TF Questions

1-Suppose you come up with a Divide & Conquer algorithm which when given a problem of size n, divides it into 3 sub-problems each of size half of the original problem. Moreover, say your algorithm takes Θ (n) time in total to divide and combine the solutions to sub-problems.

Let T(n) denote the running time of your algorithm on an input problem of size n. Assuming appropriate base cases, T(n) satisfies the following recurrence: $T(n) = 2T(n/3) + \Theta(n)$

False: The correct recurrence should be $T(n) = 3T(n/2) + \Theta(n)$.

2-A flow network with unique edge capacities could have more than one min cut.

True: Nodes S, A, B, C, T; Edges SA, AB, AC, ST, CT. Capacities 3, 2, 1, 10, 100 resp. Then {SA} and {AB, AC} are both min-cuts.

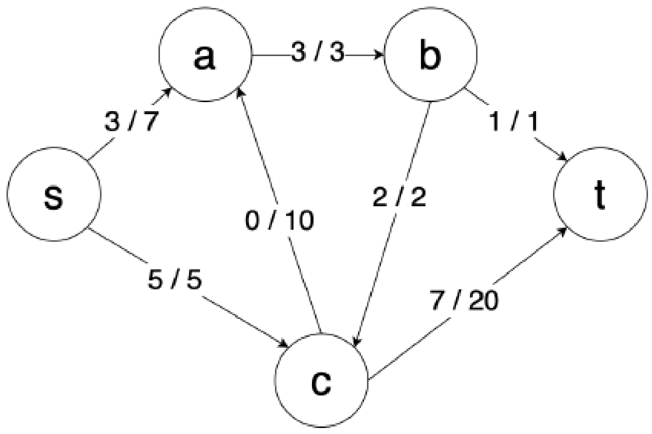
3-The bellman ford algorithm may take different number of iterations to converge, depending on the order in which it examines the vertices of the graph.

True: Demonstrated in class

4-Recall the Subset Sum problem we studied in class: Given a set of n requests (in which request i takes time w_i to process) and an integer W such that you can schedule requests any time between 0 and W, the
objective was to schedule requests to maximize the machine's utilization. We presented a solution to this problem which had a pseudo-polynomial run time of O(nW).
Now suppose we learn that W is upper bounded by n^2. Let us call this new problem (i.e. the one where W is upper bounded by n^2) as Upset Sum. Then, the same algorithm solves upset Upset Sum in polynomial-time.

True: The runtime of the same algorithm is now $O(nW) \leq O(n \times n^2) = O(n^3)$ which is polynomial in the input size.

5-Given a flow network below where each edge e is labelled f_e/c_e where f_e is the flow on e and c_e is the capacity of e. Then, f (the flow given) is a max-flow for this network.



True: There are multiple ways to see why the given flow is a max-flow of this network. We outline one way here.
Notice that the value of f is 8. Moreover, observe that the capacity of the cut {s,a}, {b, c, t} is $3 + 5 = 8$.
Since value of any flow is upper bounded by the capacity any cut, it follows that f is a max-flow.

6-The Ford-Fulkerson algorithm is not guaranteed to converge to max flow for some non-integer edge capacities.

True: As mentioned in lecture, for some combination of irrational numbers as edge capacities Ford Fulkerson converges to incorrect max flows.

7-In a flow network with no two edges having the same capacity, let e be the edge with smallest capacity. Then e must cross some min-cut.

False: Take the example as in the answer of Q2 above. Change the capacity of AC to 5. Then {SA} is the only min-cut and thus the smallest edge AB is in no min-cut.

MC Questions

8-Which of the following best describes the meaning of pseudo-polynomial runtime?

(a) The runtime includes both polynomial and logarithmic terms.
(b) The runtime is polynomial in terms of the numeric value of the input.
(c) The runtime is polynomial in terms of the number of integers in the input.
(d) The runtime is polynomial in terms of the number of bits in the input.

9-Which two of the following recurrences have the solution with the same complexity?

a) T(n) = 2T(n/2) + 1
b) T(n) = 2T(n/3) + n
c) T(n) = 2T(n/2) + n
d) T(n) = 2T(n/2) + nlogn

10-Which of the following best describes how to calculate the runtime of dynamic programming algorithms?

