1) 20 pts

Mark the following statements as **TRUE** or **FALSE**. No need to provide any justification.

[TRUE/FALSE]
Given the value of max flow, we can find a min-cut in linear time.

[TRUE/FALSE]
The Ford-Fulkerson algorithm can compute the maximum flow in polynomial time.

[TRUE/FALSE]
A network with unique maximum flow has a unique min-cut.

[TRUE/FALSE]
If all of the edge capacities in a graph are an integer multiple of 3, then the value of the maximum flow will be a multiple of 3.

[TRUE/FALSE]
The Floyd-Warshall algorithm always fails to find the shortest path between two nodes in a graph with a negative cycle.

[TRUE/FALSE]
0/1 knapsack problem can be solved using dynamic programming in polynomial time, but not **strongly** polynomial time.

[TRUE/FALSE]
If a dynamic programming algorithm has *n* subproblems, then its running time complexity is $O(n)$.

[TRUE/FALSE]
The Travelling Salesman problem can be solved using dynamic programming in polynomial time

[TRUE/FALSE]
If flow in a network has a cycle, this flow is not a valid flow.

[TRUE/FALSE]
We can use the Bellman-Ford algorithm for undirected graph with negative edge weights.

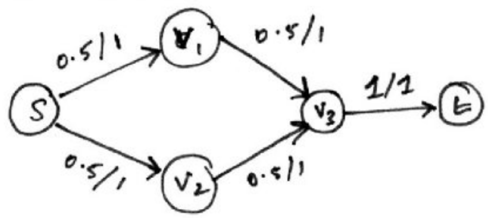
2) 10 pts

Given $N(G(V,E), s, t, c)$, a flow network with source s, sink t, and positive integer edge capacities $c(e)$ for every $e \in E$. Prove or disprove the following statement:

A maximum flow in an integer capacity graph must have integral (integer) flow on each edge.

False.

Proof by example:



→ In this graph; each edge has capacity = 1. (Integer)

But we can have edges with decimal flow (0.5)

Which satisfies both max flow constraints:

- Capacity constraint : $0.5 \leq 1$ ✓
- Conservation of flow ✓.

Q3 Score: **10.0**

3) 10 pts

The subset sum problem is defined as follows: Given a set of n positive integers $S=\{a_1, a_2, ..., a_n\}$ and a target value T, does there exist a subset of S such that the sum of the elements in the subset is equal to T? Let's define a boolean matrix M where M(i, j) is true if there exists a subset of $\{a_1, a_2, ..., a_i\}$ whose sum equals j. Which one of the following recurrences is valid? (circle one)

1) $M(i, j) = M(i-1, j) \vee M(i, j- a_i)$

2) $M(i, j) = M(i-1, j) \vee M(i-1, j- a_i)$

3) $M(i, j) = M(i-1, j-1) \vee M(i-1, j)$

4) $M(i, j) = M(i, j- a_i) \vee M(i-1, j - a_i)$

Answer : 2

Proof: Any any time, we can either use the value of the ith element of the array <u>or</u> not use that value.

→ If we use the ith value, the value (a_i) should be deducted from the target value (j) and the recurrence should occur on the subproblem having one less element in the array (i-1) and the new target value ($j - a_i$).

→ If we do not use the last value; we just ignore that value and the ~~subproblem~~ recurrence will occur on the subproblem with one less element (i-1) but same target value (j).

Q4 Score: **14.0**

4) 15 pts

This problem involves partitioning a given input string into disjoint substrings in the cheapest way. Let $x_1x_2 \ldots x_n$ be a string, where particular value of x_i does not matter. Let $C(i,j)$ (for $i \leq j$) be the given precomputed cost of each substring $x_i \ldots x_j$. A partition is a decomposition of a string into disjoint substrings. The cost of a partition is the sum of costs of substrings. The goal is to find the min-cost partition.
An example. Let "ab" be a given string. This string can be partitioning in two different ways, such as, "a", "b", and "ab" with the following costs $C(1,1) + C(2,2)$ and $C(1,2)$ respectively.

