CS570 Fall 2021: Analysis of Algorithms Exam III

	Points		Points
Problem 1	18	Problem 5	14
Problem 2	6	Problem 6	18
Problem 3	10	Problem 7	16
Problem 4	18		
	Total	**100**	

Instructions:

1. This is a 2-hr exam. Open book and notes but no internet access allowed.
2. If a description to an algorithm or a proof is required, please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
3. No space other than the pages in the exam booklet will be scanned for grading.
4. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
5. Do not detach any sheets from the booklet. Detached sheets will not be scanned.
6. If using a pencil to write the answers, make sure you apply enough pressure, so your answers are readable in the scanned copy of your exam.
7. Do not write your answers in cursive scripts.

8. This exam is printed double sided. Check and use the back of each page.

1) Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification. (18 pts total)

[TRUE/FALSE] (2 pts)
The NP-Hard class of problems does not contain any decision problems.

[TRUE/FALSE] (2 pts)
If every edge in a Flow Network is saturated due to flow F, then F must be a max flow.

[TRUE/FALSE] (2 pts)
If every edge in a Flow Network G is saturated due to flow F, then G has a unique max flow.

[TRUE/FALSE] (2 pts)
If there exists an algorithm that solves problem X with pseudo-polynomial runtime, then X must be NP-Hard.

[TRUE/FALSE] (2 pts)
Based on the definition of polynomial time reduction, we can say that linear programming is polynomial time reducible to max flow

[TRUE/FALSE] (2 pts)
Suppose there is a 7-approxmiation algorithm for the general Traveling Salesman Problem. Then there exists a polynomial time solution for the 3-SAT problem.

[TRUE/FALSE] (2 pts)
We can use the non-scaled version of the Ford-Fulkerson algorithm to find max-flow in weakly polynomial time, and use the scaled version of the algorithm to find max-flow in strongly polynomial time.

[TRUE/FALSE] (2 pts)
Suppose $T(n) = T(n/2) + \log n$, then $T(n) = O(\log n)$.

[TRUE/FALSE] (2 pts)
If X <=p Y, then an α-approximation algorithm for Y will give us an α-approximation algorithm for X

2) Multiple choice (6 pts total)

2.1 (3 pts)
Consider 3 problems A, B, and C where we know problem A is NP-complete, $A \leq_p B$, and $C \leq_p A$. Based on this information select all correct answers below.

- B is NP-Complete
- C is in NP
- $C \leq_p B$
- B is NP-Hard

2.2 (3 pts)
Consider the following linear program:

maximize $x_1 + x_2$
subject to $x_1 + 2x_2 \leq 2$
$2x_1 + x_2 \leq 2$
$x1, x2 \geq 0$

What is the (optimal) solution of this linear program?

(a) $(x_1, x_2) = (1/3, 1/3)$
(b) $(x_1, x_2) = (2/3, 2/3)$
(c) $(x_1, x_2) = (1/3, 2/3)$
(d) None of the above

3) (10 pts)

Recall the Min-Flow problem from discussion set 9:

Given a directed graph G=(V,E) a source node s ϵ V, a sink node t ϵ V, and lower bound l_e for flow on each edge e ϵ E, find a feasible s-t flow of minimum possible value.

Note: there are no capacity limits for flow on edges in G.

Provide a linear programming formulation to solve this problem.
Describe your:

a) LP variables (2 pts)

Variables will be f(e)'s i.e., flow over the edge e for all edges e ∈ E

b) Objective function (2 pts)

Minimize ∑ f(e) for all e out of s (or into t)

-1 if it says max instead of min
-1 if doesn't sum over only the edges out of s (or only the edges into t)

c) Constraints (6 pts)

3 points each for lower bound and conservation of flow

Lower bound constraints:

f(e) >= le for all edges e ∈ E

conservation of flow

∑ f(e) for all e into v = ∑ f(e) for all e out of v, for all v ∈ V except for s and t

-1 point if conservation constraint doesn't exclude s,t.

