DEPARTMENT OF CYBER SECURITY

Design and Analysis of Algorithm HW 6

SUBMITTED BY	211042 Noman Masood Khan A
SEMESTER	Fifth
SUBJECT	DAA
SUBMITTED TO	Sir Ammar Masood

15.2-1

The greedy-choice algorithm for the fractional knapsack problem involves the thief grabbing the highest value per weight object in the knapsack until the supply of that item is exhausted, and then begins grabbing the second highest value per weight, and so on. Due to the structure of this problem, there is only one sub-problem to solve (which is the next highest value per weight item), and is greedy by definition (what is the most valuable item per weight currently available at this moment in time). So, by definition, this has the greedy-choice property because the greedy algorithm generates the optimal solution.

15.2-7

❖ Solution Algorithm

Greedy Algorithm: Sort the set A and B so that the elements of the set A are a1>a2>a3…. a_n and elements of set B are b1>b2>b3….bn. Now pair ai with bi.

❖ Proof

Consider S as an optimal solution in which a_1 is paired with b_p and a_q is paired with b_1. In this solution, $a_1 > a_q$ and $b_1 > b_p$.

Consider another S` in which a_1 is paired with b_1 and a_q is paired with b_p and all other pairs are same as the S.

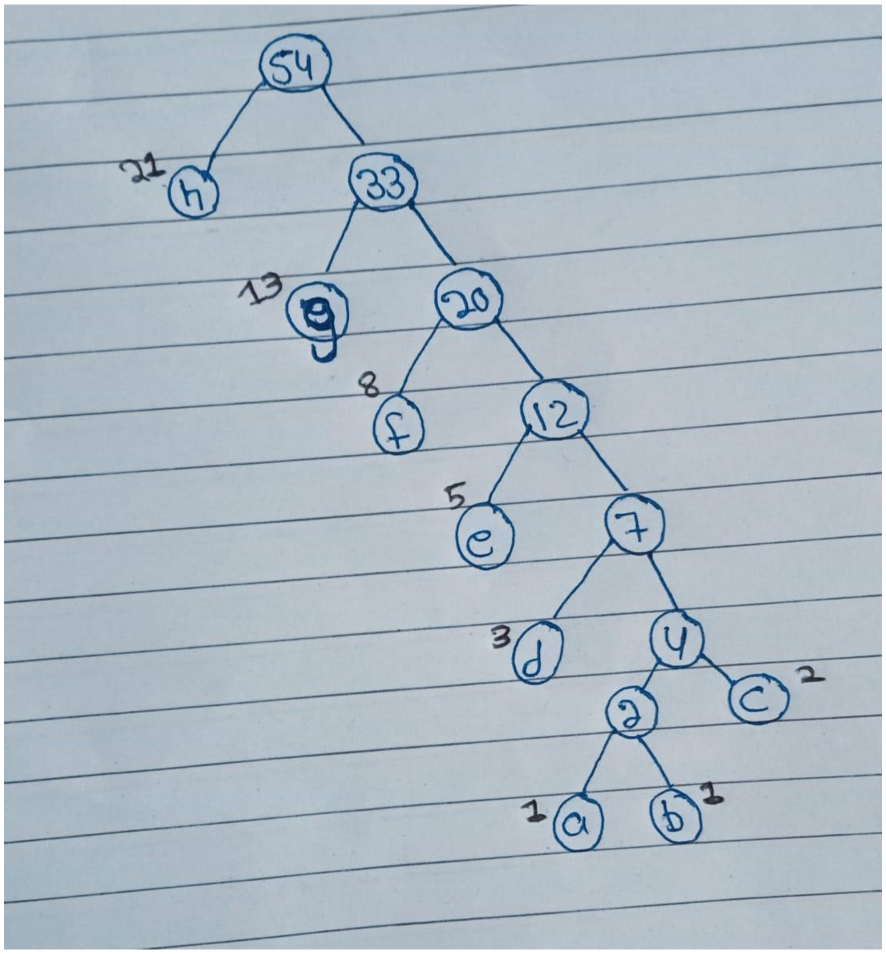
Then,

$$\frac{PayOff(S)}{Payoff(S`)} = \frac{\prod_S ai\text{^}bi}{\prod_{S`} ai\text{^}bi} = \frac{(a1)^{bp}.(a_q)^{b1}}{a1^{b1}.(a_q)^{bp}} = (\frac{a_1}{a_q})^{bp-b1}.$$

Since $a_1 > a_q$ and $b_1 > b_p$, then Payoff(S) / Payoff(S`) < 1.