(a) The number of unique subproblems times the input size of each subproblem
(b) The number of unique subproblems times the input size of each subproblem plus the runtime of solving each subproblem
(c) The number of unique subproblems times the runtime of solving each subproblem
(d) The number of unique subproblems times the runtime of solving each subproblem times the number of occurrences of each subproblem

Problem 11

Recall the weighted interval scheduling problem we studied in class: given n requests where request i has start time si, finish time fi and weight wi -- the goal is to select a subset S of mutually non-overlapping requests such that it maximizes the total weight of S.

Let us denote each request i by the 3-tuple (si, fi, wi). Given the following set of requests: {(5, 7, 20), (1, 3, 10), (4, 9, 40), (2, 6, 20), (8, 10, 30)}, what is the weight of the optimal subset S (i.e. what is the value of the optimal solution)?

You can use the definition of the sub-problems and the recurrence formula given in class {opt(i) = Max (wi + opt(p(i)), opt(i-1))} for this problem and then solve the problem numerically. We are not looking for the optimal subset, only for the weight of the optimal subset.

You need to show all your steps in solving this problem using dynamic programming with the given recurrence formula. In other words, just giving the final answer will not receive any credit.

Sort requests in increasing order of finish times:

{(1, 3, 10), (2, 6, 20), (5, 7, 20), (4, 9, 40), (8, 10, 30)}.

opt(1) = Max(w1 + opt(p(1)), opt(0)) = Max(20 + opt(0) = 20 + 0 = 20, 0) = 20

opt(2) = Max(w2 + opt(p(2)), opt(1)) = Max(10 + opt(1), opt(1)) = Max(10 + 20 = 30, 20) = 30

opt(3) = Max(w3 + opt(p(3)), opt(2)) = Max(40 + opt(2), opt(2)) = Max(10 + 40 = 50, 20) = 50

opt(4) = Max(w4 + opt(p(4)), opt(3)) = Max(20 + opt(0), opt(3)) = Max(20 + 10 = 30, 50) = 50

opt(5) = Max(w5 + opt(p(5)), opt(4)) = Max(30 + opt(2), opt(4)) = Max(30 + 30 = 60, 50) = 60

The value of the optimal solution is 60.

Rubric:

- -4 if not solving this problem numerically, or some steps are missing, or the final answer is not correct
- -4 if not using Dynamic Programming with the given formula
- -2 for any other minor issue

Problem 12

The cows are back in Exam 2! Except this time, they will be jumping haybales.

Consider an *n* x *m* field of haybales. The "#"'s identify the locations of the haybales.

```
..#.###.
.###.#..
..#.#.#.
.#..#...
```

The cow starts in the top left corner and is trying to make its way to the bottom right corner. The cow can jump either right or down in the field a distance of up to *k* cells with a single jump. Your task is to determine the minimum number of jumps needed to reach the bottom right cell without the cow landing headfirst on any haybales.
Design a dynamic programming solution to this problem. Analyze the runtime and memory usage. Your runtime should not be worse than O(nmk) and your memory usage should not be worse than O(nm).

a) Define (in plain English) unique subproblems to be solved

4 pts – minimum jumps to get to bottom right for each cell on grid
3pts – mention of minimum jumps over each cell
2pts – minimum jump answer that does not include each cell, or jumps to target cell or from source cell for each cell
1pt – path or decision left/right
0pts – restatement of overall problem

b) Write the recurrence relation for subproblems. No justification needed.

6 pts – fully correct recurrence relation
4-5pts – minor typo
3pts –missing k loop
2pts – misread question, if consistent

c) Using the recurrence formula in part b, write pseudocode (using iteration) to compute the minimum number of jumps needed to reach the bottom right cell without landing headfirst on any haybales. Make sure you have initial values properly assigned.

6 pts – fully correct pseudo code, initialization, return
4-5pts – minor typos, unclear initialization

3pts – missing k loop
1-2pts – missing iterative solution

d) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not.

