The subproblems that a particular subproblem,

$\delta_k(s,w)$ depends on have the structure $\delta_i(s,p)$ where $i < k$, and we stop recursing when $i = 0$ ($\delta_0(a,b)$ is not ∞ iff $a=b$)

Thus it is impossible for a subproblem

$\delta_k(s,w)$ to depend eventually on itself ($\delta_k(s,w)$) as k value decreases every time a subproblem dependency occurs.

Q.4.d The running time complexity is asymptotically the number of subproblems times the time taken per subproblems.

#subproblems: Any subproblem for computing $\delta_k(s,t)$ has structure $\delta_i(s,p)$ where $i < k$ and p is some arbitrary vertex. We start with $k = n-1$ (in an n-vertex graph the length of any path is atmost $n-1$). Thus there are $n-1$ choices for k and n choices for p. Thus #subproblems $\leq (n-1) \times n = O(n^2)$

[Note: Even if you compute it as: a subproblem is $\delta_k(u,v)$ where $0 \le k \le n-1$, $1 \le u,v \le n$, and thus #subproblems $\le O(n^3)$, you will get full points]

Time taken/subproblem: We need to compute the minimum which is over w which have an edge $u \to v$ going into v. Thus the minimum is taken over at most E elements in the worst case.

With memoization we can assume that $\delta_{k-1}(s,w)$ is $O(1)$ time, weight$(w \to v)$ is another $O(1)$ lookup in the adjacency matrix of G, doing the addition of these quantities is $O(1)$. Thus each quantity inside the min can be computed in $O(1)$ time, and thus computing min takes, $E \times O(1) \le O(E)$ time.

Total time: $O(n^2) \times O(E)$

2.4.b

Initialize δ = int[n] x int[n] x int[n], a 3-dimensional global array to hold the memoized values $\delta[k][u][v]$, initialized to -1 to denote that it is not computed yet

```
for k=0 to n-1:
    for u=1 to n:
        δ[k][u][u]=0
    end for
end for
return computeδ(n,s,t)
```

Base case for recursion.

Procedure: computeδ(k,u,v):

```
If k=1:
    If u→v ∈ E(G): δ[1][u][v] = wt(u→v) & return;
    else: δ[1][u][v] = ∞ & return;
else
    If there is no w s.t. w→v ∈ E(G):
        δ[k][u][v]=∞ & return;
```

else (if at least one $w \to v \in E(G)$)

$min \leftarrow \infty$

for each $w \in Adj(v)$: (That is $w \to v \in E(G)$)

if ($\delta[k-1][u][w] = -1$): compute $\delta(k-1, u, w)$

else:

if ($min > \delta[k-1][u][w] + wt(w \to v)$)

$min \leftarrow \delta[k-1][u][w] + wt[w \to v]$

end if

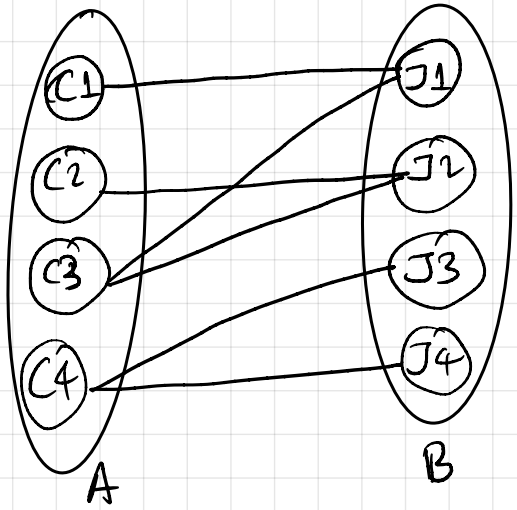
end for

$\delta[k][u][v] = min$ & return;

Q.4.d Estimate the asymptotic running time complexity of the DP algorithm by computing the number of subproblems and the time taken per subproblem. **(2 points)**