- This contradicts the assumption that S is the optimal solution.

15.3-3



Huffman Code

Characters	Codes
a	1111100
b	1111101
c	111111
d	11110
e	1110
f	110
g	10
h	0

20.1-1

- Out-Degree Computation for all vertex

  Θ (V + E), Where V is the vertex and E is an edge.

In adjacency list for the directed graph, all the vertices which are out degree of the vertex are connected with the vertex and a list builds up. So, when counting the out-degree, we iterate through the vertices V of the list and for each vertex we iterate through the edges connect in the list and count them. This process is carried out for each vertex V and their respective Edges E.
So, our iteration depends on the vertex and edges of the graph in adjacency list.

- In-Degree Computation for all vertex

  Θ (V + E), Where V are vertices and E are Edges.

Adjacency list is not built up on the In-degree. Therefore, counting In-degree uses different method. The method is to count the frequency or count the number of times it appears in the adjacency list. For that we have to iterate over each vertex of the graph with vertex V and then for each vertex iterate till end over its adjacency list. In this case, we are iterating over all the vertices and then all the connected vertices which can be represented as edges E. So, the total time will be the sum of the total vertices and their edges E.

20.2-2

v.d is the minimum distance of vertex from the source vertex. Values were calculated on the rough page and the result are merely written here.

d values from the BFS search on the figure 20.3 are :

Vertices	r	s	t	u	v	w	x	y	z
Distance	2	1	1	0	2	3	2	1	3

v.π is the predecessor of the vertex in the graph.

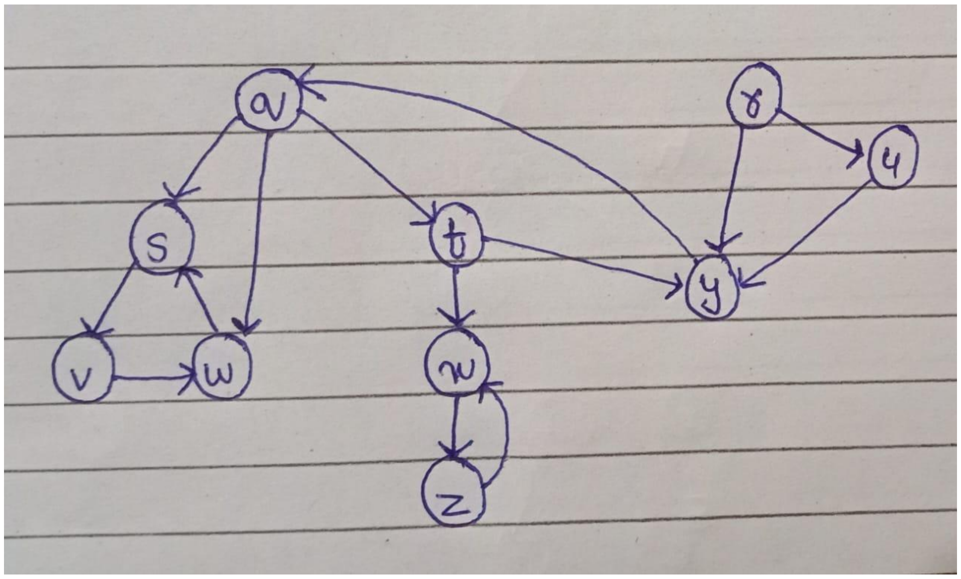
Vertices	r	s	t	u	v	w	x	y	z
Parent	s	u	u	NIL	s	r	y	u	x

