CPSC 629: Analysis of Algorithms, Fall 2003

Solutions to Homework 5

Solution to 1 (34.1-1)
Suppose the LONGEST-PATH-LENGTH problem can be solved in polynomial time by an algorithm A. Given an instance of LONGEST-PATH $\langle G, u, v, k\rangle$, run the algorithm A on $\langle G, u, v\rangle$. Compare k with the result k' returned from A. $\langle G, u, v, k\rangle$ is a "YES" instance of LONGEST-PATH if and only if $k \leq k'$. The running time is the same as A, which is a polynomial.

Conversely, if the LONGEST-PATH problem can be solved in polynomial time by an algorithm A, then given an instance of LONGEST-PATH-LENGTH $\langle G, u, v\rangle$, run the algorithm A on $\langle G, u, v, k\rangle$ for $k = 1, 2, \ldots, n-1$. The smallest k such that $\langle G, u, v, k\rangle$ is a "YES" instance of LONGEST-PATH is the solution for the LONGEST-PATH-LENGTH problem on instance $\langle G, u, v\rangle$. It is clear that the running time is n times the running time of A, which is still a polynomial.
□

Solution to 2 (34.1-4)
The dynamic programming algorithm of running time $O(nW)$ is not a polynomial time algorithm, because W is the maximum weight of items that the thief can put into his knapsack. Without loss of generality, assume that the input of the problem, W, is given as a binary number. Then the size of the input is $O(n \log W)$. The running time $O(nW)$ is not a polynomial in $O(n \log W)$ because for any fixed constant c, $nW > (n \log W)^c$ for sufficiently large W.

Comments on common errors: The W is not independent of the input size. It is given as part of the input. It is reasonable to assume that W is given as a binary number in the input. □

Solution to 3 (34.1-6)
If two languages $L_1, L_2 \in P$, then there exists algorithms A_1, A_2 that decide membership in L_1, L_2, respectively, in polynomial time. Given an instance $I \in L_1 \cup L_2$, run the algorithm A_1 on I and then run A_2 on I. We decide $I \in L_1 \cup L_2$ if and only if A_1 or A_2 or both return "YES".

If a language $L \in P$, then there exists an algorithm A that decides membership in L in polynomial time. Given an instance $I \in \overline{L}$, run the algorithm A on I. We decide $I \in L$ if and only if A returns "NO". □

Solution to 4 (34.2-1)
Define a certificate for the language GRAPH-ISOMORPHISM to be a permutation π that maps vertices of G_1 to vertices of G_2. The verification algorithm first checks that π is a permutation, then checks that (u, v) is an edge if and only if $(\pi(u), \pi(v))$ is an edge. The algorithm runs in time $O(V + E)$.

Comments on common errors: It is not enough to check the degrees of the vertices in two graphs. For example, a 2-regular graph can be a single cycle or a collection of smaller cycles. □

Solution to 5 (34.2-4)
If two languages $L_1, L_2 \in NP$, then there exists algorithms A_1, A_2 that verify certificates for memberships in L_1, L_2, respectively, in polynomial time. Given an instance $I \in L_1 \cup L_2$ and a certificate C, run the algorithm A_1 and A_2 on C. We decide $I \in L_1 \cup L_2$ if and only if at least one of them accepts C.

The same approach as for proving P is closed under complement does not work for NP. If a language $L \in NP$, then there exists a polynomial-time algorithm A that verifies the certificates of L. But in order to prove that $\overline{L} \in NP$, we need a polynomial-time algorithm A' that verifies the certificates of $\overline{L}$. Note that the algorithm A does not help in finding A'. For example, given a 3-CNF formula, the certificate of its satisfiability is a satisfying assignment, but there is no known short certificate of its unsatisfiability. In fact, it is unknown whether NP is closed under complement, i.e., whether NP=co-NP. It is a famous open problem and it is widely believed that they are not equal. □

Solution to 6 (34.3-3)
By definition of reducibility, if $L \leq_P \overline{L}$, then there exists a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$x \in L$ if and only if $f(x) \in \overline{L}$.

Then the same function f also satisfies that for all $x \in \{0,1\}^*$,

$x \in \overline{L}$ if and only if $f(x) \in L$,

which is equivalent to saying that $\overline{L} \leq_P L$ □

Solution to 7 (34.4-6)
Denote by A the polynomial time algorithm to decide formula satisfiability. Given a formula F, run A on F, if A says "NO", return "NOT EXIST"; else do the following for each variable v_i, $i = 1, \ldots, n$:

assign 1 to v_i, and run A on F with remaining unassigned variables;
if A says "NO", assign 0 to v_i.