a) Define (in plain English) subproblems to be solved. (5 pts)

- For each substring in the input string; we have to find all possible combinations of substrings.
- The combination of substrings with minimum cost will be the min-cost partition.

b) Write the recurrence relation for subproblems. (7 pts)

Let OPT(i,j) be the min-cost of a partition of a string from indexes i to j.

$$OPT(i,j) = \begin{cases} OPT(i,i) = C_{ii} \\ OPT(i,j) = \min\left[C_{ik} + opt(k+1, j) \right] \quad \text{for all } i \leq k \leq j \end{cases}$$

c) Compute the runtime of the algorithm. (3 pts)

- Each subproblem takes $O(n)$ time to compute.
- each subproblem is computed for all combinations of i & j
  $\therefore$ n^2 times

$\therefore$ Overall complexity = $\boxed{O(n^3)}$

for i from 1 to n
for j fr 1 to i
Reccurence () → O(
end for
end for.

O(n) O(n)

5) 15 pts

You have two rooms to rent out. There are n customers interested in renting the rooms. The i^{th} customer wishes to rent one room (either room you have) for d[i] days and is willing to pay bid[i] for the entire stay. Customer requests are non-negotiable in that they would not be willing to rent for a shorter or longer duration. Devise a dynamic programming algorithm to determine the maximum profit that you can make from the customers over a period of D days.

n i d(i) bid(i)

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

· Since we have 2 rooms, the first customer can be put up in room one or room 2. If he is put in room one, the subproblems would be ~~st~~ handled by each room with different parameters. The number of days for that room would reduce while the number of days for the other room would remain the same. ∴ we have to find the ~~min~~ max such combination.

b) Write the recurrence relation for subproblems. (7 pts)

Let rooms be A and B. Let $OPT_A(i,j)$ be optimal solution for using room a with i customers and j days. Same for OPT_B.

$OPT_A(i,j) = \max(bid[i] + opt_A(i-1, j-d[i]), bid[i] + opt_B(i-1, j))$

$opt_B(i,j) = \max(bid[i] + opt_B(i-1, j-d[i]), bid[i] + opt_A(i-1, j))$

$\boxed{\text{Overall}: \max(opt_A(i,j), opt_B(i,j))}$

c) Compute the runtime of the algorithm. (4 pts)

$\boxed{\text{Overall Complexity}: O(n^2)}$

Each subproblem is solved in $O(n)$ time.

The subproblems are done n time each.

for i = 1 to n.

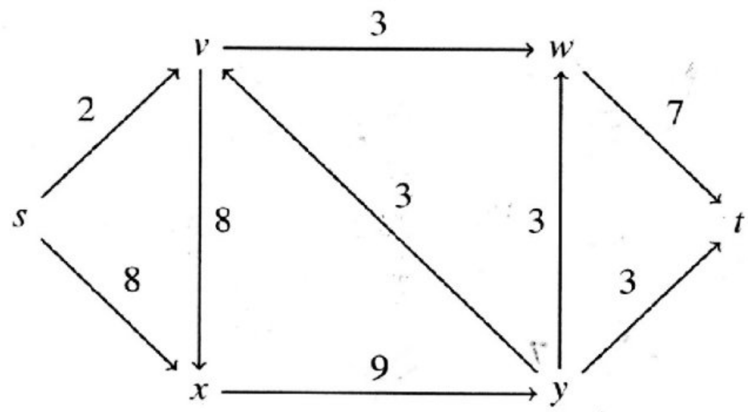
O(n) (each subproblem → (O(n))

end for

Q6 Score: **15.0**

6) 15 pts

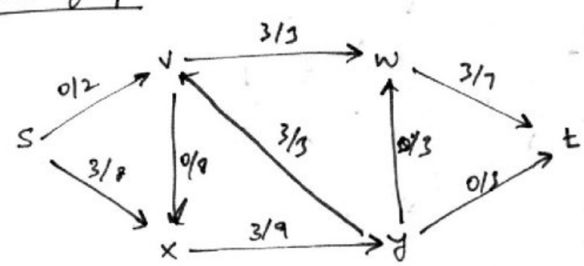
You are given the following graph. Each edge is labeled with the capacity of that edge.



