

COT 5405 - Fall 2009

Homework 6(solution)

April 21, 2009

Grading Policy:

- Please contact TA Chunchun Zhao by email or in his office hours for any grading issues.
- Maximum score is 100 points.
- Each completed question worth 10 points.
- Problem 3 is graded for 60 points(15 points for each).
- Partial credit is given if you dont answer a question completely.
- Please notify the TA if you find anything wrong in this solution.

Problem 1 (Exercise 32.4-1)

Compute the prefix function p for the pattern `ababbabbabbababbabb` when the alphabet is $S = \{a, b\}$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
$\pi[i]$	0	0	1	2	0	1	2	0	1	2	0	1	2	3	4	5	6	7	8

Problem 2 (Problem 34-1)

An independent set of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in E is incident on at most one vertex in V' . The independent-set problem is to find a maximum-size independent set in G .

1. Formulate a related decision problem for the independent-set problem, and prove that it is NP-complete. (Hint: Reduce from the clique problem.)
2. Suppose that you are given a "black-box" subroutine to solve the decision problem you defined in part(a). Give an algorithm to find an independent set of maximum size. The running time of your algorithm should be polynomial in $|V|$ and $|E|$, where queries to the black box are counted as a single step. Although the independent-set decision problem is NP-complete, certain special cases are polynomial-time solvable.
3. Give an efficient algorithm to solve the independent-set problem when each vertex in G has degree 2. Analyze the running time, and prove that your algorithm works correctly.
4. Give an efficient algorithm to solve the independent-set problem when G is bipartite. Analyze the running time, and prove that your algorithm works correctly. (Hint: Use the results of Section 26.3.)

Part a:

A decision problem for the independent set problem could be like this: Independent Set = $\{ \langle G, k \rangle : \text{graph } G \text{ has an independent set with size } k \}$

The first step is to prove that independent set problem is in NP . For a given graph $G = \langle V, E \rangle$ and k , we use the V' which is the subset of V in the independent set as a certificate for G . The verification algorithm affirms that $|V'| = k$, and then it checks that there is no edge between any pair of vertices in V' . This verification can be done in polynomial time. Thus, Independent set problem is NP . If there is a proof that clique problem can be reduced to Independent Set problem, Independent set problem can be claimed to be NP-complete. The reduction algorithm is to take the input of clique problem $\langle G, k \rangle$ and compute the complement G' by doing this in polynomial time. Then, G has a clique of size k if and only if G' has an independent set of size k . Because clique and be reduced to Independent set problem, the Independent set problem is NP-Complete.

Part b:

Run the subroutine on G from $k = 1$ to n and find the largest K_{max} for which the subroutine returns "yes". Construct a graph G_v by removing the vertex v and all edges inducing v in G . Run the subroutine for $\langle G_v, K_{max} \rangle$. If it returns "yes", the vertex v should not be included in Independent set and then recursively run the subroutine on G_v to find the independent set with size K_{max} . If it returns "No", the vertex v is included in the independent set and then removing all neighbors of v and recursively running the subroutine for $\langle G_v, K_{max}-1 \rangle$. This algorithm is obviously running in polynomial time on $|V|$ and $|E|$.

Part c:

If each vertex in G has degree 2, the G is a collection of disjoint cycles. In each cycle, the vertexes selected are not neighbors to each other. For a single cycle C_i , the max size of independent set is $\lfloor |C_i|/2 \rfloor$. Therefore, the max size of independent set in G is $K_{max} = \sum \lfloor |C_i|/2 \rfloor$ for all cycles. The algorithm returns "yes" if and only if $k \leq K_{max}$.