Alternate solution (more complicated): penalties apply same as for the solution before

f denotes feasible flow satisfying lower bounds, f' denotes flow in the reverse direction (to decrease from feasible flow f)

d) LP variables (2 pts)

Variables will be f(e), f'(e) over edge e for all edges e ∈ E

e) Objective function (2 pts)

Minimize ∑ (f(e) - f'(e)) for all e out of s (or into t)

f) Constraints (6 pts)

1.5 points each for lower bound and conservation of flow

Lower bound feasibility of f:

f(e) >= le for all edges e ∈ E

capacity for reverse flow f':

f'(e) <= f(e) - le for all edges e ∈ E

conservation of flow for f as well as f':

∑ f(e) for all e into v = ∑ f(e) for all e out of v, for all v ∈ V except for s and t

∑ f(e) for all e into v = ∑ f(e) for all e out of v, for all v ∈ V except for s and t

4) (18 pts)
After a snowstorm, all the traffic lights went out on Main Street. Main Street is a linear street with n traffic lights. Because traffic lights are interconnected, fixing one traffic light will automatically fix its adjacent traffic lights (traffic lights adjacent to light i are lights i-1 and i+1 if they exist). However, fixing an already fixed light will do nothing. You are given a list of the times needed (in hours) to fix each light: $T = t_1, t_2, t_3, \ldots, t_n$. You are also given a list of pays for each light that you work on: $P = p_1, p_2, p_3, \ldots, p_n$. Note that you are only given pay for the lights that you have worked on, not the adjacent lights that are fixed automatically. You are given k hours to work on the traffic lights. Find the maximum pay possible.

An example to this question:
Total 9 lights: $L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8, L_9$
Time needed to fix: T = 1, 4, 1, 4, 1, 1, 5, 2, 2
Pays: P = 250, 130, 100, 60, 50, 70, 270, 160, 140
You are given k = 6 hours
Solution: You should fix L_1, L_3, L_6, L_8 to earn a maximum pay of 580.

a) Define (in plain English) subproblems to be solved. (4 pts)

OPT[i][j] = Max pay value for lights 1..i given a total of j hours to work on traffic lights

- alternate solution can consider lights i to n (recurrence and pseudo-code changes accordingly)

-1 for saying "i lights" and not specifying which i
-2 for saying "light i" rather than the set of i lights above

b) Write a recurrence relation for the subproblems (5 pts)

Very similar to the 0-1 knapsack problem:

if we can fix light i within the j hours left, i.e. $t_i \leq j$
OPT[i][j] = max(OPT[i - 1][j], OPT[i - 2][j - t_i] + p_i)
else:
OPT[i][j] = OPT[i - 1][j] (Okay to exclude this case here IF correctly covered in the pseudo-code via base cases or otherwise)

Scoring: up to -2 per mistake depending on how severe.

c) Using the recurrence formula in part b, write pseudocode using iteration to compute the maximum possible pay. (4 pts)
Make sure you specify
- base cases and their values (2 pts)
- where the final answer can be found (e.g. opt(n), or opt(0,n), etc.) (1 pt)

- OPT[0][j] = 0 for all j= 0 ... k (can skip this by adding base cases for OPT[2][j] instead)
- OPT[i][0] = 0 for all i= 0 ... n (this can be excluded as it's handled by the $t_i > j$ case in the loop)

- OPT[1][j] = 0 if $t_1 > j$ otherwise p_1 for all j= 1 ... k

```
For i from 2 to n:
    For j from 1 to k:

        if t_i <= remaining time j:
            OPT[i][j] = max(OPT[i - 1][j], OPT[i - 2][j - t_i] + p_i)
        else:
            OPT[i][j] = OPT[i - 1][j]
```

Total_pay = OPT[n][k]

Scoring: up to -2 per mistake depending on how severe. No repeated penalties from the recurrence (but other errors arising from the inaccurate recurrence cannot be accepted.)

c) What is the complexity of your solution? (1 pt)

Runs in O(nk)

Is this an efficient solution? (1 pt)

No. This is not an efficient algorithm. This is a pseudo-polynomial time solution because the polynomial function nk contains a numerical value of an input term k

5) (14 pts)
Given an undirected, connected graph G with all edges having integer weights in the range 1 to 10, and starting vertex s, design an algorithm to find the smallest distance from s to all other nodes in the graph. The designed algorithm must run asymptotically faster than Dijkstra's and state why.

Given graph G we will construct graph G' as follows:
G' will have the same initial set of nodes as G
For each edge e=(v,u) with cost c (1<c<10) in G we will create a sequence of c edges of cost 1 each, to form a path of cost c to go from v to u.