In the end, all the variables in F are assigned value 0 or 1. It is easy to use induction to prove that this is a satisfying assignment. □

Solution to 8 (34-1)

- a) A decision problem for independent-set problem:

  INDEPENDENT-SET $= \{\langle G, k\rangle$: G has an independent set of size at least $k.\}$

  Define the certificate to be a set of nodes of size at least k. It is easy to verify the certificate. To show that INDEPENDENT-SET is NP-complete, we reduce the clique problem to it. Given an instance $\langle G, k\rangle$ of the clique problem, take the complement of G, denoted by $\overline{G}$. Then G has a clique of size k if and only if $\overline{G}$ has a independent set of size k. This proves that CLIQUE $\leq_P$ INDEPENDENT-SET.

- b) Run the subroutine for the INDEPENDENT-SET problem on $\langle G, k\rangle$ for $k = 1, \ldots, n$ and find the largest $k = k_0$ such that the subroutine returns "YES". Pick a vertex v in G. Denote by $G(\overline{v})$ the subgraph of G constructed by removing v and its induced edges. Run the subroutine on $\langle G(\overline{v}), k_0\rangle$. If the subroutine returns "YES", then recursively find an independent set of size k_0 in $G(\overline{v})$. If the subroutine returns "NO", then include v in the independent set and recursively find an independent set of size $k_0 - 1$ in $G(\overline{v})$. It is obvious that the algorithm described above finds the maximum independent set in polynomial time.

- c) Given an instance $\langle G, k\rangle$ of the INDEPENDENT-SET problem, where each vertex in G has degree 2. Observe that a 2-regular graph G is a collection of disjoint cycles, i.e., $G = C_1 \cup \ldots \cup C_k$, where C_i's are cycles and $C_i \cap C_j = \emptyset$ for any $i \neq j$. For each cycle C_i, the size of the maximum independent set in the subgraph induced by C_i is $\lfloor |C_i|/2 \rfloor$. Since the cycles are disjoint, then the size of the maximum independent set of G is $k_0 = \sum_i \lfloor |C_i|/2 \rfloor$. The algorithm returns YES if and only if $k \leq k_0$.

- d) Let the bipartite graph be $G = L \cup R$, where $L \cup R = G$ and $L \cap R = \emptyset$. Construct a flow network F as follows: add a source s, for every vertex l_i in L, add an edge (s, l_i) with capacity 1; add a sink t, for every vertex r_i in R, add an edge (r_i, t) with capacity 1; for every edge in G, direct it from L to R, and add infinite capacity. Use the max flow algorithm to find the min cut in the above flow network F. For every vertex l_i in L, if (s, l_i) is not in the cut, include l_i in a set I. Similarly, for every vertex r_i in R, include r_i in the set I if (r_i, t) is not in the cut. The set I is an independent set, because for any pair of vertices v_i, v_j in it, if (v_i, v_j) is an edge, then (v_i, v_j) has to be cut, resulting in a infinite large cut. The running time is $O(VE^2) = O((n+2)(n+m)^2) = O(nm^2)$.

Comments on common errors: In part (c), a 2-regular graph is not necessarily a single cycle. It could consist of a collection of disjoint cycles. In part (d), simply taking one side does not guarantee a maximum independent set. An alternative solution to part (d) that uses maximum matching goes as follows: First find a maximum matching M in the bipartite graph. Include into a set S all vertices that are not induced by edges in M. For each edge in M, at least one end is not connected to any vertex in the set S. The reason is that if there is an matched edge whose both ends are connected to vertices in S, then there must be an augmenting path in the graph, contradicting the fact that M is maximum. Take that "free" end and put it into the set S. This process can continue until every matched edge has exactly one edge in S. This method gives a maximum independent set of size $n - m$ in a bipartite graph, where m is the size of the maximum matching M. This is so called König-Egervary Theorem.
□

Solution to 9 (34-3)
Clearly the transformation is polynomial time.

Check that the 3SAT input formula is satisfiable if and only if the transformed input for 3COLOR (the graph) is 3-colorable.

⇒: Assume the formula is satisfiable. Fix a satisfying truth assignment. Color the nodes of the graph like this:

- If variable x_i is true, then color node x_i blue and $\overline{x_i}$ green, for all i.
- If variable x_i is false, then color node x_i green and $\overline{x_i}$ blue, for all i.
- Color node TRUE blue.
- Color node FALSE green.
- Color node RED red.
- For each clause of the formula, color its five nodes appropriately. There are 7 cases, depending on which of its literals are true (note that at least one literal is true, since we have a satisfying truth assignment). The details are left to you.