Part d:

Assume the $G = L \cup R$. Add a source vertex s and a target vertex t into the G to construct a network flow. Add edges (s, L_i) to each vertex in L with capacity 1 and add edges (R_j, t) with capacity 1. Add infinite capacity to each edge directing from L to R . Now using Max flow algorithm to find the min cut in the flow network. Every vertex in L and R which is not in the cut should be included in the set IS . Any pair of vertex (L_i, R_j) in IS has no edge because if (L_i, R_j) is an edge it should have been cut by an infinite large cutting. Therefore the set IS is an independent set. The running time for the algorithm is $O(VE^2)$.

Problem 3 (Exercise 34-2)

Bonnie and Clyde have just robbed a bank. They have a bag of money and want to divide it up. For each of the following scenarios, either give a polynomial-time algorithm, or prove that the problem is NP-complete. The input in each case is a list of the n items in the bag, along with the value of each.

1. There are n coins, but only 2 different denominations: some coins are worth x dollars, and some are worth y dollars. They wish to divide the money exactly evenly.
2. There are n coins, with an arbitrary number of different denominations, but each denomination is a nonnegative integer power of 2, i.e., the possible denominations are 1 dollar, 2 dollars, 4 dollars, etc. They wish to divide the money exactly evenly.
3. There are n checks, which are, in an amazing coincidence, made out to "Bonnie or Clyde." They wish to divide the checks so that they each get the exact same amount of money.

4. There are n checks as in part (c), but this time they are willing to accept a split in which the difference is no larger than 100 dollars.

Part a:

Let x and y be the two different denominations and n be the total number of coins. Let a be the coefficient for x so that a is the number of x -denomination coins. Therefore: $ax + (n - a)y = T$ where T is the total amount of money. Using the formula constraint above, check for each possible number of x -denomination coins if T can be divided evenly. This algorithm runs in $O(a)$ time.

Part b:

This problem can be solved in polynomial time. We can sort all coins by decreasing order first. Give Bonnie the largest one, then give Clyde coins until Clyde has the same amount as Bonnie. This will always happen if the rest of money is no less than their difference, because the denominations of coins are power of 2. When they have the same amount, give Bonnie the largest coin in the remaining coins, repeat the above process until all coins are gone. If after giving Bonnie one coin C , the sum of remaining coins is less than C . Then we can not find an evenly division. Otherwise, we can.

Part c:

NP complete. We prove this by reduction from the subset-sum problem:

- The problem is in NP since it is a trival matter to verify a given solution in polynomial time.
- Reduction from the subset-sum problem: Let the sum be $T/2$ where T is the total amount of money.

Part d:

NP-complete. To prove this problem is in NP-hard, it needs to reduce the problem in (c). To construct a set of checks in part (d), multiply every element in C by 1000, and call this new set C' . C can be divided evenly if and only if C' has a solution whose difference is not larger than 100. And this transformation is obviously polynomial time. Therefore, this problem is NP-complete.

Problem 4 (Exercise 35.2-4)

The bottleneck traveling-salesman problem is the problem of finding the hamiltonian cycle such that the cost of the most costly edge in the cycle is minimized. Assuming that the cost function satisfies the triangle inequality, show that there exists a polynomial-time approximation algorithm with approximation ratio 3 for this problem. (Hint: Show recursively that we can visit all the nodes in a bottleneck spanning tree, as discussed in Problem 23-3, exactly once by taking a full walk of the tree and skipping nodes, but without skipping more than two consecutive intermediate nodes. Show that the costliest edge in a bottleneck spanning tree has a cost that is at most the cost of the costliest edge in a bottleneck hamiltonian cycle.)

Solution:

Based on problem 23–3, the maximum-weight (costliest) edge in the BST is at most the weight/cost of the costliest edge in the a bottleneck Hamiltonian cycle. In first step, as hinted, is that we demonstrate that we can visit all the nodes in a bottleneck spanning tree (BST) exactly once. We do this by taking taking a full walk of the tree, skipping up to 2 consecutive intermediate nodes. This heuristic gives an approximation ratio of no more than 3 because by skipping no more than 2 intermediate nodes, we are no more than 3 edges away from the optimal solution at each iteration.