

Quiz 2

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 120 minutes to earn 120 points. Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- This quiz booklet contains 13 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your quiz at the end of the exam period.
- This quiz is closed book. You may use two $8\frac{1}{2}'' \times 11''$ or A4 crib sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader
1	2	10		
2	10	30		
3	3	20		
4	2	20		
5	3	20		
6	3	20		
Total		120		

Name: _____

Recitation: Christina Christina Jayant Jayant Jason Matthew
 10 AM 11 AM 12 PM 1 PM 2 PM 3 PM

Problem 1. BFS/DFS [10 points] (2 parts)

Give the visited node order for each type of graph search, starting with s , given the following adjacency lists and accompanying figure:

$$\text{adj}(s) = [a, c, d],$$

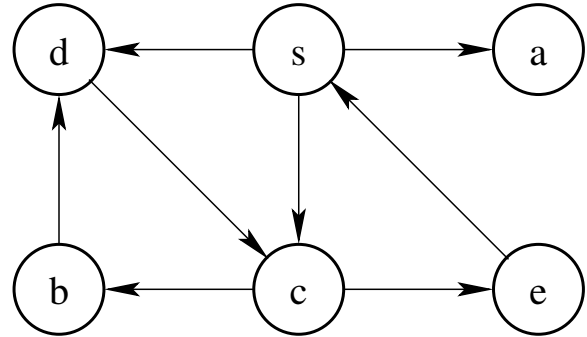
$$\text{adj}(a) = [],$$

$$\text{adj}(c) = [e, b],$$

$$\text{adj}(b) = [d],$$

$$\text{adj}(d) = [c],$$

$$\text{adj}(e) = [s].$$



(a) Breadth First Search

(b) Depth First Search

Problem 2. Miscellaneous True/False [30 points] (10 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (No credit if no explanation given.)

- (a) **T F** While running DFS on a directed graph, if from vertex u we visit a finished vertex v , then the edge (u, v) is a cross-edge.

Explain:

- (b) **T F** Changing the RELAX function to update if $d[v] \geq d[u] + w(u, v)$ (instead of strictly greater than) may produce different shortest paths, but will not affect the correctness of the Bellman-Ford outputs d and π .

Explain:

- (c) **T F** The running time of Radix sort is effectively independent of whether the input is already sorted.

Explain:

- (d) **T F** Let P be a shortest path from some vertex s to some other vertex t in a directed graph. If the weight of each edge in the graph is increased by one, P will still be a shortest path from s to t .

Explain:

- (e) **T F** If a weighted directed graph G is known to have no shortest paths longer than k edges, then it suffices to run Bellman-Ford for only k passes in order to solve the single-source shortest paths problem on G .

(f) **T F** If a topological sort exists for the vertices in a directed graph, then a DFS on the graph will produce no back edges.

Explain:

(g) **T F** Dynamic programming is more closely related to BFS than it is to DFS.

Explain:

(h) **T F** A depth-first search of a directed graph always produces the same number of tree edges (i.e. independent of the order in which the vertices are provided and independent of the order of the adjacency lists).

Explain:

- (i) **T F** Suppose we do a DFS on a directed graph G . If we remove all of the back edges found, the resulting graph is now acyclic.

Explain:

- (j) **T F** Both DFS and BFS require $\Omega(V)$ storage for their operation. (That is, for working storage, above and beyond the storage needed to represent the input.)

Explain:

Problem 3. Miscellaneous Short Answer [20 points] (3 parts)

- (a) Suppose you want to get from s to t on weighted graph G with nonnegative edge weights, but you would like to stop by u if it isn't too inconvenient. (Here too inconvenient means that it increases the length of your travel by more than 10%.)

Describe an efficient algorithm that would determine an optimal s to t path given your preference for stopping at u along the way if not too inconvenient. It should either return the shortest path from s to t , or the shortest path from s to t containing u .