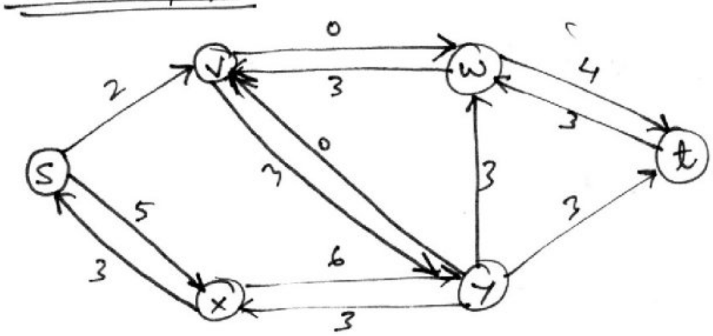
a) Draw the corresponding residual graph after sending as much flow as possible along the path s→x→y→v→w→t. (5 pts)

· The max flow along the path s→x→y→v→w→t is 3.

→ Flow graph:

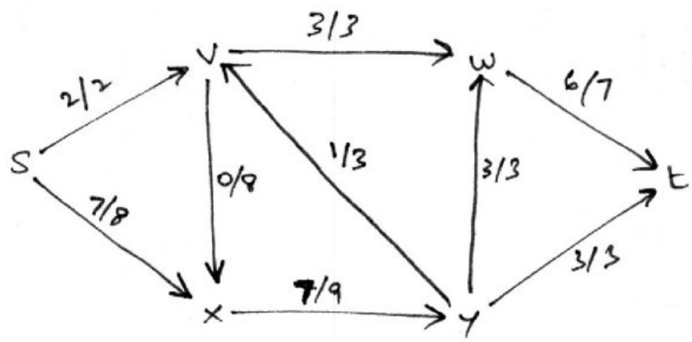


→ Residual Graph:



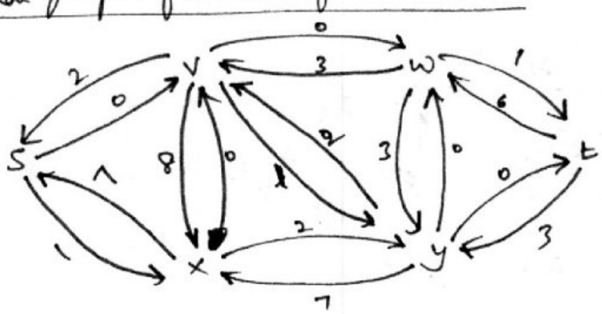
b) Find the value of a max-flow. (5 pts)

The max flow is : 9.



c) Find a min-cut? (5 pts)

Residual graph for max flow : 9.



∴ Min cut (A, B) :

