Algorithms Exam [1]

Oct. 22 2012 **kl 14 - 18** **"väg och vatten"** **salar'**

Ansvarig:
Devdatt Dubhashi Tel. 073 932 2226 Rum 6479 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 corectness *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] 2012 LP 1, INN200 (GU) / TIN090 (CTH).

Problem 1 Planning a Party [10] Moa is planning her housewarming party. She wants to invite as many of her friends as possible. However, she wants each person she invites to have at least 5 other people they already know (so they don't feel totally lost), and at least 5 others they don't already know (so they could make new friends). She has made a graph with a vertex for each of her friends, and an edge between two friends if they know each other.

(a) [4 pts] Give a $O(n^2)$ time algorithm to find the largest such set of people she could invite (return empty set if it is impossible).

(b) [3 pts] Give a brief inductive argument for correctness.

(c) [3 pts] Give a brief justification of how the algorithm can be implemented in $O(n^2)$ time.

Solution to Problem 1:

a) Let $|V|$ be the set of vertices in the input graph G While ($|V|$ is not $\emptyset$)
Find $v \in V$ s.t. v has degree less than 5
Find $v \in V$ s.t. v has degree more than than $|V| - 5$
If v is found, remove it with all adjacent edges
Else terminate cycle
End While
Return $|V|$

b) Correctness: the algorithm provides a correct answer after n iterations.
Proof - induction on number of iterations:
Base case - no iterations at all. If there are no such vertices, then everyone has 5 friends and 5 not-yet-friends. Correct answer is $|V|$
Induction step - suppose output for $n = k$ iterations is correct. Consider graph that requires $n = k + 1$ iterations. Some vertex is removed on the first iteration, it is easy to see that it can't be in the solution (it has either too many or too few friends, and its degree can only decrease with other vertex removal). Therefore graphs before and after 1st iteration are equivalent instances. According to induction, we get a correct answer for an instance, that we obtain after k following iterations, thus algorithm works correctly.

c) As every edge is removed at most once, total number of iterations is linear on number of (removed) edges. Remark: worst case is reachable on full graph (everyone is friends to others).

Problem 2 Buying a new car[10] You are given a NAVTEQ map of Sweden with all the cities and the highways connecting them in the form of a graph $G = (V, E)$. Each stretch of highway between two cities has a length $\ell_e \geq 0, e \in E$ (in km). You want to drive from Göteborg to Kiruna. There's only one problem: you car can hold enough gas to travel L km. There are gas stations at the cities but not in between. Therefore you can only travel a route if each one of its intercity stretches e has length at most $\ell_e \leq L$.

(a) [3 pts] Give a *linear* time algorithm to find if it is possible at all to go from Göteborg to Kiruna.

1. Remove all edges e from G such that $\ell_e > L$
2. Use breadth-first search on the resulting graph, starting in Göteborg to determine if a path to Kiruna exists.

Time complexity: 1. At most $O(|V|+|E|)$; 2. BFS can be done in $O(|V|+|E|)$ } linear!

(b) [3 pts] You are now planning to buy a new eco–friendly "green" car with a small fuel capacity. You would like to find the mnimum fuel tank capacity that is needed to go from Göteborg to Kiruna. Describe a $O(|V| + |E| \log |V|)$ time algorithm to solve this problem.

1. Construct a minimum spanning tree (MST) using Prim's algorithm on G, with edge weights ℓ_e for $e \in E$. Start the algorithm in Göteborg. At each point an edge (u, v) is included, store $p(v) = u$.
2. Find a path from Göteborg to Kiruna and return the maximum edge weight ℓ_e for edges e on that path. The path is found by recursively using $p(\cdot)$ starting in p(Kiruna).

```
u = Kiruna; lmax = 0
while (u != Göteborg)
    lmax = max(l_(p(u),u), lmax)
    u = p(u)
```

If no path exists, return "no path"

(c) [2 pts] Argue that your algorithm is correct.

By the cut property of MSTs (4.17), every MST contains every edge e such that e is the min-cost edge with one end in $S \subset V$ and one in $V - S$ for a set S. This means that any u→v path in a MST of G is the "narrowest" or minimum maximum cost path from u to v. This means that the path we construct above requires the least fuel tank capacity.

(d) [2 pts] Justify the running time with the use of appropriate data structures and algorithms.

