Quiz 2 Solutions

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 6 problems, with multiple parts. You have 120 minutes to earn 120 points.
- This quiz booklet contains 11 pages, including this one.
- This quiz is closed book. You may use two double-sided letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheets. No calculators or programmable devices are permitted. Cell phones must be put away.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- 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. Please be neat.
- Good luck!

Problem	Title	Points	Parts	Grade	Initials
1	True or False	40	10		
2	Who Charged the Electric Car?	20	3		
3	Planning Ahead	10	1		
4	Maze Marathoner	20	3		
5	6.046 Carpool	10	1		
6	Paths and/or Cycles	20	2		
Total		120			

Name: _____

Problem 1. True or False. [40 points] (10 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false **and briefly explain why**.

(a) T F [4 points]

Using similar techniques used in Strassen's matrix multiplication algorithm, the Floyd–Warshall algorithm's running time can be improved to $O(V^{\log_2 7})$.

Solution: False. There is no way to define negation.

(b) T F [4 points]

For graphs $G = (V, E)$ where $E = O(V^{1.5})$, Johnson's algorithm is asymptotically faster than Floyd–Warshall.

Solution: True. $O(VE + V^2 \log V) = o(V^3)$ when $E = o(V^2)$.

(c) T F [4 points]

Consider the directed graph where each vertex represents a subproblem in a dynamic program, and there is an edge from p to q if and only if subproblem p depends on (recursively calls) subproblem q. Then this graph is a directed rooted tree.

Solution: False. It is a Directed Acyclic Graphic (DAG).

(d) T F [4 points]

In a connected, weighted graph, every lowest weight edge is always in *some* minimum spanning tree.

Solution: True. It can be the first edge added by Kruskal's algorithm.

(e) T F [4 points]

For a connected, weighted graph with n vertices and exactly n edges, it is possible to find a minimum spanning tree in $O(n)$ time.

Solution: True. This graph only contains one cycle, which can be found by a DFS. Just remove the heaviest edge in that cycle.

(f) T F [4 points]

For a flow network with an integer capacity on every edge, the Ford–Fulkerson algorithm runs in time $O((V + E)\,|f|)$ where $|f|$ is the maximum flow.

Solution: True. There can be $O(|f|)$ iterations because each iteration increases the flow by at least 1.

(g) T F [4 points]

Let $C = (S, V \setminus S)$ be a minimum cut in a flow network. If we strictly increase the capacity of every edge across C, then the maximum flow of the network must increase.

Solution: False. There could be another min cut whose capacity does not change. Then the max flow remains the same.

(h) T F [4 points]

Every linear program has a unique optimal solution.

Solution: False. There can be many optimal solutions if the objective function is parallel to one of the constrains.

Alternative solution: False. There could be no solutions at all.

Alternative solution: False. There could be no solutions at all.

(i) T F [4 points]

3SAT cannot be solved in polynomial time, even if $\mathrm{P} = \mathrm{NP}$.

Solution: False. If P = NP, then all problems in P are also NP-hard, and these problems have polynomial-time algorithms.

(j) T F [4 points]

Repeatedly selecting a vertex of maximum degree, and deleting the incident edges, is a 2-approximation algorithm for Vertex Cover.

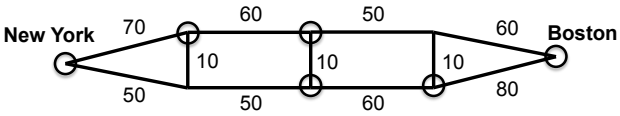
Solution: False: it can be as bad as a log-log approximation, see L17 notes.

Problem 2. Who Charged the Electric Car? [20 points] (3 parts)

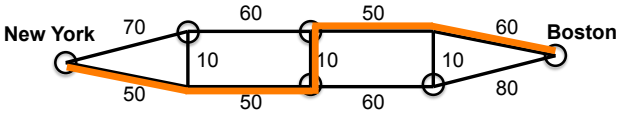
Prof. Musk is driving his Nikola electric car from Boston to New York. He wants to take the shortest path, but his car can only drive m miles before needing to charge. Fortunately, there are Furiouscharger charging stations on the way from Boston to New York, which instantaneously charge the battery to full.