A = { s, v, x, y }

B = { w, t }

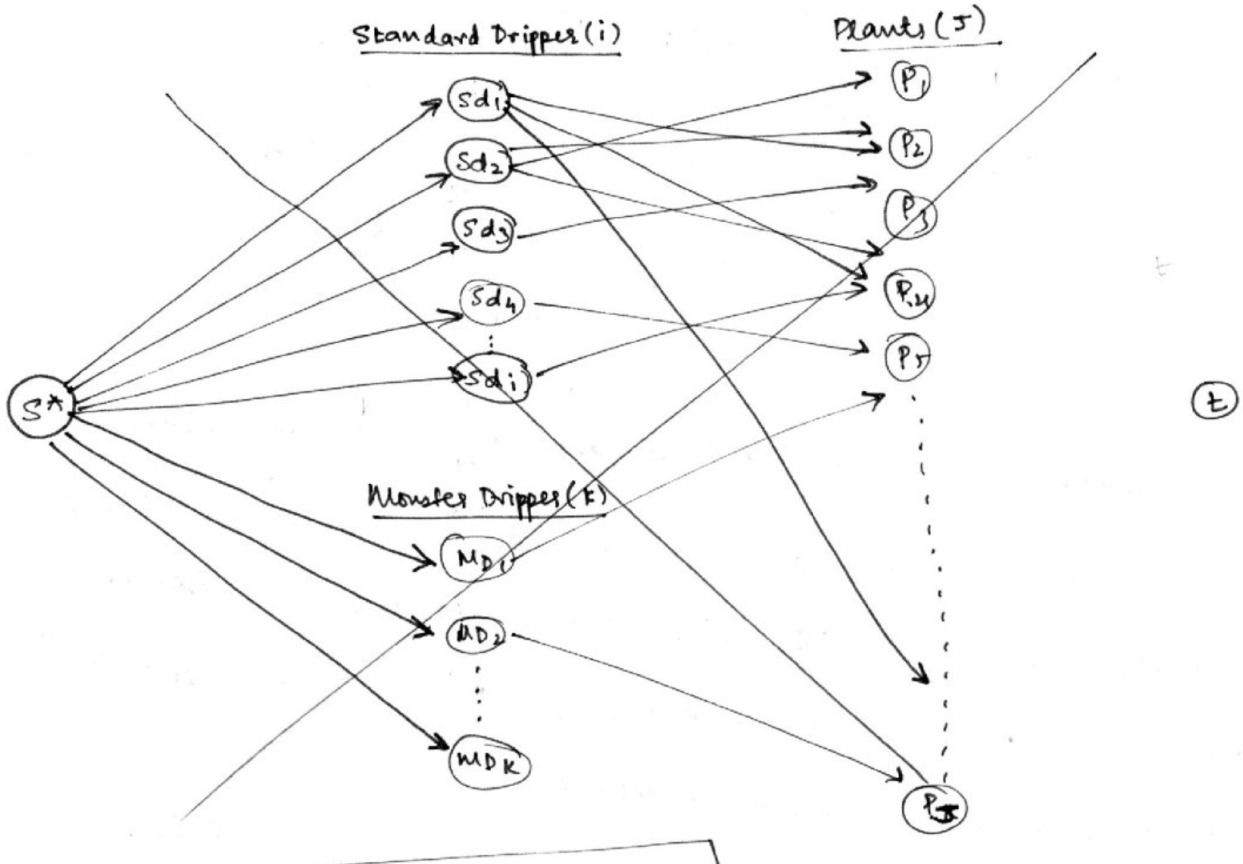
Q8 Score: **15.0**

7) 15 pts

Consider a drip irrigation system, which is an irrigation method that saves water and fertilizer by allowing water to drip slowly to the roots of plants. Suppose that the location of all drippers are given to us in terms of their coordinates (x^d, y^d). Also, we are given locations of plants specified by their coordinates (x^p, y^p).