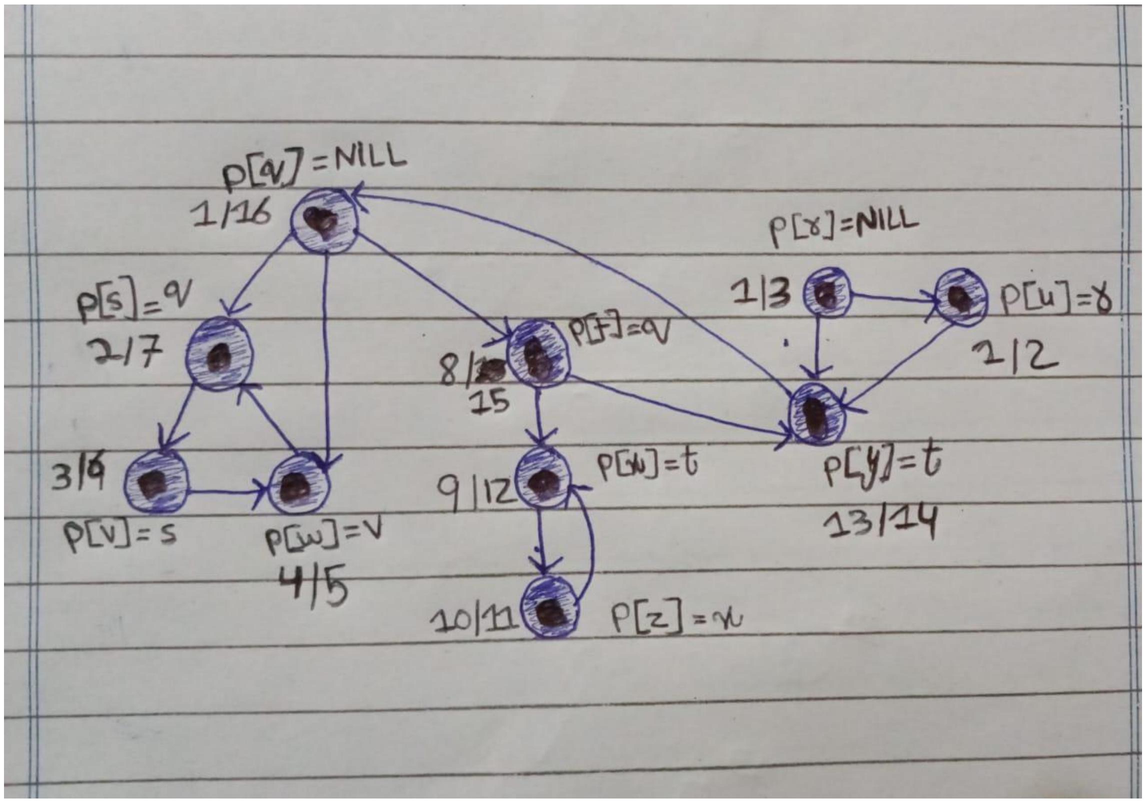
20.2-4

Adjacency matrix contains rows and columns where rows = number of vertices and columns = number of vertices also. When the vertices are connected, we have 1 in the matrix else 0 if no edge is between the vertices.
When iterating over the matrix, loop will iterate complete rows and columns of the matrix, which is only be possible with the nested loops. The program will be taking one index from row and checking its connected vertex in the column. As we know that the nested loop's running time is O(n^2). Therefore, BFS graph running time when represented as adjacency matrix will also be $O(|V|^2)$.