All edges in G' will have a cost of 1 unit. G' will have no more than 10m edges (m=number of edges in G)

We can run BFS to find the shortest path from s to t at a cost of O(10m+n) = O(m+n) which is asymptotically faster than Dijkstra's algorithm.

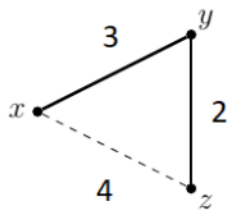
Rubics:
The only completely correct and efficient solution with linear time is the one above and Dial's Algorithm which will receive full points.

If use the solution above:
Correctly construct graph G', 8 points, if don't mention cost 1 for each unit -3 points
run BFS 3 points
time complexity analysis 3 points

Any attempt that leverages the property that **there's only a small range of integer weights** and do modification on the algorithm will receive 7 points.

Only running Dijkstra or whatever for the shortest path with nothing special will receive 4 points.

Any methods that fail to find the shortest path will receive 0 points (include MST, LP, max flow, iteratively adding edge in the increasing order of weight, directly running BFS on the weighted graph) Counterexample for MST and some greedy algorithm:

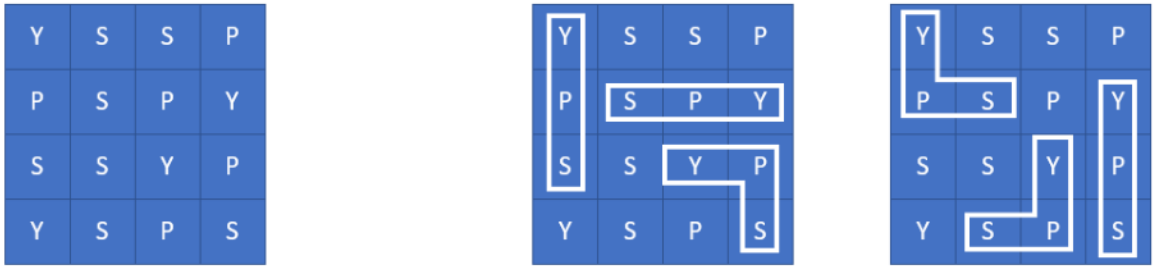


the shortest path of x-z should be 4

6) (18 pts)

Counter Espionage Academy instructors have designed the following problem to see how well trainees can detect SPY's in an nXn grid of letters S, P, and Y. Trainees are instructed to detect as many disjoint copies of the word SPY as possible in the given grid. To form the word SPY in the grid they can start at any S, move to a neighboring P, then move to a neighboring Y. (They can move north, east, south or west to get to a neighbor.) The following figure shows one such problem on the left, along with two possible optimal solutions with three SPY's each on the right. Give an efficient network flow-based algorithm to find the largest number of SPY's.

Note: We are only looking for the largest **number** of SPYs not the actual location of the words. No proof is necessary.



We construct a Flow Network as follows, with all edges having capacity 1:

1. Create one layer of nodes for all the Ss in the grid. Connect this layer to a source vertex s, directing edges from s to S.
2. Create two layers of nodes for all the Ps in the grid. Connect the first layer to the Ss based on whether or not they are adjacent in the grid, directing edges from S to P. Also, connect the first layer to the second layer by connecting the nodes that correspond to the same location. In other words, we are representing P's as an edge with capacity 1. This is similar to how we solved the node disjoint paths problem.
3. Create a layer of nodes for all the Y s in the grid. Connect these to a sink vertex t, directing edges from Y to t. Also, connect them to the second layer of Ps based if the P and Y are adjacent in the grid, directing edges from P to Y .

Note that we don't need to represent nodes S or P with edges of capacity 1 since there are edges of capacity 1 going into S's and edges of capacity 1 leaving Y's which will force these letters to be picked only once.

Claim/Answer: Value of Max Flow in this Flow Network will give us the maximum number of disjoint SPYs.