- (b) Explain how the “rod-cutting” problem described in class can still be solved by dynamic programming, even if cuts now cost \$1 each. (In class, we assumed cuts were free.) Here is the pseudocode for the original solution, where the cuts were free:

```
r = [0] * (n + 1)
for k in range(1, n + 1):
    ans = p[k]
    for i range(1, k):
        ans = max(ans, p[i] + r[k - i])
    r[k] = ans
```

(It suffices to explain how to express r_n , the maximum revenue achievable for a rod of size n , in terms of r_1, r_2, \dots, r_{n-1} and the prices p_i that the market will pay for a piece of length i , for $i = 1, 2, \dots, n$.)

- (c) Suppose that you implement Dijkstra's algorithm using a priority queue algorithm that requires $O(V)$ time to initialize, worst-case $f(V, E)$ time for each EXTRACT-MIN operation and worst-case $g(V, E)$ time for each DECREASE-KEY operation. How much (worst-case) time does it take to run Dijkstra's algorithm on an input graph $G = (V, E)$?

Problem 4. A Series of Tubes [20 points] (2 parts)

Consider a network of computers represented by a directed graph G . Each vertex $v \in V$ represents a computer, and each edge $(u, v) \in E$ represents a network link from u to v .

- (a) Let each edge (u, v) in G have a weight $w(u, v) \in (0, 1]$, representing the probability that a packet going from u to v is successfully delivered. Give an efficient algorithm to find the path along which a packet has the *highest* probability of reaching its destination (i.e., the path for which the product of the edge weights is *maximized*).

Hint: Transform the weights and use a shortest path algorithm.

- (b) Now, instead of weighted edges, consider weighted vertices. In other words, network links never drop packets, but nodes might. Edges are not weighted, but each vertex v has a weight $w(v) \in (0, 1]$, representing the probability that an incoming packet is *not* dropped by that node. Give an efficient algorithm to find the path along which a packet has the highest probability of reaching its destination (i.e., the path for which the product of the vertex weights is maximized).

Hint: Transform the graph and use the algorithm from part (a).

Problem 5. Fast Flyer [20 points] (3 parts)

You are given a list of all scheduled daily flights in the US, giving departure airports, departure times, destination airports, and arrival times. We want an algorithm to compute travel times between airports, including waiting times between connections. Assume that if one flight arrives at time t and another departs at time $t' \geq t$, travelers can make the connection. Further assume that at a given airport, no two events (arrivals or departures) occur at the same time, and that there are at most 100 events at any airport during a given day. All times are given in GMT; don't worry about time zones.

Construct a weighted graph so that given a departure airport, a departure time T , and a destination airport, we can efficiently determine the earliest time T' that a traveler can be guaranteed (according to the schedules!) of arriving at her destination *on that day* (ignore overnight flights and overnight airport stays).

- (a) What do vertices represent? What do edges in your graph represent, and what is the weight of an edge?
- (b) Give an upper bound on the number of edges and vertices in your graph if there are n airports in the US and m daily flights. Justify your bound.

- (c) What algorithm would you use to compute the shortest travel times, and what is its running time in terms of the number of vertices, V , and the number of edges, E ?

Problem 6. Articulation Points [20 points] (3 parts)

We define an articulation point as a vertex that when removed causes a connected graph to become disconnected. For this problem, we will try to find the articulation points in an *undirected* graph G .

- (a) How can we efficiently check whether or not a graph is disconnected? (Hint: think of a recent problem set question)

- (b) Describe an algorithm that uses a brute force approach to find all the articulation points in G in $O(V(V + E))$ time.

An observer comments that the polynomial time algorithm is rather slow and suggests to use a linear time DFS approach instead. Let's explore this idea some more.

- (c) Suppose we run DFS on graph G . Consider the types of edges that can exist in a DFS tree produced from an *undirected* graph, recalling that cross edges can't happen in the DFS of an undirected graph. Argue that a non-root, non-leaf vertex u is an articulation point *if and only if* there exists a subtree rooted at a child of u that has no back edges to a proper ancestor of u .

(FYI: The above idea can be extended to yield an $O(V + E)$ algorithm to find all articulation points, but we don't have time for all of that on this quiz!)

SCRATCH PAPER

SCRATCH PAPER