Purdue University — S. Hambrusch-C. Guerra
CS 381 — Fall 1999

Solution Sketches to Sample Final Exam Questions

1.) (i) Let Π_1 and Π_2 be two decision problems. Assume $\Pi_1 \leq_P \Pi_2$. If there exists a polynomial time algorithm for Π_1, does this imply that there exists a polynomial time algorithm for Π_2?

No. It is possible that Π_1 is in class P and Π_2 is NP-complete.

(ii) Let Π_{dec} be an NP-complete decision problem and let Π_{opt} be its corresponding optimization problem. Assume Π_{opt} can be solved in polynomial time. What does this imply for Π_{dec}? What does this imply for the class of NP-complete problems and the class NP, respectively?

It implies that the decision problem Π_{dec} can be solved in polynomial time. Since Π_{dec} is NP-complete, this implies that every problem in NP can be solved in polynomial time. Hence, P=NP would follow.

(iii) Does there exist a polynomial time algorithm to determine whether an undirected graph contains a clique of size 3? Explain.

Yes. Assume the graph is represented by an adjacency matrix of size $n \times n$. There exist n choose 3 triples which could form a clique of size 3. Try each triple and check whether the three edges needed are in the graph. This results in an $O(n^3)$ time algorithm.

This algorithm is brute-force (but all that was asked for was a polynomial-time solution.) Improving the time is trickier than it appears. For example, using DFS and using backedges to detect triangles does not work in $O(n + m)$ time.

(iv) If an NP-complete problem can be solved deterministically in $O(n^3)$ time, can every problem in class NP be solved in $O(n^3)$ time?

It implies that every problem in NP can be solved in polynomial time. It does not imply that the polynomial is $O(n^3)$.

(v) Let G be an connected, undirected, and weighted n-vertex graph. Each one of its m edges has weight 2. Describe and analyze an algorithm that finds a minimum spanning tree of G. Your algorithm should be faster than Kruskal's algorithm.

Any spanning tree is a minimum spanning tree. Use DFS or BFS to generate a spanning tree in $O(n + m)$ time.

2.) For each of the problems listed below state the asymptotic running time of the best algorithm you know. If you think the problem is NP-complete, state so (you do not need to give a running time). You do not need to give details about the algorithm. If you wish to make any comments, please limit them to two lines.

1. Sorting n integers $a_1, a_2, \ldots, a_n$ with $0 \leq a_i \leq n^2 \log^2 n$.

   Using radix sort, we can sort in $O(n)$ time. Observe that a_i can be represented by the triple (x_i, y_i, z_i) with $a_i = z_i + n * y_i + n^2 * x_i$, $0 \leq x_i, y_i, z_i < n$.

2. Determining the $\frac{n}{5}$-th largest element in an unsorted set of size n.

   Sorting costs $O(n \log n)$ time. Using the linear time selection algorithm results in $O(n)$ time (this algorithm was not described in class, only mentioned).

3. In a directed, weighted graph $G = (V, E)$ with positive weights and $|V| = n$ and $|E| = m$, determine the shortest path between a given pair of vertices.

   There exists no algorithm to determine the shortest path between a given pair of vertices i and j faster than determining the shortest path from i to all vertices. Using Dijkstra's algorithm gives $O(m \log n)$ time ($O(n^2)$ time is also possible and can be faster for dense graphs).

4. In an n-node rooted tree T, determine the number of leaves whose parent has more than one child.

   $O(n)$ time.

5. Given a boolean formula in conjunctive normal form (i.e., $C_1 \wedge C_2 \wedge \ldots \wedge C_k$, where every C_i contains an arbitrary number of literals $\vee$-ed together), determine whether there exists a truth assignment to the variables satisfying the formula.

   This problem is NP-complete.

6. Given a boolean formula in disjunctive normal form (i.e., $C_1 \vee C_2 \vee \ldots \vee C_k$, where every C_i contains an arbitrary number of literals $\wedge$-ed together), determine whether there exists a truth assignment to the variables satisfying the formula.

   Choose one clause C_i which contains no contradiction (i.e., a variable and its negation) and satisfy C_i by setting all literals to true. This satisfies the entire formula. Can be done in linear time.

3.) Assume $G = (V, E)$ is an undirected, connected, weighted graph, $|V| = n$, $|E| = m$, represented by adjacency lists. Weights can be positive as well as negative.

For each problem listed below, give the asymptotic time bound of the best algorithm you know. Give a 2-sentence explanation of your solution.

(i) Determine whether G contains at least 10 edges of cost ≥ 100:

$O(n+m)$ time (BFS or DFS traversal).

