CSIE 2136: Algorithm Design and Analysis (Fall 2015)

Final

Time: 14:20-17:20 (180 minutes) January 12, 2015

Instructions

- This is a 3-hour close-book exam.
- There are 6 questions worth of 100 points plus 35 bonus points. The maximum of your score is 100, so you can allocate your time and select problems based on certain optimal strategies of your own.
- Please write clearly and concisely; avoid giving irrelevant information.
- Please print **your name, student ID, and page number on every answer page**.

Problem 1: Short Answer Questions (25 points)

Answer the following questions and briefly justify your answer.

(a) (3 points) True or False: If P $\neq$ NP, then there is no polynomial-time algorithm to solve any NP-complete problem.

(b) (3 points) True or False: If A can be reduced to B in $O(n^2)$ time and there is an $O(n^3)$ algorithm for B, then there is an $O(n^3)$ algorithm for A.

(c) (3 points) True or False: Every computational problem is decidable.

(d) (3 points) True or False: It is more efficient to represent a sparse graph using an adjacency matrix than using adjacency lists.

(e) (3 points) True or False: If the amortized cost for every operation on a data structure is $O(1)$, the running time for performing a sequence of n operations is $O(n)$ in the worst case. (Assuming the data structure is empty at the beginning.)

(f) (3 points) True or False: If there is a randomized algorithm that solves a decision problem in time t and outputs the correct answer with probability 0.5, then there is a randomized algorithm for the problem that runs in time $\Theta(t)$ and outputs the correct answer with probability at least 0.99.

(g) (3 points) Provide a counterexample to show that the following divide-and-conquer algorithm *may not* correctly find a minimum spanning tree:

- Divide: Given a graph G, partition G into two parts by using a cut.
- Conquer: Find a MST for each part.
- Combine: Combine the two MSTs using the minimum edge of the cut.

(h) (4 points) Please write down up to three of your peers who help you most for the ADA class. Anything else you would like to say to your instructor or any suggestions to help improve next year's class?

Problem 2: Basic Graph Problems (25 points)

(a) (5 points) Given a pre-order traversal list $\{A, B, D, E, F, C\}$ and a post-order traversal list $\{D, F, E, B, C, A\}$, please reconstruct a legal **binary tree**. (There can be more than one solution.)

(b) (5 points) Given a BFS traversal list $\{A, B, C, E, D, F\}$ and a DFS traversal order $\{A, B, C, F, D, E\}$, please reconstruct a legal **undirected conntected graph**. (There can be more than one solution.)

(c) (5 points) Consider the graph in Figure 1. What is the shortest path distance from s to t computed by Dijkstra's algorithm (Figure 2)? What is the actual shortest path distance from s to t?

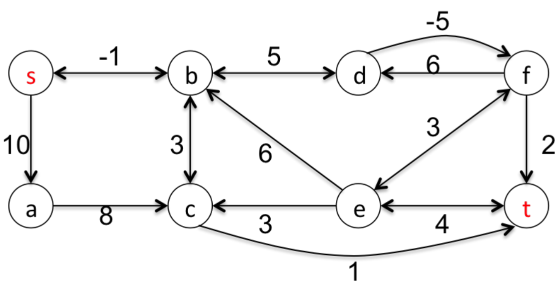

Figure 1: Find the shortest path distance from s to t

(d) (10 points) Given the path relaxation property, please explain why Bellman-Ford algorithm (Figure 3) can correctly compute the shortest path distance from the source s to another vertex t when there is no negative cycle.

Hint: Path relaxation property: If $p = \langle v_0, v_1, \cdots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

Problem 3: Approximation Algorithms (20 points)

in Homework 5, we learned how to reduce 3-CNF-SAT to k-CNF-SAT for any integer $k > 3$ in polynomial time. Answer the following related questions:

(a) (10 points) The MAX-k-CNF-SAT problem is similar to the k-CNF-SAT problem, except that it aims to return an assignment of the variables that maximize the number of satisfied clauses (i.e., clauses that evaluate to 1). Design a randomized $(2^k/(2^k - 1))$-approximation algorithm for k-CNF-SAT ($k > 3$). The $(2^k/(2^k - 1))$ approximation ratio means that $Pr[\text{a clause is satisfied}] = ((2^k - 1)/2^k)$.

Your algorithm should run in polynomial time with respect to the number of variables. To simplify this question, you can assume that no clause contains both a variable and its negation.

(b) (10 points) Your friend Ada claimed that, for any k, Ada can design a $(2^k/(2^k - 1))$-approximation algorithm for the MAX-3-CNF-SAT problem. Given a k and a formula in the 3-CNF form, her algorithm first converts the formula into the k-CNF form, and then solves the MAX-k-CNF-SAT problem (as described in part (a)). Do you agree with Ada? Justify your answer.

Problem 4: Independent Set (20 points)

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'. In other words, no two vertices in V' are joined by an edge.

The *independent set* problem is to find a maximum-size independent set V' in G. We denote by IND-SET the decision version of the independent set problem: Given a graph G and a value k, determine whether there is an independent set of size k in G.