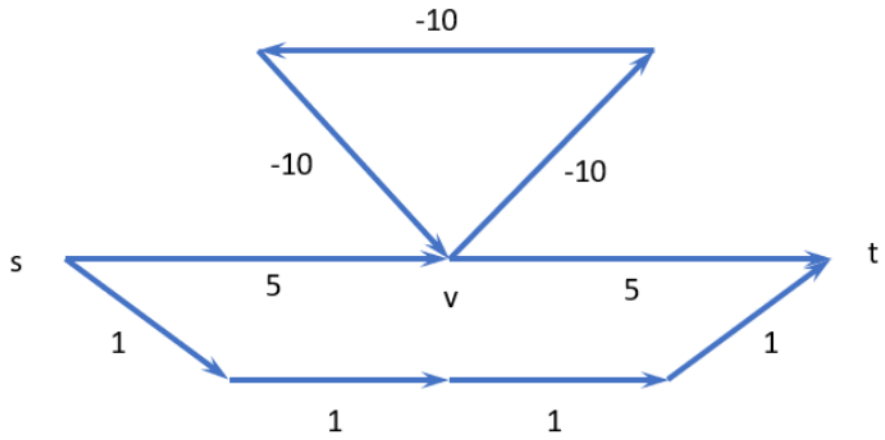
Scoring breakup:
Full points even if the solution splits every node into two nodes and connects respective pair with an edge of unit capacity (to ensure each node is picked only once)

4 Points: If solution satisfies the constraint of picking S's only once
4 Points: If solution satisfies the constraint of picking P's only once (Make sure solution has split P's into two vertices, otherwise the same P can be selected more than once if that P is adjacent to 2 S's and 2 Y's)
4 Points: If solution satisfies the constraint of picking Y's only once
4 Points: If edges are added only between the nodes which are adjacent. i.e. If the solution satisfies the constraint of picking only adjacent S, P and Y.

2 Points: For the correct claim/answer. i.e. If solution associates largest number of disjoint SPYs to the max flow in the network

7) (16 pts)
We know that the Bellman-Ford algorithm is not guaranteed to find the shortest **simple** path if the graph contains negative cost cycles.

a) Give an example of a **directed** graph (with numerical edge weights) with nodes s and t identified on the graph, where after the Bellman-Ford algorithm finds the shortest path from s to t and after we remove any negative cycles on that path, that simple path will not be the shortest simple path from s to t. (4 pts)



The shortest path found by Bellman Ford will use the negative cycle that is connected to node v. But once we remove the negative cycle, the path svt will not be the shortest path from s to t.

Note: Removing a cycle means removing any nodes or edges that cause the B-F path to have a cycle, and must end up with a simple path that was part of the B-F path (instead of no path).
Points are only given if the answer shows that the follow two paths are different, and the second one has lower cost than the first:
1. the shortest simple path by removing cycle from B-F
2. the shortest simple path in the original graph

2 points for creating a weighted graph with a negative cycle
1 point for mentioning the shortest path before and after removing the cycle
1 point for selecting two nodes with labels s and t

-3 the shortest simple path from B-F (without cycle) is the same as the shortest simple path in the original graph (there is only one path or the path still has the lowest cost after removing the cycle)

-1 undirected graph

b) Prove that finding the shortest **simple** path in an **undirected** graph with negative cost edges is NP-hard. (12 pts)

Hint: You can use HAM-Path or HAM-Cycle for your reduction.

Refer to the problem 'Shortest simple path w/ negative cost edges' as SSPN. Proof using HAM-PATH.

(1) We will show that HAM-PATH $\leqslant_p$ SSPN

- Given an instance of the HAM-PATH problem on graph G = (V, E)
- Construct G' with the same exact set of nodes and edges as in G with all edges having a weight/cost of -1
- Claim: G has a HAM Path if and only if SSPN instance has a path of cost – (|V | - 1) between two arbitrary nodes. (Need to call the blackbox $O(n^2)$ times each time using a different pair of starting and ending nodes)

Proof:
If there is a path of length – (|V | - 1) between two nodes s and t in G' this will give us a HAM Path in G
If there is a HAM Path in G, there will be two nodes s and t in G' (the two ends of the HAM Path in G) where cost of their shortest path will be – (|V | - 1)

(2) Proof using HAM Cycle.

We will show that HAM Cycle $\leqslant_p$ Shortest simple path w/ negative cost edges

- Given an instance of the HAM Cycle problem on graph G = (V, E)
- Construct G' with the same exact set of nodes and edges as in G with all edges having a weight/cost of -1
- Repeat for all edges:
  Remove edge (uv) in G', add two new nodes v' and u', and add edges uu' and vv' with weights -1.
  - Claim: G has a HAM-Cycle if and only if SP w/ negative cost edge has a path of length – (|V | + 1) between v'u'.

