Algorithms Exam [1]

Oct. 29 2014
kl 8:30 - 12:30
väg och vatten salar

Ansvarig:
Chien-Chung Huang Tel. 031 772 1699 Rum 6455 EDIT

Points :	60	
Grades:	Chalmers	5:48, 4:36, 3:28
	GU	VG:48, G:28
	PhD students	G:36
Helping material :	course textbook, notes	

- 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.
- Try to give the most efficient solution you can for each problem - your solution will be graded on the basis of its correctness *and* efficiency – a faster algorithm will get more credit than a slower one. In particular a brute force solution will not get any credit.
- 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] 2014 LP 1, DIT600 (GU) / TIN093 (CTH).

Problem 1 Stable Marriage [5] Suppose that we have a set of boys {Leonard, Raj, Sheldon} and a set of girls {Amy, Bernadette, Penny}.

(a) [2 pts] Construct their preferences so that there is only one stable matching.
(Sketch of the) Solution
$L : ABP$
$R : BPA$
$S : PAB$

$A : LRS$
$B : RSL$
$P : SLR$

(b) [3 pts] Construct their preferences so that there are three stable matchings.
(Sketch of the) Solution
$L : ABP$
$R : BPA$
$S : PAB$

$A : RSL$
$B : SLR$
$P : LRS$

Problem 2 Graph [5] Given a graph $G = (V, E)$, give a polynomial time algorithm to decide whether the graph is bipartite or not. Remember that a bipartite graph means that the set V of vertices can be separated into two parts U and W so that all edges connect a vertex in U and a vertex in W. (Hint: What is so special about the cycle in a bipartite graph?)

(Sketch of the) Solution
There is no odd-length cycle in a bipartite graph. There can be many ways of solving the problem. For instance, use DFS or BFS to find out a cycle, and if it is of odd length, report not, otherwise, remove it and continue. Report correct in the end.

Problem 3 Flow and Cut I [10] Let $G = (V, E)$ be a directed simple graph with capacity $c : E \to R_{\geq 0}$. Two special vertices s and t are the *source* and the *destination*. In our lecture, we explain the capacity scaling max-flow algorithm in detail. The following is called the "bit-scaling algorithm" of a very similar spirit. Suppose that each capacity $c(e)$ is represented as a K bit binary number. (We can always add some zeros if necessary to make each capacity exactly K-bits.) Let us define a sequence of problem P_1, P_2, $\cdots$, P_k, where P_k represents the problem with the same network $G = (V, E)$ and the *truncated* capacities c_k on the edges e. That is, for each edge, its capacity $c_k(e)$ is the number represented by the first k bits of $c(e)$.
Let f_k^* be the maximum flow in problem P_k, for all $1 \leq k \leq K$. For simplicity, we also define f_0^* is be the zero flow, i.e., $f_0^*(e) = 0$ for all $e \in E$. Also let $c_0(e) = 0$ for all $e \in E$.

(a) [2 pts] Argue that $2f_{k-1}^*$ is still a flow in the problem P_k.

(Sketch of the) Solution
Observe that $c_k(e)$ is either $2c_{k-1}(e)$ or $2c_{k-1}(e)+1$. So on every edge $e \in E$, $2f^*_{k-1}(e) \leq 2c_{k-1}(e) \leq c(e)$. So the capacity constrain is not violated in problem P_k. It is easy to see that the flow conservation for each vertex ($\neq s, t$) is also satisfied.

(b) [6 pts] How much bigger can the flow value of f^*_k be compared to $2f^*_{k-1}$? (Hint: Look at the min-cut in problem P_{k-1})

(Sketch of the) Solution

Let C be the min-cut in problem P_{k-1}. The flow value of f^*_k is bounded by $\sum_{e=(u,v),u\in C,v\notin C} c_k(e)$ (weak duality).

Observe that $2f^*_{k-1}$ has the property (1) for each $e = (u, v)$ with $u \in C$ and $v \notin C$, $c_k(e) \leq 2f^*_{k-1}(e)+1$, and (2) for each $e = (u, v)$ with $u \notin C$ and $v \in C$, $2f^*_{k-1}(e) = 0$, and (3) the flow value of $2f^*_{k-1}$ is computed as $\sum_{e=(u,v),u\in C,v\notin C} 2f^*(e) - \sum_{e=(u,v),u\notin C,v\in C} 2f^*_e$.