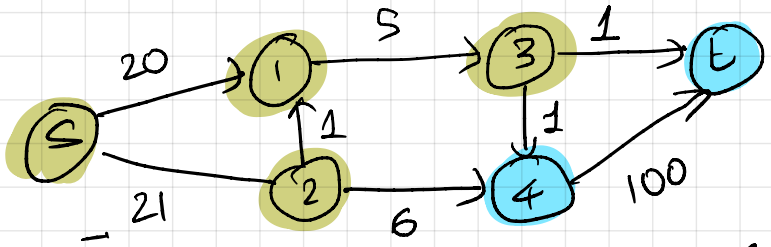
Section 3 : Max-Flow Min-Cut

Question 5 Given a bipartite graph G(A,B), a perfect matching is a set of edges of G so that no two of them share an end vertex and every vertex in A and B is the end point of some edge in the set. For example if A denote a set of candidates, and B denotes a set of jobs, and (candidate_i,job_j) is an edge G denoting that candidate_i prefers job_j, then a perfect matching of G(A,B) is an assignment of jobs to candidates, so that every job is assigned to a unique candidate who prefers that job and all the candidates end up getting a job. Using the Ford-Fulkerson algorithm for computing the maximum flow in a network, find if there is a perfect matching in the following bipartite graph? Draw the (s,t) network on which the computing the maximum flow lets us answer whether or not there is a perfect matching from G(A,B). Compute the max-flow of this network, and draw the residual graph for each step. **(8 points)**



Question 6 The maximum flow of a graph is equal to the capacity of the minimum cut for any flow network. For the following flow network, find a cut of capacity < 10 and explain why this implies that maximum flow in the network is at most 10. (**4 points**)

Since max-flow = min cut and capacity of any (S,T) cut in the graph is greater than or equal to the capacity of the min-cut (as it is the minimum), showing a cut of capacity ≤10, shows that max flow ≤10

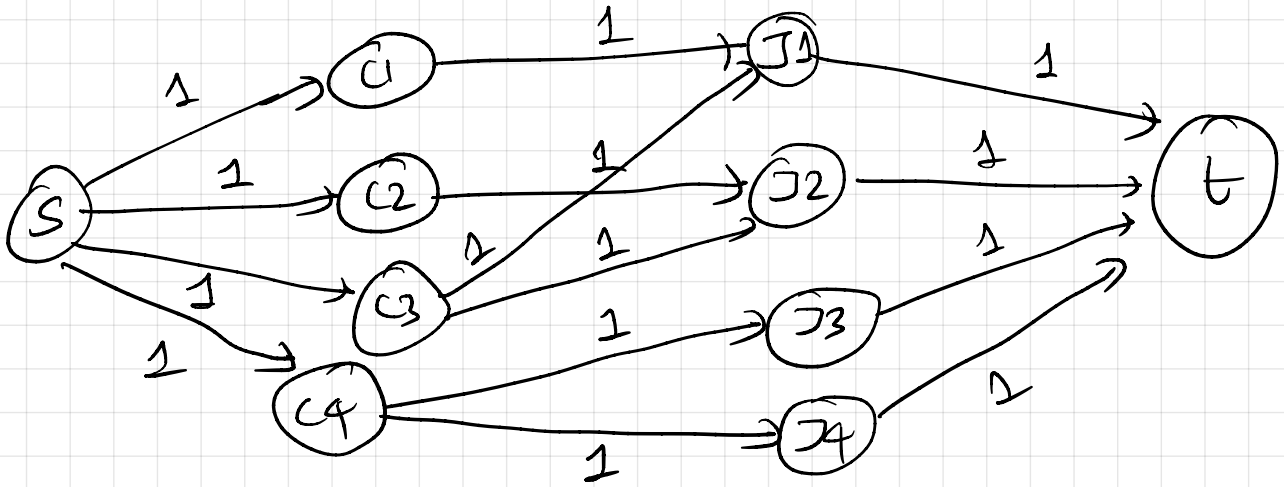


cut(S,T) (This is a valid cut as s is in S, and t is in T)