3 pts - correct runtime, credit given to both pseudo-polynomial and reduced polynomial
2 pts – correct runtime, no polynomial statement
1 pts - incorrect runtime if question was misread

Surprisingly, there is an O(nm) algorithm to this problem, but is probably too difficult for an exam and requires a clever combination of a data structure plus dynamic programming to shave off the O(k) factor from the solution below, which is all we require and takes O(nmk) time. The problem is an extension of the grid problem shown in discussion.

We consider the state OPT(i,j) where (i,j) is the current cell of the cow trying to reach the bottom right corner. We then consider transitioning down to cells OPT(i+x,j) or right to cell OPT(i,j+x) where x is a value from [1,k]. The base case is the bottom right corner which requires 0 jumps to reach and a base case of infinity if (i,j) contains a haybale. Some care is needed to not transition to states off the grid. We get the recurrence:

$$\mathrm{OPT}(i,j) = 1 + \min\Big(\min_{x \text{ in } [1,k] \text{ and } i+x <= n} \mathrm{OPT}(i+x, j),\; \min_{x \text{ in } [1,k] \text{ and } i+x <= m} \mathrm{OPT}(i, j+x) \Big)$$

The solution requires O(nm) states. For each state we consider O(k) neighboring states, yielding a solution with O(nm) memory and O(nmk) runtime.

Problem 13

Given n positive integers a_1, a_2, …, a_n. What is the maximum value of a_i / a_j where i < j. Design an O(n)-time divide-and-conquer algorithm, and justify the time complexity of your algorithm.

a) Describe your divide step

Divide the array into two (almost) equal subarrays: a_1 to a_n/2 and a_n/2+1 to a_n

- 2 points, no partial credit.

b) Describe the information you will send back to the parent subproblem during recursion at each step

Absolute minimum in the subproblem solved
Absolute maximum in the subproblem solved
Maximum a_i/a_j in the subproblem solved where i<j

- 6 points
  - -2 if missing either absolute minimum/maximum or maximum a_i/a_j

c) Describe your combine step

Absolute minimum = Min (left and right absolute min's)
Absolute maximum = Max (left and right absolute max's)
Maximum a_i/a_j = Max (left maximum, right maximum, left absolute max/right absolute min)

- 2 points, no partial credit.

d) Write complete pseudocode to present your divide and conquer solution.

Solution:

This is almost the same problem as the stock market problem solved in class

```
FUNC(a_l, …, a_r){ // return (ans, min, max) for a[l,…,r]
    if l == r then
        return (none, a_l, a_r)
    m = floor((l+r)/2);
    (ans_l, min_l, max_l) = FUNC([a_l, …, a_m])
    (ans_r, min_r, max_r) = FUNC([a_{m+1}, …, a_r])
    Return (MAX{ans_l, ans_r, max_l/min_r}, MIN{min_l, min_r},
MAX{max_l, max_r})
}
```

- 6 points
  - -2 if the result is not always optimal for missing certain cases
  - -4 if the algorithm does not find the maximum value of a_i / a_j where i < j
  - No partial credit if it is incomplete (outputs are unknown or the code is not implementable)

e) Justify the runtime complexity of your algorithm

The complexity of this algorithm is T(n) = 2T(n/2) + O(1) = O(n).

- 3 points
  - No partial credit if the runtime analysis is not in O(n) or the analysis is not correct.

Problem 14

Consider the below flow-network, for which an s-t flow has been computed. The numbers *x/y* on each edge shows that the capacity of the edge is equal to *y*, and the flow sent on the edge is equal to *x*.

a. What is the current value of flow?

No partial credit if the answer is not correct.

b. Draw (or list edges with edge capacities of) the residual graph for the corresponding flow-network.

At most 2 points if the answer is not completely correct.