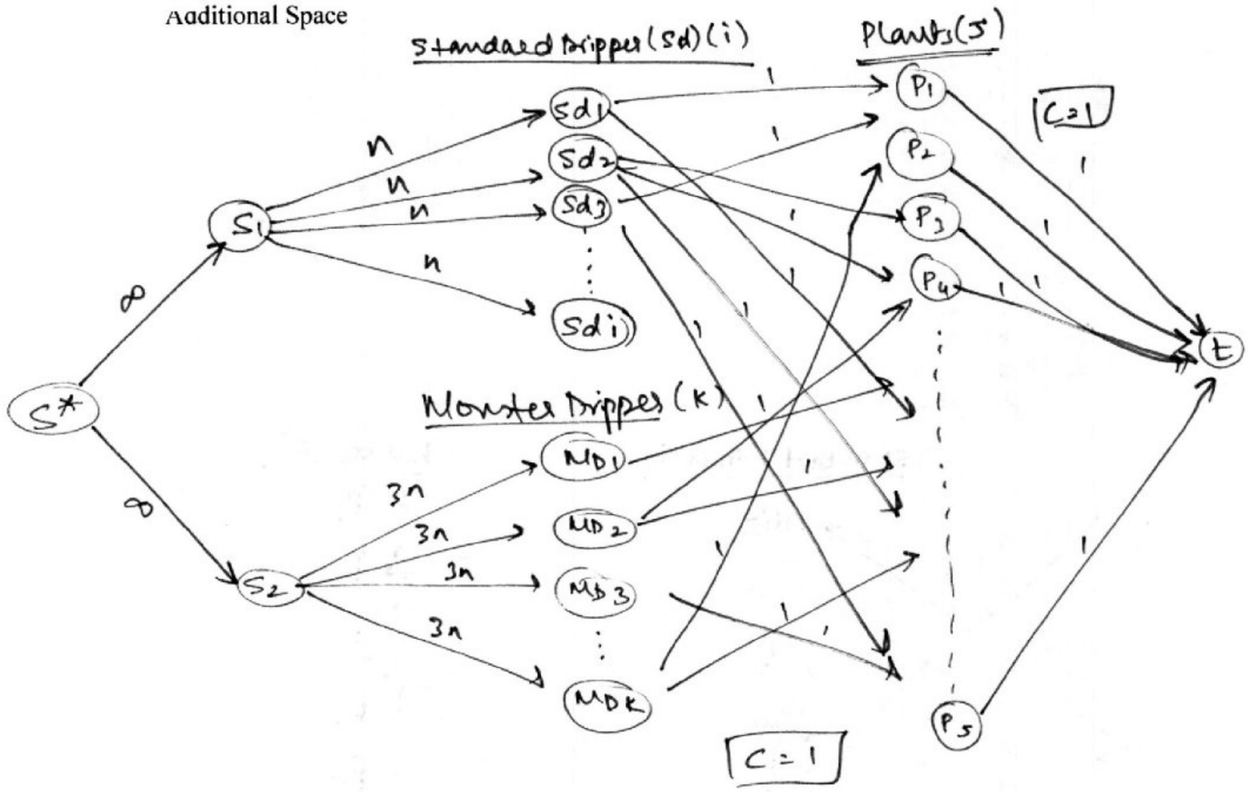
A dripper can only provide water to plants within distance l. A single dripper can provide water to no more than n plants. However, we recently got some funding to upgrade our system with which we bought k monster drippers, which can provide water supply to three times the number of plants compared to standard drippers. So, we now have i standard drippers and k monster drippers.

Given the locations of the plants and drippers, as well as the parameters l and n, decide whether every plant can be watered simultaneously by a dripper, subject to the above mentioned constraints. Justify carefully that your algorithm is correct and can be obtained in polynomial time.



Continued on Additional sheet →

Additional Space



→ This problem can be modeled as a max flow problem.

· Create nodes Sd1, Sd2 .. Sdi for all the standard drippers

· Create nodes MD1, MD2 ... MDK for all Monster drippers.

· Add 2 local source nodes S1 and S2 to suppy the standard and monster drippers.

· Add a super source node S* for ~~fot~~ both S1 and S2.

· ~~The capacity~~ Add nodes P1, P2 ... PJ for all plants.

· Add a sink node 't' and connect all PJ to (t).

Additional Space

Edge capacities:

- from S* to S1 and S2; set capacity as ∞.
- from S_1 to standard dripper, set capacity as 'n'.
- from S_2 to monster dripper set capacity as '3n'.
- Connect edges (S_D and M_D → P) only if that plant is reachable from that dripper. The capacity of these edge is ~~One~~ 1.
- Connect all plants to the target node with capacity 1.

Claim: If the max-flow on this graph is equal to the number of plants (5) then each plant can be watered simultaneously.

Proof:
- Each dripper receives the maximum amount of flow as the number of plants that it can water (n or 3n).
- each dripper is connected by a unit capacity edge to a plant which signifies that, it will take responsibility of watering that plant.
- Each plant can be watered by only one dripper (because (p → t) has capacity 1.
- If flow = number of plants. Then as each plant can receive only one unit of flow; ∴ each plant is watered.

9E2F0461-F8E7-4098-8038-8DC0F08B865D

Additional Space

Also, we can say that if flow is ≤ no. of plants; some plant does not receive water.

omplexity: Max flow can be computed in polynomial time.

Q12 Not graded

Additional Space

Q13 Not graded