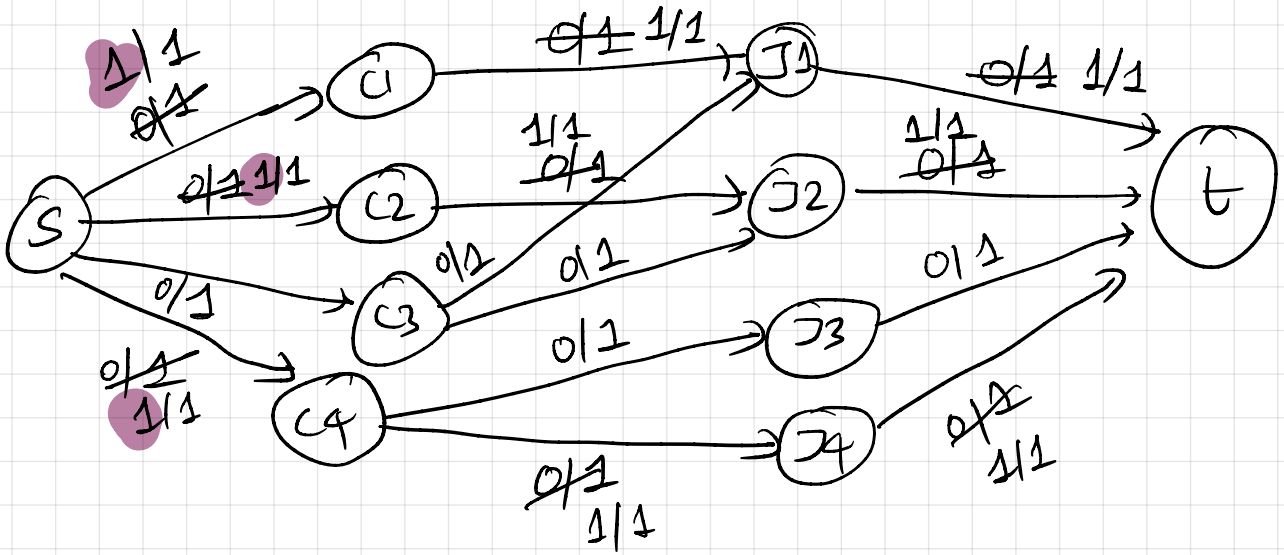
{S, 1, 2, 3} {4, T}

capacity(S,T) = wt(3→t) + wt(3→4) + wt(2→4) = 6 + 1 + 1 ≤ 10.

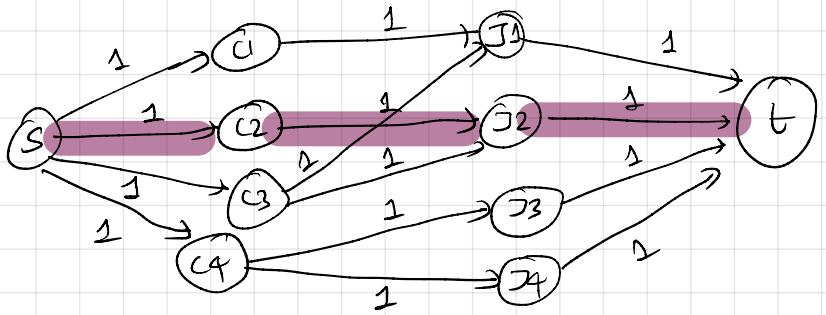
Question 5: There is perfect matching between Jobs and Candidates if and only if the max-flow in the following graph has flow value = # candidates = # jobs = 4



For any edge we will denote its current flow, capacity as $u \xrightarrow{f/w} w$. We start with all edges having flow 0.

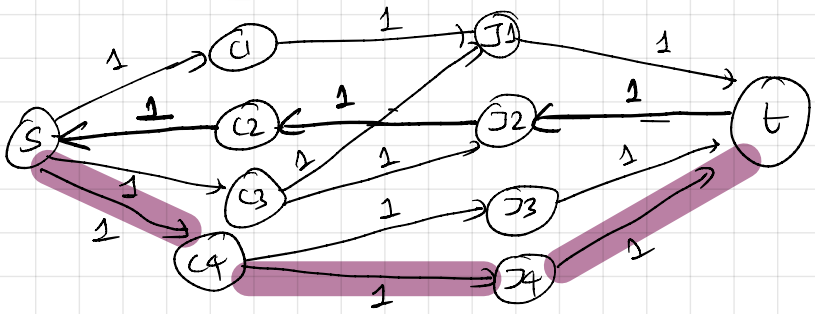


Residual Graph #1

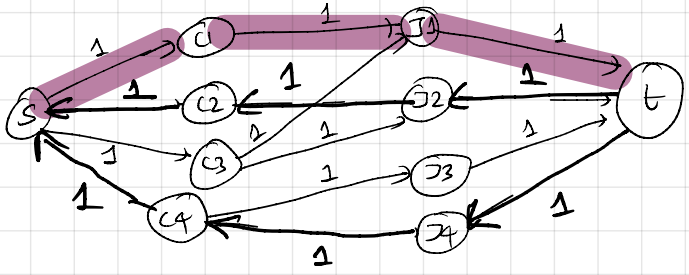


(augmenting path highlighted)