With heap-based priority queues, Prim's algorithm runs in $O(|E|\log|V|)$. Our modification with storing $p(u)$ adds a constant term at each iteration which does not change the asymptotic complexity. Finding the maximum k_e as described in b) requires at most looking at each node, hence $O(|V|)$. Total: $O(|V|+|E|\log|V|)$

Problem 3 Merging again[10] You have k lists each of size n each *already sorted.* Your task is to merge them into one sorted list.

(a) [2 pts] One way to do this is by successively merging the lists: merge first two to get a list of length $2n$, then merge the third with this to get a list of length $3n$ etc. How many comparisons does this require? Give your answer in $O(\cdot)$ notation as a function of k and n.

(b) [4 pts] Describe a"Divide–and–Conquer" algorithm to solve the problem.

(d) [2 pts] Justify the running time with the use of appropriate data structures and algorithms.

Problem 3 Merging again[10] You have k lists each of size n each *already sorted.* Your task is to merge them into one sorted list.

(a) [2 pts] One way to do this is by successively merging the lists: merge first two to get a list of length $2n$, then merge the third with this to get a list of length $3n$ etc. How many comparisons does this require? Give your answer in $O(\cdot)$ notation as a function of k and n.

Comparisons $(n,k) = \sum_{i=2}^{k} i\,n - 1 = n \sum_{i=2}^{k} i - (k-1)$

$= \frac{n}{2}(k+2)(k-1) - (k-1) = \frac{n}{2}(k^2+k-2) - (k-1)$

$\in O(nk^2)$

(b) [4 pts] Describe a"Divide–and–Conquer" algorithm to solve the problem.

```
merge(L) {// L array of lists
  let n = size L; let m = ⌈n/2⌉
  if (n==1) return L[0]; // base case, one list
  if (n==2) { let l1 = L[0], l2 = L[1], n1 = size l1, n2 = size l2
              let l be list of size (n1+n2); i=0, j=0, c=0
              while (i<n1 || j<n2) {
                if (j>=n2 || (i<n1 && l1[i]<=l2[j])
                  l3[c] = l1[i]; else l3[c] = l2[j]
                c++;}
  if (n>2) then merge (merge ([L[0],...,L[m-1]],
                       merge ([L[m+1],...,L[n-1]]))
```

(c) [4 pts] Write a recurrence for the running time of the algorithm and solve the recurrence (in $O(\cdot)$ notation) giving a brief justification.

$T(k,n) = 2T(k/2, n) + kn - 1$ take $k = 2^j$

(Divide, merge)

$= 2[2T(2^{j-2}, n) + 2^{j-1}n - 1] + 2^j n - 1$

$\vdots$

$= 2^2 T(2^{j-2}, n) + 2^j n - 2^1 + 2^j n - 2^0$, $T(1,n) = 1$

$\vdots$

$= 2^j \cdot T(1,n) + j2^j n - \sum_{i=0}^{j-1} 2^i = 2^j + j2^j n + (1 - 2^j)$

$= k + (\log k) kn + C \in \Theta(nk \log k)$

Problem 4 Yuckdonanlds[10] Yuckdonald's is considering opening a series of restaurants along the Swedish west coast. The n possible locations are on a straight line (basically) and the distances along the line from the start at the bottom of Skane are (in integer km) and increasing order: $m_1, m_2, \cdots, m_n$. The constraints are:

- At each location Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location i is $p_i, \geq 0$, for $i = 1, \ldots n$.
- Any two resturants should be at least distance k km apart (where k is a fixed positive integer).

Let $OPT(i)$ denote the maximum profit Yuckdonalds's can expect to get considering only the first i locations.

(a) [1 pt] In this notation, what is the final solution we want, and a base case where the value is easy to compute? (Give the base case value!)

(b) [4 pts] Write a recurrence for $OPT(i)$.

(c) [2 pts] Using (b) and (c), implement the recurrence efficiently in pseudocode.

(c) [4 pts] Write a recurrence for the running time of the algorithm and solve the recurrence (in $O(\cdot)$ notation) giving a brief justification.

Problem 4 Yuckdonanlds[10] Yuckdonald's is considering opening a series of restaurants along the Swedish west coast. The n possible locations are on a straight line (basically) and the distances along the line from the start at the bottom of Skane are (in integer km) and increasing order: $m_1, m_2, \cdots, m_n$. The constraints are:

- At each location Yuckdonald's may open at most one restaurant. The expected profit from opening a restaurant at location i is $p_i, \geq 0$, for $i = 1, \ldots n$.
- Any two resturants should be at least distance k km apart (where k is a fixed positive integer).