20.3-2

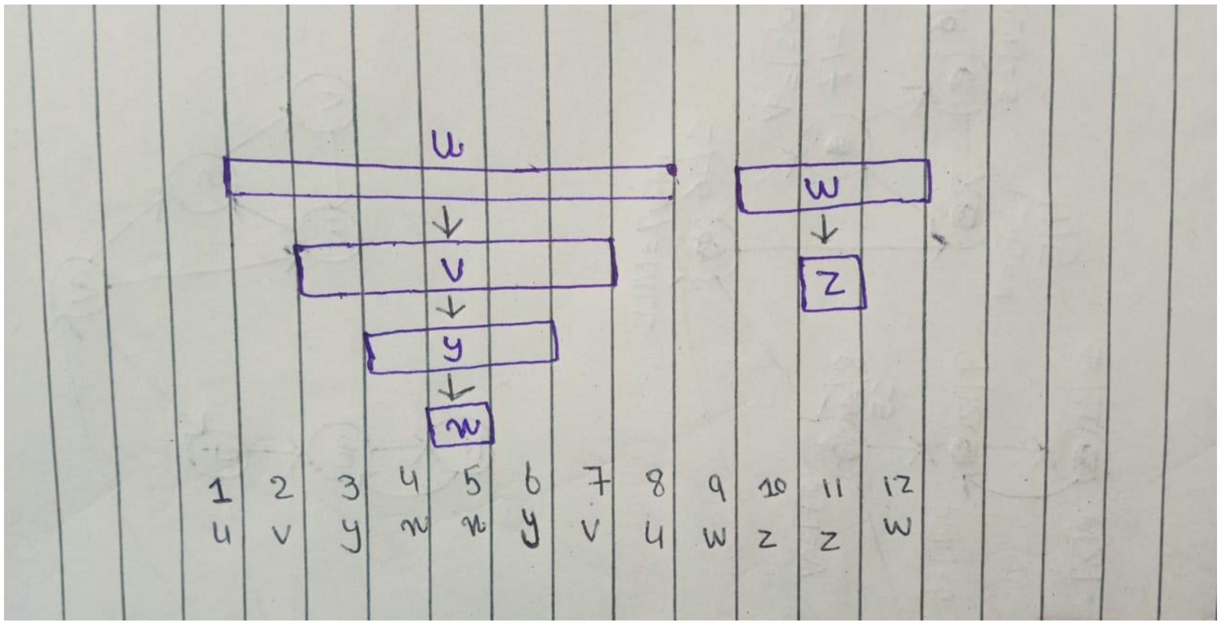
Since the DFS algorithm is considering the vertices in the alphabetic order, so the DFS loop is starting on q.





The for loop was run in the alphabetic order as well as the adjacency list was also considered to be in the alphabetic order.

20.3-3



20.4-1

After performing the DFS over the graph, following start and finish time was identified from the graph for each vertex

Vertex	Start time	Finish Time
m	1	20
q	2	5
t	3	4
r	6	19
u	7	8
y	9	18
v	10	17
w	11	14
z	12	13
x	15	16
n	21	26
o	22	25
s	23	24
p	27	28

Now for the topological sorting, we place vertex with the decreasing finish time. This will sort all the vertexes.

p n o s m r y v x w z u q t.

20.4-3

- Algorithm for determining the Cycles in the undirected graph.

