) Give the overall strategy of a Divide-and-Conquer algorithm. **Solution**: Divide the set of buildings into two roughly equal size parts. Compute the skylines of the two subsets recursively, and then combine them into an overall skyline.

(b) Explain clearly what the recursive subproblems should return. **Solution**: Each of the two sub-problems should return two lists representing the skyline of the corresponding skyline: the left subproblem returns the arrays x_L, h_L and thr right subproblem x_R, h_R.

(c) Explain clearly the Conquer step: how will you combine the solutions to the subproblem to get a solution for the original problem? This is the most crucial part! **Solution**: Like Mergesort, we scan the x–arrays from left to right and at each coordinate we find which of the two skyline is higher using the height arrays. Thus we merge the two x arrays and create the corresponding overall h array as we go along.

(d) Write a recurrence for the running time and give a short justification. Deduce the running time of the algorithm in $O()$ notation. **Solution**: The recurrence for the running time is $T(n) = 2T(n/2) + O(n)$ because the divide and combine steps can be done in time $O(n)$. The soltuion to the recurrence is $T(n) = O(n \log n)$.

Problem 5 Taxi Göteborg [10] You are working for Taxi G̈teborg and are dveloping software for an automated response system. At a given point in time, there are m taxis at different locations in the city and there are n customers requesting a taxi, also from various parts of the city. Customer i needs the taxi within t_i minutes. Using some other nifty software using Dijkstra with GPS, you can determine if taxi j can reach customer i within t_i minutes in G̈teborg traffic. Now you need to determine the maximum number of customers you can satisfy. Give an efficient algorithm and analyse its running time. **Solution** First create abipartite graph with taxis on the left, customers on the right, and connect taxi j to customer i if your GPS software says it can raech the customer in time t_i. Now create aflow network with a source s having edges to all taxis of capacity 1 a sink t which has edges from all customers with capacity 1 and edges int he bipartitte graph are of capacity 1 and directed from taxis to customers. There is a flow of integer value f from s to t exactly if f customers can be satisfied hence the maximum number of satisfied customers equals the max flow in the network. Since all capacities are 1, the Ford Fulkerson algorithm can compute this in time $O(nm)$.

Problem 6 Multiple Interval Scheduling [10] We've seen the Interval Scheduling problem in class; here we consider a computationally much harder version of it that we'll call MULTIPLE INTERVAL SCHEDULING.As before, you have a processor that is available to run jobs over some period of time. (E.g. 9 AM to 5 PM.)
People submit jobs to run on the processor; the processor can only work on one job at any single point in time. Jobs in this model, however, are more complicated than we've seen in the past: each job requires a *set* of intervals of time during which it needs to use the processor. Thus, for example, a single job could require the processor from 10 AM to 11 AM, and again from 2 PM to 3 PM. If you accept this job, it ties up your processor during those two hours, but you could still accept jobs that need any other time periods (including the hours from 11 to 2).
Now, you're given a set of n jobs, each specified by a set of time intervals, and you want to schedule as many jobs as possible. Call this problem OPT-MS. Consider the

decision version of the problem. Y/N-MS: For a given number k, is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time?

(a) Suppose someone gives you a black box to solve the decision version Y/N-MS in polynomial time. Show that then you could use it to solve OPT-MS also in polynomial time. **Solution**: Run Y/N-MS with $k = 1, 2...$ until you get a "no" answer. The tunnning time is at most $np(n)$ where $p(n)$ is running time of Y/N-MS.

(b) Show that Y/N-MS is in NP. **Solution**: A certificate is a set of k jobs and the certifying algorithms simply checks that no two of these jobs has an interval in common, clearly polynomial time.

(c) State in one line the significance of showing that this problem is NP-complete. **Solution**: It is very unlikely that there is an efficient i.e. polynomial time algorithm for this problem.

(d) Show that Y/N-MS is NP-complete. (Hint: reduction from INDEPENDENT-SET: think of the nodes of a graph as job requests with the incident edges as its associated time intervals.). **Solution**: iven a graph $G = (V, E)$ and $k \geq 0$, divide the interval 10 Am to 5 Pm into m disjoint intervals, one corresponding to each edge in E. Now create n jobs one corresponding to each vertex in V. The intervals for the job corresponding to $v \in V$ are those corresponding to the edges incident on v in$G$. It is easy to see that two jobs conflict iff they correspond to vertices that are adjacent in G. Thus a set of jobs can be scheduled exactly if te corresponding vertices form an independent set in G. Thus there is a set of at least k jobs that can be scheduled iff there is an independent set of size at lest k in G.