Let $OPT(i)$ denote the maximum profit Yuckdonalds's can expect to get considering only the first i locations.

(a) [1 pt] In this notation, what is the final solution we want, and a base case where the value is easy to compute? (Give the base case value!) FINAL : $OPT(m)$
BASE : $OPT(0) = 0$
$P_0 = 0$

(b) [4 pts] Write a recurrence for $OPT(i)$.

ASSUMPTION : max$S$ RETURNS THE GREATEST INTEGER OF A SET

$prev(i) = \max S(\{j \mid 1 \leq j < i \ \& \ m_j + K \leq m_i\} \cup \{0\})$
$(i \in [1, m])$

$OPT(i) = \max(p_i + OPT(prev(i)), OPT(i-1))$

(c) [2 pts] Using (b) and (c), implement the recurrence efficiently in pseudocode.

```
opt_calc {
  OPT[0..m] := 0;
  For i:=1 to n do {
    max := OPT[prev[i]];
    total := max + p_i;
    if (total > OPT[i-1])
      (2)
      then OPT[i] := total;
      else OPT[i] := OPT[i-1];
  }
  // EXERCISE 4.e
  return OPT[n];
} 5
```

NOTE
THE FUNCTION prev IN 4.b CAN BE PRE-COMPUTED IN $O(m)$ TIME AND STORED IN THE ARRAY prev

(d) [1 pt] What is the time and space complexity of your algorithm?

TIME $O(m)$ SPACE $O(m)$

(e) [2 pts] How do you actually find the optimal set of places to open the restaurants (not just the optimal profit value)? Give just a short description of what modifications you make to (c).

REPLACE THE COMMENT IN PSEUDOCODE 4.C WITH:

```
while (i ≥ 1) {   // WHEN EXECUTED THE FIRST TIME i = m
  if (OPT[i] > OPT[i-1]) {
    then printlm i;
         i := prev[i];
    else i := i-1;
  }
}
```

Problem 5 Are you at risk of cancer? [10] A common machine learning task is classification: a typical example is that we have a set of n microarray data from n patient from their tissue sample and we want to classify which of them are susceptible to cancer. For each patient, we have done some statistical tets on the microarray data to get *likelihood* values $0 \leq p_i \leq 1$ for whether patient i is cancerous (assume these are rational numbers). In addition, based on other patient record data, we also have *similarity* values $0 \leq S(i,j) \leq 1$ for how similar patient $i$and patient j are with respect to several physiological factors: $S(i,j) = 0$ means patients i and j are not similar at all, $S(i,j) = 1$ means they are very similar (assume these are also rational values). Our belief is that if two patients are similar, then they are both likely to be susceptible or both non susceptible. Our task is to classify the patients as susceptible $\ell_i = 1$, or not susceptible $\ell_i = 0$. We would like to find a labelling ℓ, to *maximize* the score:

$$q(p,S) := \sum_{i:\ell_i=1} p_i + \sum_{i:\ell_i=0} (1-p_i) - \sum_{(i,j):\ell_i \neq \ell_j} S(i,j).$$

(The last term is a *penalty* for labelling two similar patients differently.)

(d) [1 pt] What is the time and space complexity of your algorithm?

(e) [2 pts] How do you actually find the optimal set of places to open the restaurants (not just the optimal profit value)? Give just a short description of what modifications you make to (c).

Problem 5 Are you at risk of cancer? [10] A common machine learning task is classification: a typical example is that we have a set of n microarray data from n patient from their tissue sample and we want to classify which of them are susceptible to cancer. For each patient, we have done some statistical tets on the microarray data to get *likelihood* values $0 \leq p_i \leq 1$ for whether patient i is cancerous (assume these are rational numbers). In addition, based on other patient record data, we also have *similarity* values $0 \leq S(i,j) \leq 1$ for how similar patient $i$and patient j are with respect to several physiological factors: $S(i,j) = 0$ means patients i and j are not similar at all, $S(i,j) = 1$ means they are very similar (assume these are also rational values). Our belief is that if two patients are similar, then they are both likely to be susceptible or both non susceptible. Our task is to classify the patients as susceptible $\ell_i = 1$, or not susceptible $\ell_i = 0$. We would like to find a labelling ℓ, to *maximize* the score:

$$q(p,S) := \sum_{i:\ell_i=1} p_i + \sum_{i:\ell_i=0} (1-p_i) - \sum_{(i,j):\ell_i \neq \ell_j} S(i,j).$$