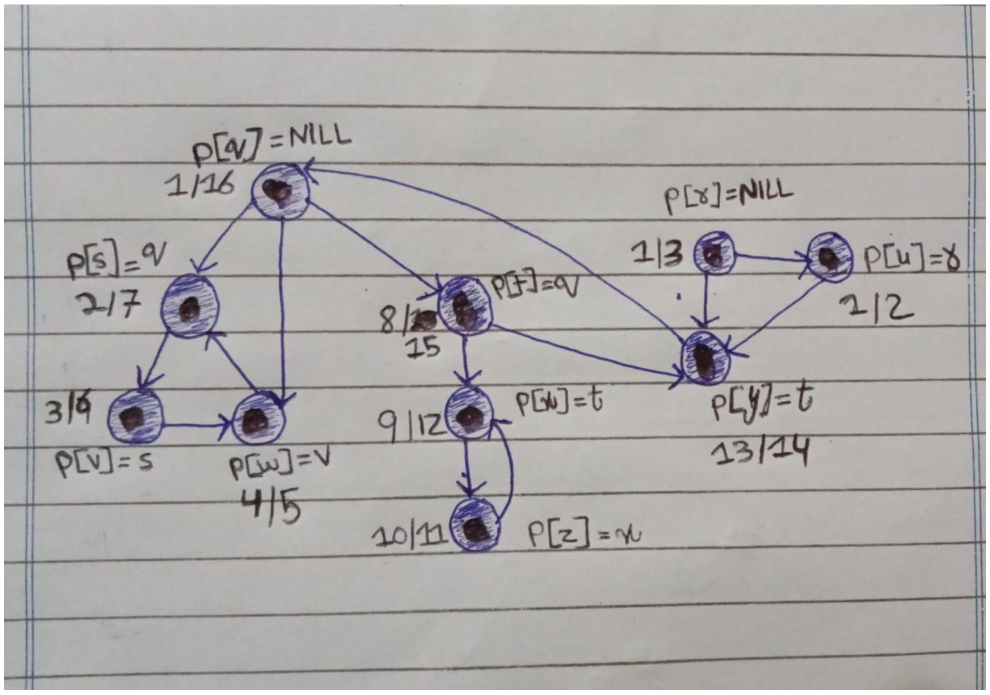
To determine that the graph contains cycles in it, we need to check whether the graph contains the back edges in it. If it contains back edges than it is cyclic else it is acyclic.

Step 1: Run the DFS algorithm over the graph G(V,E).
Step 2: Find a back edge.
Step 3: If it contains back edges, return. Print Graph is cyclic
Step 4: else the graph is acyclic.

The complexity is O(V) instead of O(E + V). Since if there is a back edge, it must be found before seeing |V | distinct edges. This is because in a acyclic (undirected) forest, |E| ≤ |V | - 1.

20.5-2

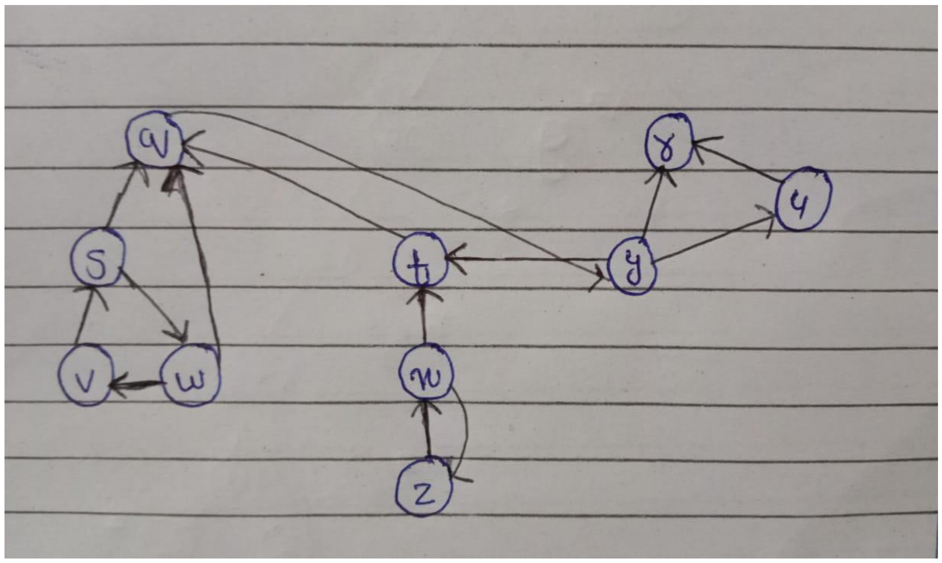
The DFS over the graph is :



- Finish Time for the Each Vertex

`	q	R	s	t	u	v	W	x	y	Z
Start	1	1	2	8	1	3	4	9	13	10
Finish	16	3	7	15	2	6	5	12	14	11

- Transpose of the Graph



- Finishing time after the transpose of the graph

	q	r	s	t	u	v	w	x	y	z
d	5	1	15	7	3	17	16	11	6	12
f	10	2	20	8	4	18	19	14	9	13

- Vertices of Trees are : {r}, {u}, {q, y, t}, {x, z}, {s, w, v}