The road network is given to you as a weighted undirected graph $G = (V, E, w)$ along with the subset $C \subseteq V$ of vertices that have charging stations. Each weight $w(e)$ denotes the (positive) length of road e. The goal is to find a shortest path from node $s \in V$ to node $t \in V$ that does not travel more than m miles between charging stations. Assume that $s, t \in C$.

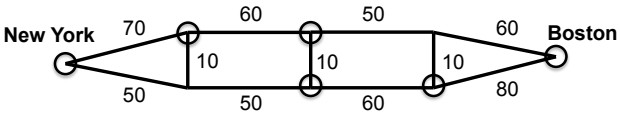
(a) [4 points] Draw the shortest path from Boston to New York in the following graph if $m = \infty$. Charging stations are marked as circles.



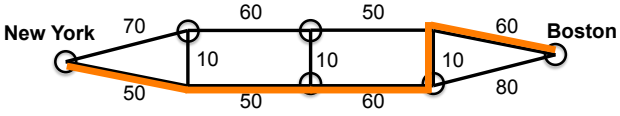
Solution:



(b) [4 points] Draw the shortest path from Boston to New York in the following (identical) graph if $m = 100$.



Solution:



(c) [12 points] Give an algorithm to solve the problem. For full credit, your algorithm should run in $O(VE + V^2 \log V)$ time.

Solution: Our algorithm consists of two steps – the first step involves running Johnson's algorithm on the original graph G to obtain shortest path lengths for every pair

of vertices. Let $\delta(u, v)$ represent the length of the shortest path between vertices u and v in G.

For the second step, we build a graph G' with vertex set C. For every pair of vertices u and v in the new graph G', draw an edge between u and v with weight $\delta(u, v)$ if $\delta(u, v) \leq m$ and ∞ otherwise.

Now, run Dijkstra's algorithm on G' between Boston and New York to get the shortest path. (Note that New York and Boston have charging stations and so are vertices in the graph G').

Running Johnson's algorithm on the original graph G takes $O(VE + V^2 \log V)$. Creating the graph G' takes $O(E)$ time, and running Dijkstra's algorithm on G' takes $O(V^2 + V \log V)$ time; this gives a total runtime complexity of $O(VE + V^2 \log V)$.

Problem 3. Planning Ahead [10 points] (1 part)

You have N psets due right now, but you haven't started any of them, so they are all going to be late. Each pset requires d_i days to complete, and has a cost penalty of c_i *per day*. So if pset i ends up being finished t days late, then it incurs a penalty of $t \cdot c_i$. Assume that once you start working on a pset, you must work on it until you finish it, and that you cannot work on multiple psets at the same time.

For example, suppose you have three problem sets: 6.003 takes 3 days and has a penalty of 12 points/day, 6.046 takes 4 days and has a penalty of 20 points/day, and 6.006 takes 2 days and has a peanlty of 4 points/day. The best order is then 6.046, 6.003, 6.006 which results in a penalty of $20 \cdot 4 + 12 \cdot (4 + 3) + 4 \cdot (3 + 4 + 2) = 200$ points.

Give a greedy algorithm that outputs an ordering of the psets that minimizes the total penalty for all the psets. Analyze the running time and prove correctness.

Solution: Sort by increasing d_i/c_i and do the problem sets in that order. This takes $O(N \log N)$ time.

Proof – If unsorted, we can improve by swapping.

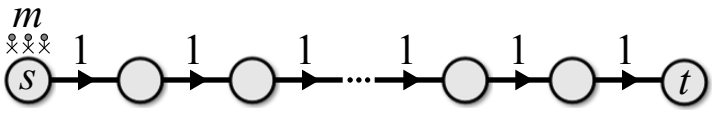
$$d_i/c_i > d_j/c_j \implies c_j d_i + (c_i d_i + c_j d_j) > c_i d_j + (c_i d_i + c_j d_j) \quad (1)$$

$$\implies c_j(d_i + d_j) + c_i d_i > c_i(d_i + d_j) + c_j d_j \quad (2)$$

Problem 4. Maze Marathoner [20 points] (3 parts)

A group of m teens need to escape a maze, represented by a directed graph $G = (V, E)$. The teens all start at a common vertex $s \in V$, and all need to get to the single exit at $t \in V$. Every night, each teen can choose to remain where they are, or traverse an edge to a neighboring vertex (which takes exactly one night to traverse). However, each edge $e \in E$ has an associated capacity $c(e)$, meaning that at most $c(e)$ teens can traverse the edge during the same night. The goal is to minimize the number of nights required for all teens to escape by reaching the goal t.

