

COT 5405 - Summer 2008

Homework 5(solution)

July 5, 2008

Grading Policy:

- Please contact TA Shahed Nejhum by email or in his office hours for any grading issues.
- Maximum score is 100 points.
- Each completed question worth 10 points.
- 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 (Problem 33-2)

Part a:

Proof by contradiction. Let's assume there exist i and j , $i < j$, and $y_i < y_j$. Assume $p_i = (x_i, y_i)$. First we show that p_i has the maximum y -coordinate in L_i . If there exist a point, p'_i , in L_i whose y -coordinate is larger than p_i , then p'_i dominates p_i since p_i is the leftmost point in L_i thus its x -coordinate value is less than p'_i , which contradicts with the fact that p_i is in L_i . It is also the case for p_j . Since $y_i < y_j$, y_j is larger than all the y -coordinate value of all the points in L_i , which indicates that p_j is not dominated by any point in L_i . Thus it should also belong to L_i , which contradicts with our assumption. So if $i < j$, $y_i > y_j$.

Part b:

If $j \leq k$, to proof that the maximal layers of Q' is the same as of Q except $P = (x, y)$ is added into L_j , we need firstly to proof that P does not dominate any point in L_i where $i < j$. It is obvious since P is left to any point in Q thus it actually does not dominate any one in Q . Secondly, since $y_j < y < y_{j+1}$, the left most point in L_{j+1} (whose y -coordinate value is y_{j+1}) dominates P . Similarly, all leftmost points in L_i where $i < j$ dominate P . Hence, maximal layers L_i where $i < j$ remains the same. According to the result from (a), P is not dominated by any point in set $Q - \{L_j\}$ where $i < j$ because it has the largest y -coordinate value in that set. This fact directly indicates that P should be included in L_j . If $j = k + 1$, which means y is smaller than all leftmost points in the maximal layers of Q , P is dominated by at least one point of the maximal layers of Q . Hence it could only belong to a $(k + 1)$ st maximal layer which contains itself.

Part c:

First we sort all points in Q by their y -coordinate values. This step takes $O(n \log n)$ time. Then we simply sweep a line from right to left on Q and do the following: 1) Specify the y -coordinate value of the first point met as y_{max} 2) Continue sweeping, if the point met next has a y -coordinate value larger than y_{max} , then records it into current maximal layer. Set y_{max} to its y value; if the point has a smaller y value than y_{max} , finish recording the current maximal layer. 3) Remove all points recorded, start from 1) to record a new maximal layer.

Part d:

If $y = y', x > x'$, (x, y) still dominates (x', y') according to the definition. This can be solved by the following modifications of the algorithm in (c): If the sweep line finds two points share the same x-value, then the one with larger y-value is picked. In recording step 2), only strict larger points are recorded. Those with same y-value ones are left as the next layer recording.

Problem 2 (Exercise 32.4-1)

	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 3 (Problem 34-1)

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 $\text{floor}(|C_i|/2)$. Therefore, the max size of independent set in G is $K_{max} = \sum \text{floor}(|C|/2)$ 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 4 (Exercise 34-2)

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 5 (Exercise 35.2-4)

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 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.