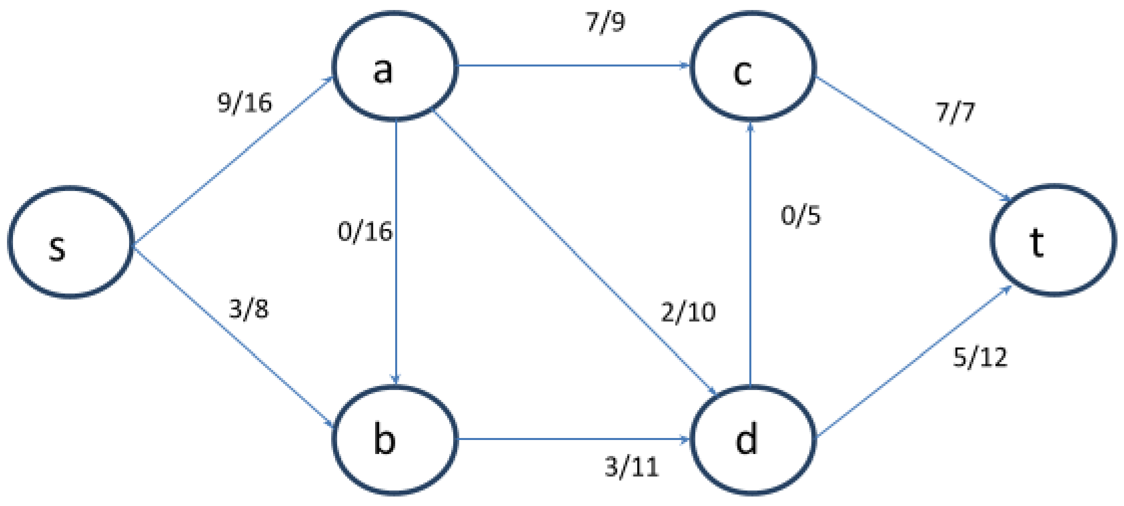
c. Calculate the maximum flow in the graph using the Ford-Fulkerson algorithm. You need to show/state the augmenting path at each step and final max flow.

No points if you do not find the maximum flow.

At most 2 points if the augmenting path is not shown.

d. Show/state the min cut found by the max flow you found in part c.

No partial credit if the answer is not correct.

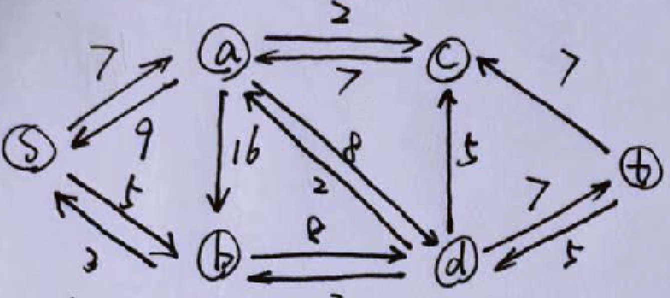


a. 12

d. min cut

$\{s, a, b, c, d\}, \{t\}$.

b.

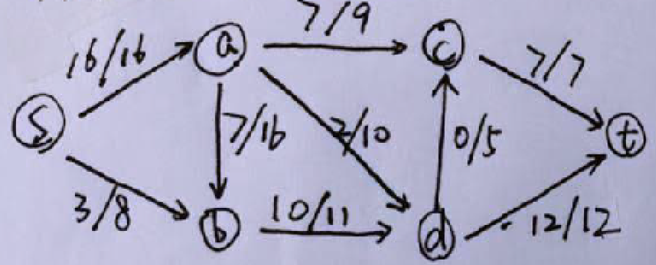


Ans 1.

C. Augmenting Path:

$s \to a \to b \to d \to t$ 7

Final Max Flow

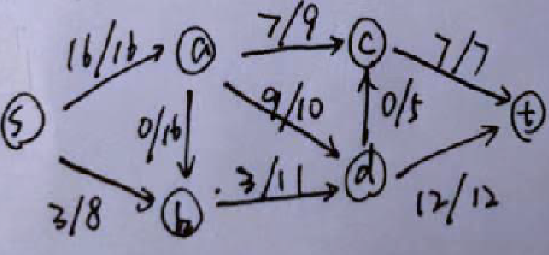


Ans 2

Augmenting Path:

$s \to a \to d \to t$ 7

Final Max Flow


s
a
b
c
d
t
16/16
7/9
7/7
0/16
9/10
0/5
3/8
3/11
12/12