(a) [3 points] First look at the special case where the maze is just a single path of length $|E|$ from s to t, and all the edges have capacity 1 (see below). Exactly how many nights are required for the teens to escape?



Solution: $|E| + m - 1$ or $|V| + m - 2$. Em or Vm will get partial credits.

(b) [7 points] The general case is more complex. Assume for now that we have a "magic" algorithm that calculates whether the teens can all escape using $\leq k$ nights. The magic algorithm runs in polynomial time: $k^\alpha\, T(V, E, m)$ where $\alpha = O(1)$.

Give an algorithm to calculate the minimum number of nights to escape, by making calls to the magic algorithm. Analyze your time complexity in terms of V, E, m, α, and $T(V, E, m)$.

Solution: Do a binary search. A sequential scan will get partial credits.

The maximum number of nights can be bounded by $O(E+m)$ (or $O(V+m)$, $O(Em)$) according to part(a). Therefore, we need to run the "magic" algorithm $O(\log(E+m))$ times. Each run takes no more than $O((E + m)^\alpha T(V, E, m))$ time. So in total, the runtime is $O((E + m)^\alpha \log(E + m)T(V, E, m))$.

Common mistake 1: runtime $O(k^\alpha \log(E + m)T(V, E, m))$. k should not appear in the runtime. You need to find the bound on k.

Common mistake 2: using sequential scan runtime $O((E + m)^\alpha T(V, E, m))$. The summation is calculated incorrectly, should be $\sum_{i=1}^{n} n^\alpha = O(n^{\alpha+1})$

(c) [10 points] Now give the "magic" algorithm, and analyze its time complexity.

Hint: Transform the problem into a max-flow problem by constructing a graph $G' = (V', E')$ where $V' = \{(v, i) \mid v \in V, 0 \leq i \leq k\}$. What should E' be?

Solution: Model this as a max flow problem. Construct a graph $G' = (V', E')$ where $V' = \{(v, i) \mid v \in V, 0 \leq i \leq k\}$. For all $0 \leq i \leq k-1$, connect (v, i) to $(v, i+1)$ with capacity ∞ (or m); this represents teens can stay at a vertex for the night. For every edge $\langle u, v\rangle$ in the original graph, connect (u, i) to $(v, i + 1)$ with capacity $c(\langle u, v\rangle)$; this represents $c(\langle u, v\rangle)$ teens can travel from u to v in a night.

The new source s' is the vertex $(s, 0)$ and the new sink t' is the vertex $(t, k - 1)$. If the max flow from s' to t' is no less than m, then people can escape within k nights.

Runtime: Both of the following are accepted.

There are $V = O(kV')$ vertices and $E' = O(k(V + E))$ edges in G'. Applying Edmonds-Karp algorithm, the total time complexity is $O(VE'^2) = O(k^3V(V + E)^2)$.

If using Ford-Fulkerson runtime, notice that we can actually stop if the max flow reaches m. So at most m iterations are needed. Runtime can be $O(m(V' + E')) = O(mk(V + E))$.

Common mistake 1: connect (v, i) to (u, i) instead of (v, i) to $(u, i + 1)$.

Common mistake 2: no edge from (v, i) to $(v, i + 1)$.

Both solutions above are equivalent of running max flow on the original graph, because there could be very long path ($> k$) with large capacity ($> m$) in the original graph. In that case, max flow is larger than m, but teens cannot escape in k nights.

Problem 5. 6.046 Carpool [10 points] (1 part)

The n people in your dorm want to carpool to 34-101 during the m days of 6.046. On day i, some subset S_i of people actually want to carpool (i.e., attend lecture), and the driver d_i must be selected from S_i. Each person j has a limited number of days ℓ_j they are willing to drive.

Give an algorithm to find a driver assignment $d_i \in S_i$ for each day i such that no person j has to drive more than their limit ℓ_j. (The algorithm should output "no" if there is no such assignment.) *Hint:* Use network flow.