Proof:
If there is a path of length – (|V | + 1) between v'u' in G' this will give us a HAM Path from v to u in G which we can turn into a HAM Cycle by adding edge vu in G.

If there is a HAM Cycle in G, there will be two nodes v and u in G' (the two ends of one of the edges on the HM Cycle) where the cost of the shortest path between them will be – (|V | + 1)

(3) Alternate proof using HAM Cycle.

We will show that HAM Cycle ⩽p Shortest simple path w/ negative cost edges

- Given an instance of the HAM Cycle problem on graph G = (V, E)
- Construct G' with the same exact set of nodes and edges as in G with all edges having a weight/cost of -1
- Copy any node say A in G' and call the copy A'. Connect A' to all the same nodes that A is connected to with edges of cost -1
- Claim: G has a HAM-Cycle if and only if G' has a path of length – |V | between nodes AA'

Proof:
If there is a shortest simple path of length – |V | between A and A' in G' this will give us a HAM Cycle in G.
If there is a HAM Cycle in G, there will be a path of cost -|V| in G' which will be the shortest path between A and A' in G'

(4) Alternative proof using HAM Path:

We will show that HAM-PATH ⩽p SSPN

- Given an instance of the HAM-PATH problem on graph G = (V, E)
- Construct G' with the same exact set of nodes and edges as in G with all edges having a weight/cost of 1
- for each pair of nodes u and v in V add u' and v' to G', and set the weight of (u, u') and (v, v') to floor(|V| / 2) and ceil(|V| / 2)

- Claim: G has a HAM Path if and only if SSPN instance has a path of cost – (|V |) between u' and v'.

Proof:
If there is a path of length – (|V |) between the two nodes u' and v' in G' this will give us a HAM Path in G
If there is a HAM Path in G, there will be two nodes u' and v' in G' (the two ends of the HAM Path in G) where cost of their shortest path will be – (|V |)

(5) Alternative Proof using HAM-PATH with extra nodes.

We will show that HAM-PATH $\leqslant_p$ SSPN
- Given an instance of the HAM-PATH problem on graph G = (V, E)
- Construct G' with the same exact set of nodes and edges as in G, plus a s node that is connected to all nodes and a t node that is connected to all nodes (except s), with all edges having a weight/cost of -1
- Claim: G has a HAM Path if and only if SSPN instance has a path of cost – (|V | +1) between two arbitrary nodes.

Proof:
If there is a path of length – (|V | +1) between two nodes s and t in G' this will give us a HAM Path in G
If there is a HAM Path in G, there will be two nodes s and t in G' (the two ends of the HAM Path in G) where cost of their shortest path will be – (|V | + 1)

(6) Alternate Solution with longest path? (D2L user 69 by last name)
This problem can be reduced from the longest path problem, which is proven to be NP-Complete. If there were a black box that could solve the problem of finding the shortest simple path in an undirected graph with negative cost edges, given an instance of the longest path problem with at least length k, where all edge weights must be positive, we can negate all the edge weights and run our solution to get the longest path in the original graph.

We claim that this holds if an only if our problem has a longest path of k. If there is a shortest path of length k, then when the graph is negated, it subsequently represents the longest path through our graph. If there is a longest path of length k,

then when we negate the graph for our shortest path problem, it will now represent the shortest possible path through the graph.

- (if the answer is not correct) 2 points for ppl who wrote something (so -10 out of 12)
- 4 points for constructing G':
  - 2 points if similar weights are assigned to all edges of G
  - 2 points if negative weights are assigned to all new edges/specific edges
  - -1 points for missing details

- 2 points for calling the shortest path for all node pairs (first solution) or edges (second solution) (not needed for solution 3 and 5) - (Guanyang: just calling for all nodes is also fine)
- 2 points for finding what shortest simple path would be the hamiltonian path/cycle (e.g., weight - |V| + 1 in the first example,)
  - -1 point for wrong weight (e.g. |V|-1)
- 4 points for "if and only if" proof:
  - 2 points for the "if" side, mapping the correct answer of shortest path to HAM
  - 2 points for the only if side, proofing the HAM path/cycle correspond to a valid shortest path
- -2 points for claiming that the problem is NP
- -2 points for opposite direction of reduction

Additional Space

Additional Space

Additional Space