CS161, Winter 2009

# CS161 Final Exam

(Do not turn this page until you are instructed to do so!)

**Instructions:** This is closed-book exam, and you are permitted to refer only to one double-sided sheet of notes, which you should have prepared in advance. You have 3 hours and the exam is worth 150 points. Make sure you print your name legibly and sign the honor code below. All of the intended answers can be written well within the space provided. You can use the back of the preceding page for scratch work. If you want to use the back side of a page to write part of your answer, be sure to mark your answer clearly. Good luck!

*The following is a statement of the Stanford University Honor Code:*

*A. The Honor Code is an undertaking of the students, individually and collectively:*

*(1) that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*

*(2) that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*

*B. The faculty on its part manifests its codence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*

*C. While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

I acknowledge and accept the Honor Code.

______________________________

(Signature)

______________________________

(Print your name, legibly!)

| Problem | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| Score | | | | | | | | | | |
| Maximum | 20 | 8 | 15 | 15 | 15 | 17 | 20 | 25 | 15 | 150 |

**Problem 1.** (20 points) [DIFFICULTY LEVEL: Relatively straightforward.] For each of the following 10 statements, circle "T" or "F", depending on whether you think the statement is true or false, respectively. You will receive 2 points for each correct answer *and -2 points for each incorrect answer*. (If you leave your answer blank, you will receive 0 points.) You need not justify your answers.

T F $n^{1/10} = O((\log n)^5)$

T F The worst-case running time of Randomized Quicksort on an array of $n$ elements is $O(n \log n)$.

T F A heap can be used to sort an array of $n$ elements with worst-case running time $O(n \log n)$.

T F A hash table can be used to sort an array of $n$ elements with expected running time $O(n)$.

T F Every algorithm that always correctly computes the median of an array of $n$ elements has worst-case running time $\Omega(n \log n)$.

T F Dijkstra's algorithm is always correct even in graphs with negative edge weights.

T F Prim's algorithm is always correct even in graphs with negative edge weights.

T F Using a suitable data structure, Dijkstra's algorithm can be implemented in $O(m \log n)$ time in graphs with $m$ edges and $n$ vertices.

T F Using a suitable data structure, Prim's algorithm can be implemented in $O(m \log n)$ time in graphs with $m$ edges and $n$ vertices.

T F Linearity of expectation (that the expectation of a sum of random variables equals the sum of the random variables' expectations) holds even for random variables that are not independent.

**Problem 2.** (8 points) [DIFFICULTY LEVEL: Relatively straightforward.]

(a) (4 points) Let $f$, $g$, and $h$ be positive real-valued functions defined on the positive integers. Assume that $f(n) = O(g(n))$ and that $g(n) = O(h(n))$. Prove, using the definition of big-oh notation, that $f(n) = O(h(n))$.

(b) (4 points) Determine whether the following statement is true or false: if $f$ and $g$ are two positive, real-valued functions defined on the positive integers and $f(n) = O(g(n))$, then $(f(n))^{10} = O((g(n))^{10})$. If you think this statement is true, provide a brief proof; if you think it is false, provide a counterexample.

**Problem 3.** **(15 points)** [DIFFICULTY LEVEL: Relatively straightforward.]

(a) **(7 points)** Explain, conceptually, why the Master Method has three different cases. Explain the conceptual meaning of each of the cases. Your answer should be 3-6 sentences.

(b) **(8 points)** Suppose that the worst-case running time $T(n)$ of an algorithm on an input of size $n$ is governed by the following recurrence, where $c > 0$ is some constant:

- $T(1) = 1$; and
- $T(n) \leq T(\lfloor n/2 \rfloor) + T(\lfloor n/3 \rfloor) + cn$ for every $n > 1$.

Prove, using the substitution method, that the algorithm runs in linear time (i.e., that $T(n) = O(n)$). Be sure to explicitly state the requirements that your constants need to satisfy.

**Problem 4.** (15 points) [DIFFICULTY LEVEL: Moderately difficul.] Consider the following two statements, one of which is true, and one of which is false:

– The first $k$ edges chosen by Kruskal's algorithm have the following property: there is no cheaper acyclic subgraph of $G$ with $k$ edges. (Assume edge costs are distinct.)

– The first $k$ edges chosen by Prim's algorithm (starting from some "root node" $r$) have the following property: there is no cheaper connected subgraph of $G$ containing the node $r$ along with $k$ other nodes. (Assume edge costs are distinct.)

Complete the following two questions.

(a) (7 points) Identify which of the above two statements is false, and provide an explicit counterexample.