For example, for the following input with $n = 3$ and $m = 3$, the algorithm could assign Penny to Day 1 and Day 2, and Leonard to Day 3.

Person	Day 1	Day 2	Day 3	Driving limit
1 (Penny)	X	X	X	2
2 (Leonard)	X		X	1
3 (Sheldon)		X	X	0

Solution: First, we create a graph with following vertices:

1.a super source s and a super sink t

2.vertex p_i for each person who wants to carpool

3.vertex d_j for each day of the class.

Then create the following edges:

1.s to p_i with capacity of ℓ_j

2.p_i to d_j with capacity of 1 if person i needs to carpool on day j

3.d_j to t with weight 1 for all j.

Finally, run max flow from s to t, and find f. If $|f| = m$, return that person i will drive on day j if the edge (p_i, d_j) has non-zero flow. If $|f| < m$, then return no valid assignment.

At a high level, the graph represents a matching between the driver and the days he/she will be driving. The capacity from s to p_i will ensure that no p_i drives more than l_i days (flow conservation). The non-zero flows from p_i to d_j means that p_i will drive on day d_j. The capacity of d_j to t will ensure that no more than 1 driver will be assigned to a particular day (flow conservation). If $|f| = m$, then all vertices d_j has an incoming flow, and therefore all d_j has a valid driver assignment. If $|f| < m$, then there is at least one d_j that has no incoming flow, and therefore does not have a driver assigned to it. By maximality of the flow, this means that there does not exist a valid driver assignment.

Ford-Fulkerson algorithm would run in $O(nm^2)$ since there are at most $nm + 2$ edges (bipartite graph), and the max flow is bounded by m. Running Edmonds-Karp would run in $O((n + m)(nm)^2) = O(n^3m^2 + n^2m^3)$, which is slower in this case.

Problem 6. Paths and/or Cycles [20 points] (2 parts)

A ***Hamiltonian path*** on a directed graph $G = (V, E)$ is a path that visits each vertex in V exactly once. Consider the following variants on Hamiltonian path:

(a) [10 points] Give a polynomial-time algorithm to determine whether a directed graph G contains *either* a cycle or a Hamiltonian path (or both).

Solution: To solve the problem, we simply run DFS on G. If a cycle exists, DFS will traverse a vertex twice and can report the cycle. If no cycle exists, then the graph is a DAG.

If the graph is a DAG, then we can run a topological sort on the graph. If there is a complete, or unique, ordering of every vertex in the graph, the graph has a Hamiltonian Path, and we accept the graph.

(b) [10 points] Show that it is NP-hard to decide whether a directed graph G' contains *both* a cycle and a Hamiltonian Path, by giving a reduction from the HAMILTONIAN PATH problem: given a graph G, decide whether it has a Hamiltonian path. (Recall from recitation that the HAMILTONIAN PATH problem is NP-complete.)

Solution: We construct a graph $G' = (V', E')$ from G, where

$$V' = \{u_1, u_2, u_3\} \cup V$$

$$E' = \{(u_1, u_2), (u_2, u_3), (u_3, u_1)\} \cup \{(u_1, V) : v \in V\} \cup E$$

G' always has a cycle of length 3 - (u_1, u_2, u_3). For any Hamiltonian Path P in G, (u_2, u_3, u_1, P) is a Hamiltonian Path in G'. For any Hamiltonian Path P' in G', P' must be of the form (u_2, u_3, u_1, P), where P is a Hamiltonian path for G. Thus in all cases, solving B(G') is equivalent to solving Hamiltonian Path for G.

An alternate solution used the algorithm for cycle and Hamiltonian Path as an oracle. An input graph to the Hamiltonian Path is used as an input to the cycle and Hamiltonian Path algorithm. If the cycle and Hamiltonian Path algorithm accepts, then the original graph is also accepted. However, if the cycle and Hamiltonian path algorithm rejects the input, we use DFS to determine if the graph contains a cycle and the cycle or Hamiltonian Path algorithm from part a. If DFS cannot find a cycle but the cycle or Hamiltonian path algorithm accepts, then we accept the graph, and reject the graph otherwise.

SCRATCH PAPER

MIT OpenCourseWare
http://ocw.mit.edu

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.