These observations implies that the flow value of f^*_k can only be m larger than $2f^*_{k-1}$.

(c) [2 pts] So we should have an idea of how to design the algorithm now. Find the maximum flow f^*_1 in P_1. Double it (i.e., multiply the flow on each edge by the factor of 2) and treat it as the initial flow in P_2. Find the next maximum flow f^*_2 in P_2 by augmentation. Continue in this manner for the following problems P_3, $\cdots$, P_K. As P_K is just the same as the original problem. The maximum flow f^*_K in P_K is the solution to the original problem. So what is the running time of this algorithm? You should justify your answer.

(Sketch of the) Solution
As f^*_k can be only m larger than $2f^*_{k-1}$. There can be only m augmentations for each problem P_k. Each augmentation takes $O(m)$ time. So for each problem P_k, we need $O(m^2)$ time. In total we need $O(Km^2)$ time.

Problem 4 Flow and Cut II [15] Let $G = (V, E)$ be a directed graph with capacity $c : E \to R_{\geq 0}$. Two special vertices s and t are the *source* and the *destination*. Furthermore, suppose that we also allow s to have incoming edges and t to have outgoing edges.

(a) [4 pts] Recall that in our lecture, we said that to find a maximum flow, it suffices to repeatedly find an augmenting path from s to t in the residual graph G_f, where f is the current flow. Suppose that we now find a directed path p from s to t in G_f. Explain why is that f augmented along the path p is still a flow. Your answer should cover all cases. (You should explain what is "augmentation" and how it is done, as well as why it still results in a flow).

(Sketch of the) Solution
We need to define a residual graph and what is meant by finding a path in the residual graph, how the flow is updated in augmentation, and argue that after the augmentation, the flow conservation law and the capacity constraints are satisfied. (For the last part, one has to take into account that some edges in the

residual graph are forward arcs and some are backward arcs and there are four cases to take care of).

(b) [4 pts] Suppose that for some edges e, the capacity $c(e)$ can be infinite. Then clearly the flow values can be unbounded (then, there is no solution to the maximum flow problem), depending on the network. Nonetheless, can we check whether there is a maximum flow of bounded value? You are allowed to use an algorithm that takes exponential time.

(Sketch of the) Solution
As the flow value is always bounded by the cut size (weak duality), we just have to check whether there exists an s-t cut whose size is finite. This can be done by considering all s-t cuts.

(c) [7 pts] We are given a set J of jobs and a machine. Each job $j \in J$ has a release time r_j and a deadline d_j and it takes exactly one hour to process the job. We assume that release times and deadlines are always the beginnings of the hours, e.g. 7AM, 5PM and so on.

Suppose that the machine has different processing capacities c at different hours of the day, for instance $c(3AM - 4AM) = 8$ means that during the period of 3AM to 4 AM, the machine can process up to 8 jobs.

We have to solve the decision question: is there a feasible schedule to process all jobs in J? Give a polynomial time algorithm and argue that your algorithm is correct. (Hint: Maximum flow is the apparent way to go.)

(Sketch of the) Solution

Create 24 nodes, each of which represents an hour and each of which has an arc towards the sink and the capacity of that arc is the capacity of the machine during that hour.

Next create a node for each job. The source s has an outgoing edge of capacity one toward each such job node. Finally, draw an arc from a job node to an hour node with capacity 1 if that hour is within the release time and the deadline of that job.

Find the maximum flow in this network and if the flow value is $|J|$, the number of jobs, then return yes, otherwise, no. The correctness of the algorithm follows from the fact that each feasible schedule corresponds to a feasible flow in the constructed network.

Problem 5 Longest Dominance Sequence [13] Suppose that we are given a table T. Each cell (i, j) has a positive integer $T(i, j)$ in it. The cell (i, j) dominates (i', j') if the following three conditions hold. (1) $i' < i$, (2) $j' < j$, and (3) $T(i', j') < T(i, j)$. (Clearly, the dominance relationship is transitive). The goal here is to find the longest dominance sequence (i_1, j_1)... (i_t, j_t), i.e., (i_a, j_a) dominates (i_{a-1}, j_{a-1}), for $2 \leq a \leq t$. This problem can be solved by dynamic programming as follows. Let us build another table D. In the cell $D(i, j)$ is recorded the longest dominance sequence ending at cell (i, j).