Residual Graph #2

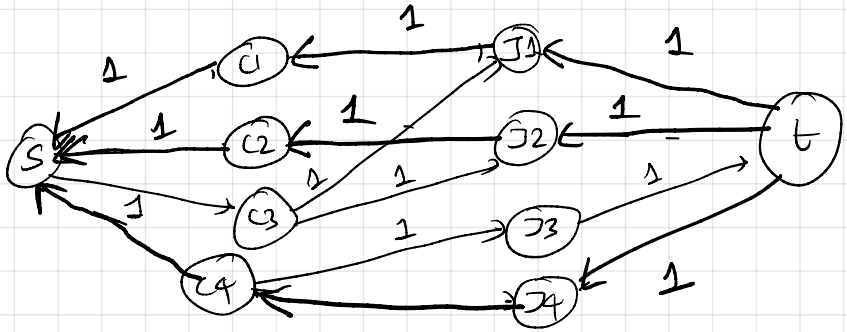


Residual Graph #3



Max-flow is the flow out of s when there is no augmenting path in the residual network and thus is equal to 3. Thus there is no Perfect Matching in G as $3 < 4$.

Residual Graph #4



In this graph there is no augmenting path (s to t path) as you can only go to C3 from s and from C3 you can only reach J1, J2 but they cannot reach t. Thus the current flow we have is the max-flow.

Section 4 : Order Statistics

Question 7 : Given an array A of n integers, a beta-approximate median for a 0 < beta < 1, is an element a = A[i] in the array such that at most beta x n elements are less than a in A and at most beta x n elements are greater than a in A. Compute a 7/10-approximate median in the array A=[21,8,7,5,14,11,10,81,9,10,21,22,23,24,25] of n=15 elements by dividing the array into consecutive blocks of 5 elements each (that is 3 blocks in total), and then finding the median of each block, and then taking the median of these three median values to be the 7/10 approximate median. Argue that it is a 7/10 approximate median by showing that at most 7n/10 = 7 x 15 / 10 ~ 11 elements are greater than the approximate median you got in A and at most 7n/10 ~ 11 elements are less than it in A. (**4 points**)

Section 5 : Randomized Algorithms

Question 8 : Markov's inequality states that for any non negative random variable X and a constant a > 0, the probability that X exceeds a x Expectation[X] is at most 1/a. Use this to argue that if randomized algorithm has an expected running time of O(n^c) for some constant c>0 (that its expected running time is polynomial), then the probability that it runs in time 2^n is very small. **(4 points)**

Question 9: Given a random process in which there are two events success and failure, where Pr[success]=p for every trial independent of other trials, the expected number of times you have to repeat the experiment before you get a success is equal to 1/p. Use this fact to argue that the expected number of times we have to roll a 12-sided dices before we get a 6 or 12 as the outcome is equal to 6 throws. Describe your success and failure events and explain your calculation of expected number of throws (**4 points**)

Section 6 : Hash Functions

Question 10: Consider the hash function h which maps a 4-bit binary number into a 2-bit binary number by mapping a 4-bit binary string (b_3,b_2,b_1,b_0) to a binary string (a_1,a_0) by letting x = integer value of the binary number (b_3,b_2,b_1,b_0) obtained as b_3 x 2^3 + b_2 x 2^2 + b_1 x 2^1 + b_0 x 2^0 (that is x is a number between integer-value(0,0,0,0)=0 and integer-value of(1,1,1,1)=15) and setting (a_1,a_0) to be the 2-bit binary representation of the integer value y obtained as y = x (mod 4). That is our hash function is h(x) = x (mod 4), but represent x and y in binary. Find the hash value of binary strings (1,1,0,0) , (0,0,0,0), (1,1,0,1) and (1,0,0,0). **(2 points)**