(The last term is a *penalty* for labelling two similar patients differently.)

let the number of patients be n, $\sum_{i=1}^{n} p_i + (1-p_i) = n$

maximizing $q(p,S)$ is equivalent to minimizing

$$q'(p,S) = n - q(p,S) = \underbrace{\left(\sum_{\ell_i=1} p_i + \sum_{\ell_i=0} (1-p_i) + \sum_{\ell_i=0} p_i + \sum_{\ell_i=1} (1-p_i) \right)}_{=n} - q(p,S)$$

$$= \sum_{\ell_i=0} p_i + \sum_{\ell_i=1} (1-p_i) + \sum_{(i,j):\ell_i \neq \ell_j} S(i,j)$$

(a) [7 pts] Construct a network so that by computing a maximum flow on it, you can find a classification that maximizes the score $q(p, S)$. State how you get the classifying labels from the computed maxflow.

network $G=(V,E)$, where $V=\{s,t\} \cup P$ ← set of patients
E constructed as follows. for each $v_i \in P$ (s, v_i) with capacity p_i
and (v_i, t) with capacity $1-p_i$. For each $v_i, v_j \in P$
two edges (v_i, v_j), (v_j, v_i) with capacities $s(i,j)$.
: a cut A-B in this network will have
· $v \in A$, (v,t) flows out of A to B; $v \in B$, (s,v) flows from A-B
· $v_i \in A$, $v_i \in B$ then (v_i, v_i) flows from A to B.
Thus taking s,t as super source, sink. if A-B is a min
cut the value of $f^{out}(A)$ is minimal, and easily seen to
minimize $q'(p,S)$. from max flow we get the mincut (7.9) Book
from the min cut A-B if $v_i \in A$ then $l_i = 1$, $v_i \in B$ then $l_i = 0$
let $|P|=n$ # patients
$|E| = m = 2n^2 - n + 2n + n = 2n^2 + 2n$

(b) [3 pts] What is the running time of your algorithm?

Considering that the capacities Pi are integer, the correct complexity would be O(C n^2), C <= n, thus O(n^3).

Problem 6 How to be a Master Chef [10] We want to become celebrity chefs by creating a new dish. There are n ingredients and we'd like to use as many of them as possible. However, some ingredients don't go so well with others: there is a $n \times n$ matrix D giving the *discord* between any two ingredients i.e. $D[i, j]$ is a real value between 0 and 1: 0 means i and j perfectly well together and there is no discord and 1 means they go very badly together. Any dish prepared with these ingredients incurs a *penalty* which is the sum of the discords between all pairs of ingredients in the dish. We would like the total penalty to be small. Consider the decision problem EXPERIMENTAL CUISINE: can we prepare a dish with at least k ingredients with total penalty at most p?

(a) [3 pts] Show that Experimental Cuisine is in $\mathcal{NP}$.

(b) [5 pts] Show that Experimental Cuisine is $\mathcal{NP}$-complete by giving a reduction from a problem known to be $\mathcal{NP}$-complete).

(c) [2 pts] What is the significance of (a) and (b) for the question of whether there is a fast algorithm to prepare a dish with the maximum number of ingredients that does not exceed a given penalty?

Solution to Problem 6

a) Certificate: a dish with at least k ingredients with total penalty of at most p.
Verifier: go through ingredients and calculate the penalty and the total number of ingredients (takes at most quadratic time).

b) Given an instance of IS, (G, k), where G is a graph, and k is a parameter, construct the following:
Set D to be an incidence matrix of G, i.,, $D[i, j] = 1$, iff vertices i and j are connected by an edge, otherwise $D[i, j] = 0$
Set $p = 0$.
(D, k, p) is an instance of Experimental Cuisine.
Claim: (D, k, p) returns YES for Experimental Cuisine iff (G, k) returns YES for Independent Set. Proof: straightforward "=¿" and "¡=". Take instances, observe that a collection of vertices form an IS iff corresponding $p = 0$. k parameter stands for size of solution in both cases.

c) optimization version is NP-hard and is unlikely to have a polynomial time solution. (Otherwise we could simply solve a corresponding decision-version NP-complete problem in polynomial time and prove that $P = NP$).