(b) (8 points) Identify which of the above two statements is true, and provide a proof.

**Problem 5.** (15 points) [DIFFICULTY LEVEL: Moderately difficult.] Let $\mathcal{H}$ be a family of hash functions, mapping a large set $U$ of items to the $n$ buckets $\{0, 1, 2, \ldots, n-1\}$. We assume that $n$ is reasonably large (at least 10, say) and that $|U|$ is much bigger than $n$ (at least $n^2$, say).

Let $S$ be a set of $n$ distinct elements of $U$ and $x$ an element of $U \setminus S$. Suppose that a hash function $h$ is chosen uniformly at random from $\mathcal{H}$ and used to hash all of the elements of $S$.

Consider the following two statements, one of which is true, and one of which is false:

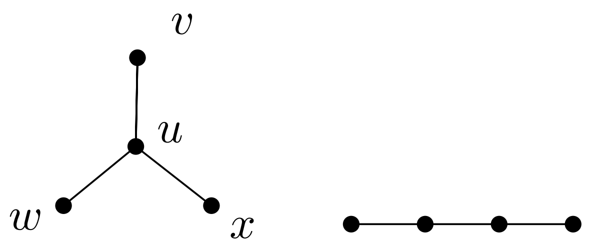
- If a hash function $h \in \mathcal{H}$, chosen uniformly at random, satisfies the property that $\Pr[h(y) = h(z)] \leq 1/n$ for all distinct items $y, z \in U$, then the expected number of elements $y$ of $S$ with $h(y) = h(x)$ is at most 1.
- If a hash function $h \in \mathcal{H}$, chosen uniformly at random, satisfies the property that $\Pr[h(y) = i] \leq 1/n$ for all items $y \in U$ and all buckets $i \in \{0, 1, \ldots, n-1\}$, then the expected number of elements $y$ of $S$ with $h(y) = h(x)$ is at most 1.

Complete the following two questions.

(a) (7 points) Identify which of the above two statements is false, and provide an explicit counterexample.

(b) (8 points) Identify which of the above two statements is true, and provide a proof.

**Problem 6.** (17 points) [DIFFICULTY LEVEL: Moderately difficul.] Let $G = (V, E)$ be an undirected graph. A subset $F \subseteq E$ of edges is a *matching* of $G$ if each vertex is the endpoint of at most one edge of $F$. (Thus each edge of a matching pairs two vertices together, with each vertex paired to at most one other one.) For example, if $G$ is a star on four vertices $u, v, w, x$ (Figure 1(a)), then the only matchings of $G$ are the subgraphs with at most one edge. If $G$ is a three-hop path (Figure 1(b)), then the first and third edges form a matching; as does the second edge by itself; as does any subset of one of these matchings.



(a) A Star on Four Vertices (b) A Three-Hop Path

Figure 1: Examples for Problem 6.

Now suppose that each edge $e$ of $G$ has a nonnegative *weight* $w_e$. We would like to compute the maximum-weight matching — i.e., the matching with the largest sum of edge weights — or at least a good approximation to it. Here is a natural greedy heuristic, in the spirit of Kruskal's algorithm:

```
• Sort the edges $e_1, \ldots, e_m$ of $G$ from largest to smallest weight.
• $M = \emptyset$
• For $i = 1$ to $m$:
    – if neither endpoint of edge $e_i = (v_i, w_i)$ is already matched —
      that is, $M$ currently contains no edges incident to either $v_i$ or
      $w_i$ — then add $e_i$ to $M$.
```

(a) **(3 points)** Show, by an explicit counterexample, that the algorithm above need not return a maximum-weight matching. For full credit, your counterexample should have the fewest-possible number of edges.

(b) **(14 points)** Prove that, for every graph and every set of non-negative edge weights, the algorithm above returns a matching whose total weight is at least 50% of that of a maximum-weight matching. (You can write your solution on the next page, if you like.)

**Problem 6 (con'd).** This page is extra space for your solution to Problem 6(b).

**Problem 7.** (20 points) [DIFFICULTY LEVEL: Moderately Difficult.] Recall that in the standard sequence alignment problem, you are given two strings $X$ and $Y$, of length $n$ and $m$, respectively, over a common alphabet $\Sigma$. The input also includes the penalty $\alpha_{gap}$ of inserting a gap and the penalty $\alpha_{ab}$ of (mis)matching each pair of characters $a, b$.

It often makes sense that the penalty for a gap of length 10, say, should be strictly smaller than 10 times the penalty of a single gap. (This reflects the possibility that a large chunk of a genome may get added or deleted via a single "mutation".)