Question 11: Argue that the above hash function only depends on the least significant two bits (b_1,b_0) of the binary number (b_3,b_2,b_1,b_0). **(4 points)**
Hint : For any three integers a,b: a + b (mod 4) = (a (mod 4) + b (mod 4)) (mod 4)

Question 7

$A = [21, 6, 7, 5, 14, 11, 10, 81, 9, 10, 21, 22, 23, 24, 25]$

Block 1 | Block 2 | Block 3

m_1 = Median(Block 1) = 5, 7, [8], 14, 21 m_2 = Median(Block 2) = 9, 10, [10], 11, 81

m_3 = Median(Block 3) = 21, 22, (23), 24, 25

$m^* = \text{Median}(m_1, m_2, m_3) = \text{Median}(8, [10], 23) = 10$

Since 10 is median of m_1, m_2, m_3 at least 2 of these medians are less than ≤ 10, (8, 10). In their respective Blocks (Block 1 & Block 2) there are 3 element which are $\leq$ their median (9, 7, 8 from Block 1, 9, 10, 10 from Block 2)

- Thus there are at least 6 elements $\leq m^*$ in A, and hence at most $15 - 6 = 9$ elements are greater than m^*.

Similiary there are 3 elements each from 2 blocks. whose medians are $\geq m^*$ (Block 2 & Block 3) which are greater than m^* (10, 11, 81 from Block 2, 23, 24, 25 from Block 3).

Question 9 The probability of getting 12/6 in the roll of a 12 sided unbiased dice is $= \frac{2}{12}$ (12 outcomes, 2 of which lead to Success).

Thus we can think of it as a random experiment where probability of success $= p = \frac{1}{6}$. Thus by the given fact expected # of tries before you get a 12/6 is $\frac{1}{p} = \frac{1}{1/6} = 6$. ∎

Question 8: Let X denote the random variable whose value is equal to the runtime of the given algorithm A. Since the runtimes are always non negative we can apply Markovs inequality to X.

We are given that $\mathbb{E}[X] = O(n^c) \leq c_1 \cdot n^c$ for some constant c_1

$$\Pr[X \geq 2^n] = \Pr\left[X \geq \frac{2^n}{c_1 n^c} \times c_1 n^c\right]$$

$$\Pr\left[X \geq \frac{2^n}{c_1 n^c} \mathbb{E}[X]\right] \leq \frac{1}{2^n / c_1 \cdot n^c} = \frac{c_1 \cdot n^c}{2^n}$$

This probability is exponentially small in n.

Question 10:

$h(x) = a \bmod 4$

$x = 1100$	$int(x) = 12$	$h(x) = 12 \pmod 4 = 0$
$x = 0000$	$int(x) = 0$	$h(x) = 0 \pmod 4 = 0$
$x = 1101$	$int(x) = 13$	$h(x) = 13 \pmod 4 = 1$
$x = 1000$	$int(x) = 8$	$h(x) = 8 \pmod 4 = 0$

Question 11

Integer value of binary number (b_3, b_2, b_1, b_0)

$$int(a) = b_3 \times 8 + b_2 \times 4 + b_1 \times 2 + b_0 \times 1$$

$$int(x) \bmod 4 = (b_3 \times 8 + b_2 \times 4 + b_1 \times 2 + b_0 \times 1) \bmod 4$$

$[(a+b) \bmod 4 = (a \pmod 4) + b \pmod 4) \bmod 4]$

$$= (b_3 \times 8 \pmod 4 + b_2 \times 4 \pmod 4 + b_1 \times 2 \pmod 4 + b_0 \times 1 \pmod 4) \pmod 4$$

$$= (0 + 0 + (b_1 \times 2) \bmod 4 + b_0 \times 2 \pmod 4) \pmod 4$$

[$8 \cdot b_3$ gives remainder 0 when divided by 4 as $8 \cdot b_3 = 4 \times 2 \times b_3$

and similarly
$4 \times b_2 \pmod 4 = 0$]

$$= (b_1 \times 2 \bmod 4 + b_0 \times 2 \bmod 4) \pmod 4$$

Thus the value of the hash function only depends on the last two bits b_0, b_1