⇐: Assume the graph is 3-colorable. Fix a 3-coloring. Let blue be the color of the node TRUE, green the color of the node FALSE, and red the color of the node RED. Each x_i node is either green or blue and $\overline{x_i}$ is the other color (blue or green). Use these colors to get a truth assignment: variable x_i is true if and only if node x_i is blue. Suppose this truth assignment is not satisfying. So some clause does not contain a true literal. You can verify that the corresponding part of the graph cannot be 3-colored, which is a contradiction.
□

Solution to 10 (35.1-5)
The answer is no. As a counterexample, assume a graph G has a maximum clique of size $n/2$ (assume n is even). Then the complement graph $\overline{G}$ has minimum vertex cover of size $n - n/2 = n/2$. A ratio-2 approximation algorithm A for vertex cover may return a vertex cover of size $n-1$ on $\overline{G}$, which corresponds to a clique of size 1 in G. The ratio of this approximation is $n/2$.
□

Solution to 11 (35.2-2)
For any instance of the TSP problem, let c_0 be the maximum distance between any pair of cities. Increase the distance between any pair of cities by adding c_0. Then it is obvious that the new costs satisfied the triangle inequality. But the 2-approximation algorithm for the TSP problem with the triangle inequality does not imply a constant ratio approximation algorithm for general TSP. To give a counterexample, assume the original instance of the TSP problem has a tour of minimum cost $C_{opt} = 2 \cdot c_0$. Then the converted TSP problem has a tour of minimum cost $C'_{opt} = 2 \cdot c_0 + n \cdot c_0 = (n+2)c_0$. The 2-approximation algorithm may return a tour of cost $C' = 2n \cdot c_0$, which corresponds to a tour in the original instance of cost $n \cdot c_0$. The ratio of this approximation is $n/2$.
□

Solution to 12 (35-1)

- a) Given an instance $\langle S, t\rangle$ of the subset sum problem, let Φ be the sum of all numbers in S. Without loss of generality, assume that every number in S is are strictly between 0 and t, and that $\Phi \geq t$. We can also assume that $t \geq \Phi/2$, because otherwise we can let $t' = \Phi - t$ and solve the instance $\langle S, t'\rangle$. Divide every number in S by t, and have a new set S', which is a collection of numbers whose values are strictly between 0 and 1. Let ϕ be the sum of numbers in set S'. We have $1 \leq \phi \leq 2$. Add into S' a new number $2 - \phi$. The set S' is an instance of the bin packing problem. It is easy to verify that the instance $\langle S, t\rangle$ of the subset sum problem is TRUE if and only if the set S' can be packed into two bins. If bin packing is solvable in P, then there exists an algorithm A that solves any instance of bin packing in polynomial time. With the above reduction, the algorithm A can also be used to solve any instance of the subset sum problem in polynomial time. This finishes the proof that the bin packing problem is NP-hard.
- b) $\lceil S\rceil - 1$ bins of unit size cannot contain a set of items whose total size is S. Thus the optimal number of bins is $k_{opt} \geq \lceil S\rceil$.
- c) If there are two bins b_i, b_j that are both less than half full, $i < j$, then all items in b_j can fit into b_i. They will be put into b_i instead of b_j according to the first-fit heuristic.
- d) First observe that elements in the smallest two bins combined have size larger than 1. For the rest of the bins, each is at least half full. Therefore, if the minimum number of bins required is k, then $S > 2k$. It follows that $k \leq \lceil 2S\rceil$.
- e) Let $S = \widehat{S} - \widetilde{S}$, where $\widehat{S}$ is an integer and $0 \leq \widetilde{S} < 1$. Then $\lceil S\rceil = \widehat{S}$, and $\lceil 2S\rceil = \lceil 2\widehat{S} - 2\widetilde{S}\rceil \leq 2\widehat{S} = 2\lceil S\rceil$. Therefore $k \leq \lceil 2S\rceil \leq 2\lceil S\rceil \leq 2k_{opt}$.
- f) Do a preprocessing of ordering the objects by their size. Place the objects into bins in increasing order of their size. Since for every object, the search for the first bin to fit starts from the end of the last search, all the searches are done in one pass. Therefore, the first-fit heuristic itself takes $O(n)$ time. In total, the running time is $O(n \log n + n) = O(n \log n)$.

Comments on common errors: In part (a), pay attention to the direction of the reduction. Keep in mind that the subset sum problem asks for a *subset* whose sum is a given number. In part (f), binary search tree does not satisfy the requirement of the First Fit heuristic, because it does not necessarily give the *first* bin that can fit the object. □

Note: The solutions given here are terse, and in some cases, incomplete. Your answers should be complete and have more details. But you will lose points if they are too long or too complex.