Suppose we know 3-CNF-SAT is NP-Complete. To prove IND-SET is NP-Hard, your friend Ada constructs a polynomial-time algorithm to reduce 3-CNF-SAT to IND-SET as follows:

1. Let $\phi = C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_k$ be a Boolean formula in 3-CNF with k clauses, and each C_r has exactly 3 distinct literals l_1^r, l_2^r, l_3^r.
2. For each $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ, introduce a triple of vertices v_1^r, v_2^r, v_3^r in V.
3. Build an edge between v_i^r and v_j^s for the following two cases:
   (1) v_i^r and v_j^s are in the same triple (i.e., $r = s$);
   (2) v_i^r and v_j^s are in different triples but l_i^r is the negation of l_j^s.

(a) (10 points) Given an instance of the 3-CNF-SAT problem $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$, please draw the instance of the IND-SET problem according to the proposed reduction.

(b) (10 points) Please justify the correctness of this reduction. That is, please show that ϕ is satisfiable if and only if G has an independent set of size k.

Hint: The proof is very similar to the proof of 3-CNF-SAT $\leq_p$ CLIQUE. What is the relationship between clique and independent set?

Problem 5: Election Day (10 points + 15 bonus points)

The election day is coming. As a devoted citizen to the Great Pirate Kingdom, Ada signs up to be a poll worker to help validate and count election votes. Ada would like to apply techniques learned in the algorithm class to analyze and improve the election process.

(a) (10 points) Ada is assigned to operate a **Binary Counter** that displays one candidate's vote count in the binary form. If it costs 2^d to flip the dth bit, please justify that the amortized cost per increment is $O(\log n)$ and the total amortized cost is $O(n \log n)$ for counting n votes.

Hint: Recall that we learned in the class that when it costs 1 to flip a bit, the amortized cost per increment is $O(1)$ and the total amortized cost is $O(n)$ in a sequence of n increments.

(b) (5 bonus points) Given the citizen ID list of the Great Pirate Kingdom, Ada needs to ensure that only citizens on the list can vote. If the n IDs are sorted, determining whether one ID is indeed on the list takes $O(\log n)$ time. To speed up, Ada proposes to use a Bloom filter to store the IDs. Please provide one advantage and one disadvantage of using Bloom filters for this purpose.

(c) (5 bonus points) Ada claims to be able to accurately predict the election result. However, making a prediction right before the election is prohibited. To convince others, Ada computes and publishes $H(winner, r)$ where r is a random value and $H(\cdot)$ is a cryptographic hash function. Ada will reveal *winner* and r after the election. Please explain why Ada cannot cheat.

(d) (5 bonus points) The election decides the winner using the plurality rule. That is, each voter awards one point to his/her top candidate, and the candidate with the most points wins. Please construct an example demonstrating that plurality is not strategyproof when there are more than two candidates. Recall that a voting rule is strategyproof if a voter can never benefit from lying about his or her preferences.

Problem 6: More Independent Set (20 bonus points)

Now let us consider a variant: The *maximum weighted independent set* problem is to find an independent set V' in G such that the total weight of V' is maximized. Although finding a weighted maximum independent set in general is NP-hard, it can be done in polynomial time on certain special graphs. Consider a weighted line graph $G = \{V, E\}$ of n vertices $V = \{v_0, v_1, v_2, \cdots, v_n\}$ and $n - 1$ edges $E = \{(v_0, v_1), (v_1, v_2), \cdots, (v_{n-1}, v_n)\}$. The weight of v_i is w_i.

(a) (5 points) Provide a counterexample to show that the following greedy algorithm **may not** find an independent set of maximum total weight.

Algorithm 1 The heaviest-first greedy algorithm

```
$V' \leftarrow \emptyset$, $S \leftarrow V$
while $S \neq \emptyset$ do
    Pick a node $v_i$ of maximum weight in $S$
    Add $v_i$ to $V'$
    Remove $v_i$ and its neighbors from $S$
end while
Return $V'$
```

(b) (15 points) Please design a polynomial-time algorithm to find an independent set of maximum total weight. You will get at most 10 points if your algorithm runs in $O(n^2)$ and at most 15 points if your algorithm runs in $O(n)$. Please briefly justify the correctness and the running time of your algorithm.

Appendix

```
DIJKSTRA$(G, w, s)$
INITIALIZE-SINGLE-SOURCE$(G, s)$
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN$(Q)$
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX$(u, v, w)$
```

Figure 2: Dijkstra's algorithm

```
BELLMAN-FORD$(G, w, s)$
INITIALIZE-SINGLE-SOURCE$(G, s)$
for $i = 1$ to $|G.V| - 1$
    for each edge $(u, v) \in G.E$
        RELAX$(u, v, w)$
for each edge $(u, v) \in G.E$
    if $v.d > u.d + w(u, v)$
        return FALSE
return TRUE
```

Figure 3: The Bellman-Ford algorithm