(ii) Determine whether G is a tree:

$O(n)$ time; we only need to consider $n-1$edges (assumes we don not include the time to create the adjacency list structure).

(iii) Find a spanning tree of G having minimum cost:

$O(m \log n)$ using Kruskal's algorithm

(iv) Given two vertices u and v, does there exist a path from u to v:

it states that G is connected; hence it is $O(1)$ time; if we would not know that G is connected, it is $O(n+m)$ time.

(v) Given vertices u and v, determine the length of the longest path from u to v:

NP-complete.

4.) Let $G = (V, E)$ be a directed graph with integer weights on its edges, $|V| = n, |E| = m$. Let s and t be two vertices in G and k be an integer. Consider the problem of determining whether the cost of the longest path from s to t is at least k. (In case there exists no path between s and t, return the answer "no".)

What is the complexity of this problem for the three classes of graphs stated below? Either describe an efficient polynomial time algorithm or show that the problem is NP-complete.

(i) G is a rooted tree.

In a tree there exists a unique path between two vertices. Hence, any path-finding algorithm can be used. $O(n)$ time.

(ii) G is an acyclic graph.

For an acyclic graph, use topological search. This costs $O(n+m)$ time.

(iii) G is an arbitrary graph.

NP-complete for arbitrary graphs. We know that Hamiltonian path is NP-complete. One can show that deciding whether there exists a Hamiltonian path between a given pair of vertices is also NP-complete. (If it were not, we can use such an algorithm n^2 times and solve the original problem in polynomial time). We then ask whether the graph contains a longest path of cost $n-1$.

5.) A point $p_i = (x_i, y_i)$ dominates point $p_j = (x_j, y_j)$ if $x_i \geq x_j$ and $y_i \geq y_j$. Given n points in arbitrary order, describe and analyze an efficient algorithm which determines the points *not* dominated by any other point.

$O(n \log n)$ time algorithm: sort points by x-coordinates; scan list from largest x-value to smallest and keep largest y-value seen so far; $O(1)$ time to determine whether a point is dominated or not.

6.) (i) Let A be a set of n numbers given in unsorted order. Consider the problem of determining $x \in A$, $y \in A$ such that $|x - y| \geq |w - z|$ for all $w \in A$, $z \in A$. State an efficient algorithm and its running time.

$O(n)$ time; the minimum and the maximum are x and y.

(ii) Let A be a set of n numbers given in unsorted order. Consider the problem of determining $a_i \in A$, $a_j \in A$ such that $|a_i - a_j| \leq |a_q - a_r|$ for all $q \neq r$. Describe an efficient algorithm for determining a_i and a_j.

$O(n \log n)$ time: sort the elements and then scan to determine the adjacent pair with minimum difference.

7.) The partition problem is defined a follows: Given a set A, determine whether the elements of A can be partitioned into two sets so that the sum of the elements in one set is equal to that of the elements in the other set. This problem is known to be NP-complete.

Consider the following 1-processor scheduling problem: Given are n jobs and job i has length l_i, penalty p_i, and deadline d_i associated with it. The n jobs are to be scheduled on the processor. If job i is not completed by time d_i, a penalty of p_i occurs.

In the Min_Pen problem we are given l_i, p_i, d_i, $1 \leq i \leq n$, and a quantity P and are to determine whether there exists a schedule such that the sum of the arising penalties is at most P. Show that the Min_Pen problem is NP-complete.

First, show that problem Min_Pen is in NP: the certificate is a schedule; check the schedule to see that it is a valid one and determine the penalty resulting from the jobs not scheduled before their deadline. This can be done in polynomial time ($O(n \log n)$ is possible).

Next show that Partition $\leq_{poly}$ Min_Pen. Let $A = \{s_1, \ldots s_m\}$ be an instance of the partition problem with $\sum_{i=1}^{n} = 2M$. Create $m = n$ jobs with $l_i = s_i$, $p_i = s_i$, $d_i = M$, and $P = M$. Clearly, this transformation takes polynomial time.

To complete the proof we need to show that Partition has a solution if and only if Min_Pen has a solution. Assume that Partition has a solution. Then there exists a set S of indices such that $\sum_{k \in S} s_k = M$. Schedule the jobs corresponding to elements in S before time M on the processor. All of them finish before their deadline and experience no penalty. The remaining jobs experience a total penalty of M. Hence, Min_Pen has a solution.

Assume now that the Min_Pen problem has a solution. In order for the penalty to be at most M, the schedule contains a job which terminates at time M (if this were not the case, the penalty would be more than M). Hence, the jobs scheduled before time M correspond to elements in the set whose sum is exactly M. This implies that the partition problem has a solution.