(a) [1 pts] What is $D(1, j)$ and $D(i, 1)$ for all i and j?

(Sketch of the) Solution
One.

(b) [6 pts] Suppose that right now we want to fill in the entry $D(i, j)$, assuming that $i > 1$ and $j > 1$. What other cells in D will be required for filling $D(i, j)$? You should explain in details how the value of $D(i, j)$ is going to be decided using those other cells that are already filled in (i.e. you should give the recurrence formula).

(Sketch of the) Solution

$D(i, j) = 1 + \max_{(i',j'),\text{where } i'<i,j'<j,T(i',j')<T(i,j)} D(i', j')$.

(c) [2 pts] The above recurrence should suggest a strategy to you: what is the order you are going to enter all the entries in D?

(Sketch of the) Solution

One possibility is the row by row. Namely
$(1, 1), \cdots, (1, n), (2, 1), \cdots, (2, n), \cdots (n, 1), \cdots, (m, n)$.

(d) [2 pts] After we fill in all the entries in D, how do we find the solution to the original problem?

(Sketch of the) Solution
Just check all entries $D(i, j)$.

(d) [2 pts] What is the running time of your algorithm? Is it really polynomial?
(Sketch of the) Solution
We need $O(m^2n^2)$ time and it is really polynomial.

Problem 6 NP-Completeness [12] For the following two problems, you should argue that (1) they belong to the class of NP, and then (2) prove that they are NP-complete by reduction.
The following is the PARTITION problem. Given a set of numbers $A = \{a_1, a_2, \cdots, a_n\}$, can we divide them into the subsets whose sums are equal? That is, can we divide A into two groups A_1 and A_2 so that $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$?
PARTITION problem is known to be NP-complete.

(a) [6 pts] Show that k-PARTITION is also NP-complete. k-PARTITION asks whether we can divide A into k groups A_1, A_2, $\cdots$, A_k so that $\sum_{a_i \in A_u} a_i = \sum_{a_i \in A_v} a)i$ for any $1 \leq u \leq v \leq k$. Prove that k-PARTITION is NP-complete. (Hint: Obviously, the reduction should be from PARTITION.)

(Sketch of the) Solution

The problem is in NP because the certifier can just check any possible partition of A into k parts and see if the sums in all these parts are equal.

Let $A = \{a_1, a_2, \cdots, a_n\}$ be the partition instance. We create a new instance $A' = \{a_1, a_2, \cdots, a_n, a_{n+1}, a_{n+2}, \cdots a_{n+(k-2)}\}$. Let $a_{n+1} = a_{n+2} = \cdots a_{n+(k-2)} =$

$\frac{\sum_{i=1}^{n} a_i}{2}$. It can be verified that A can be partitioned into two parts if and only if A' can be partitioned into k parts, where each of $\{a_{n+1}, \cdots a_{n+(k-2)}\}$ is a single part by itself.

(b) [6 pts] The following is the unrelated machine scheduling problem. We are given a set J of jobs and a set M of machines. Each job $i \in J$ has a processing time p_{ij} on machine $j \in M$. Jobs are to be processed on the machines. The load of a machine $i \in M$ is the sum of the processing times of the subset $J' \subseteq J$ of jobs assigned to it, i.e., $\sum_{j \in J'} p_{ij}$. The decision version of this problem asks whether there is an assignment so that the load of every machine is bounded by a certain value t.

Show that the problem is NP-complete. (Hint: The reduction could be from the previous problem.)

(Sketch of the) Solution

The problem is in NP because the certifier can just check any possible assignment of the jobs on the machines and see if the total load on each machine is bounded by t.

Let $A = \{a_1, a_2, \cdots, a_n\}$ be a k-partition instance. Create a set M of k machines first. For each number a_k, create a job j_k with $p_{ik} = a_k$ for all $i \in M$. Furthermore, let $t = \frac{\sum_{i=1}^{n} a_i}{k}$. The instance A can be equally partitioned into k parts if and only if there is an assignment where each machine has exactly the load of t.