To model this, consider a generalization of the sequence alignment problem in which, instead of a single gap penalty $\alpha_{gap}$, the input includes parameters $\alpha_0, \alpha_1$, and where the penalty of inserting exactly $k \geq 1$ gaps in a row is $\alpha_0 + \alpha_1 k$. ($\alpha_0$ is larger than $\alpha_1$.) For example, the string AGT can be matched with ACCCGCCT with total penalty $(\alpha_0 + \alpha_1 \cdot 3) + (\alpha_0 + \alpha_1 \cdot 2) = 2\alpha_0 + 5\alpha_1$.

Design a dynamic programming algorithm that correctly solves this generalized sequence alignment problem and runs in $O(mn)$ time. Your algorithm should compute the *value* (i.e., minimum-possible total penalty) of an optimal alignment; you do *not* need to produce the alignment itself.

In your answer, explain clearly what the subproblems are, the recurrence(s) you use to relate the solutions of different subproblems to one another, and the order in which you solve the subproblems.

You should give a concise outline of a correctness proof for your recurrence(s), but don't waste too much time doing it (4-5 sentences maximum). Also include a brief analysis of the running time of your algorithm.

**Problem 8.** (25 points) [DIFFICULTY LEVEL: Difficult.] NOTE THIS PROBLEM HAS 6 PARTS (continued on the next page).

Recall the all-pairs shortest-path problem: given a directed graph $G = (V, E)$ with lengths $c_e$ on the edges, compute the shortest-path length $l(v, w)$ for every ordered pair $(v, w) \in V \times V$.

(a) (3 points) Argue that if all of the edge lengths are nonnegative, then the all-pairs shortest-path problem can be solved in $O(nm \log n)$ time (where $n = |V|$ and $m = |E|$).

[If you use an algorithm discussed in class as a subroutine, you can assume any running time bounds that we proved about it.]

(b) (6 points) Assume that the input graph has no negative cycles (though it may have negative edges). Let $s$ be an arbitrary vertex, and let $d(v)$ denote the length of a shortest path (with no cycles) from $s$ to $v$. Note that $d(v)$ may be positive or negative. Prove the "Triangle Inequality": for every directed edge $(v, w) \in E$,

$$d(w) \leq d(v) + c_{vw}.$$

(c) **(3 points)** We continue with the assumptions and notation of part (b). For each edge $(v, w) \in E$, define the *artificial length* $a_{vw}$ of $(v, w)$ as $a_{vw} = c_{vw} + d(v) - d(w)$. Prove that if $P$ is a path from a vertex $x$ to a vertex $y$, then its total length with respect to the artificial edge lengths (i.e., $\sum_{(v,w)\in P} a_{vw}$) equals its total length with respect to the original edge lengths plus $d(x) - d(y)$.

(d) **(3 points)** In the language of part (c), prove that if $a(x, y)$ is the shortest artificial length of a path from $x$ to $y$, then $l(x, y) = a(x, y) - d(x) + d(y)$.

(e) **(7 points)** Argue that the all-pairs shortest-path problem can be solved in $O(nm \log n)$ time for graphs with arbitrary (positive or negative) edge lengths and no negative cycles.

[Assume for this part that you have already solved the four previous parts (even if you haven't).]

(f) **(3 points)** For which graphs (i.e., which values of $n$ and $m$) is the algorithm in (e) asymptotically faster than the Floyd-Warshall algorithm?

**Problem 9.** (15 points) [DIFFICULTY LEVEL: Very Difficult.] In the 2SAT problem, you are given a set of *clauses*, where each clause is the disjunction of two literals (a literal is a Boolean variable or the negation of a Boolean variable). You are looking for a way to assign a value `true` or `false` to each of the variables so that *all* clauses are satisfied — that is, there is at least one true literal in each clause. For example, here's an instance of 2SAT:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_4).$$

This instance has a satisfying assignment: set $x_1$, $x_2$, $x_3$, and $x_4$ to `true`, `false`, `false`, and `true`, respectively. Changing the value of $x_2$ from `false` to `true` yields another satisfying assignment.

For this problem, design an algorithm that determines whether or not a given 2SAT instance has a satisfying assignment. (Your algorithm does *not* need to exhibit a satisfying assignment, just decide whether or not one exists.) Your algorithm should run in $O(m + n)$ time, where $m$ and $n$ are the number of clauses and variables, respectively. Include a proof of correctness (no more than two paragraphs) and a concise running time analysis.

[Hint: you might try reducing 2SAT to the problem of computing strongly connected components, with each variable giving rise to two vertices and each clause giving rise to two directed edges.]