1) Let S = {a, b, c, d, e, f, g} be a collection of objects with benefit-weight values, a: (12,5), b: (10,4), c: (8,5), d: (11,6), e: (14,3), f: (7,1), g: (9,6). What is an optimal solution to the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight 18? Show your work.

⇨ Given S = {a,b,c,d,e,f,g}

a: (12,5),
b: (10,4),
c: (8,5),
d: (11,6),
e: (14,3),
f: (7,1),
g: (9,6).

⇨ Solving the functional knapsack problem for S by Benefit/Weight ratio for every object :-

a: (12/5) = 2.4
b: (10/4) = 2.5
c: (8/5) = 1.6
d: (11/6) = 1.8333
e: (14/3) = 4.666
f: (7/1) = 7
g: (9/6)= 1.5

⇨ Arranging in decreasing order

7, 4.66667, 2.5, 2.4, 1.83333, 1.6, 1.5

⇨ Total Remaining =18

1) Profit = 7 , Weight = 18-1 = 17
2) Profit = 7+14 = 21 , Weight = 17-3 = 14
3) Profit = 21+10 = 31 , Weight = 14-4 = 10
4) Profit = 31+12 = 43 , Weight = 10-5 = 5

⇨ Remaining weight = 5

Including d(11/6) where 5/6

⇨ Total optimal solution will be 43 + 11* 5/6 = 313/6 = 52.1666667

2) Provide an example instance of the fractional knapsack problem where a greedy strategy based on repeatedly choosing as much of the smallest-weight item as possible results in a suboptimal solution.

- ⇨ The fractional knapsack problem all the items must be sorted in an organized manner to get the best optimization. By finding the smallest – weight items as possible.
- ⇨ Given example can show the show the knapsack problem.
- ⇨ Capacity of the bag = 25

Item	Weight	Benefit
A	10	55
B	4	32
C	15	47

This is the knapsack Weight/Benefit –

Taking the first A Weight 10 and benefit 55 then.

Weight =10+4 =14

Total Weight = 25-14 = 11

The knapsack for optimal case :-

55 +32 +(47/15) *11 = 1822/15 = 121.4666667

- ⇨ Greedy Approach (selects the minimum ones first)
  Item-> 10 , 16, 4
  55+47=102
- ⇨ Comparing both the approach the minimum is 121.466667 for the knapsack problem.

3) Suppose you are given an instance of the fractional knapsack problem in which all the items have an equal weight of 3. Show that you can solve the fractional knapsack problem in this case in O(n) time.

For the given instance if we have n items of equal weights 3 then in (number of profits) Pn

The time complexity when we sort the Profits/Weights ration list for a knapsack problem is (nlogn)

The time complexity of)(n) can only be implemented if we use selection algorithm where we solve it in a linear time.

By solving the Profit/ Weight we get (n/3) elements - given in the question.

Grouping them which will contain remaining mod 3 elements of the list

Then the total of (n/3) + 1 of the list groups will be considered.

If we have 3 sets of items then.

[Pi/Wi > m] = A

[Pi/Wi = m] = B

[Pi/Wi < m] = C

Where m in the median to calculate the rations of n number.

Then we evaluate Wa= Sigma i belongs A Wi

Wb= Sigma i belongs B Wi

Where the weight A > W then use recursion on the following set of the items and compare from the given capacity of the bag.

If else Weight A< W consider every item from the set A and from that subtract the remaining capacity of the bag.

Compare WA + WB > W that will have a remaining of w-Wa in the bag recursively and return the bag.

There are just if else comparitions in the algorithsm which O(1) time Complexity.

There for O(n) will be functional knapsack problem

4) Alena says that she ran the Huffman coding algorithm for the four characters, A, N, G, and Z, and it gave her the code words, 0, 10, 101, 110, respectively. Give examples of four frequencies for these characters that could have resulted in these code words or argue why these code words could not possibly have been output by the Huffman coding algorithm.

Char	Frequencies
A	20
N	15
G	9
Z	4

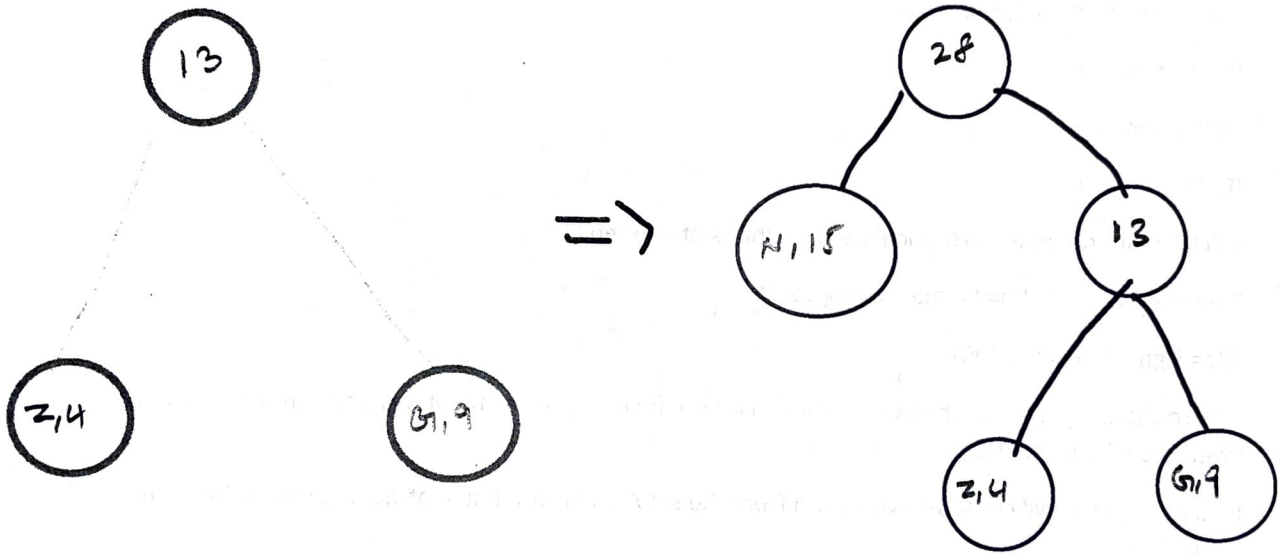
Constructing the following

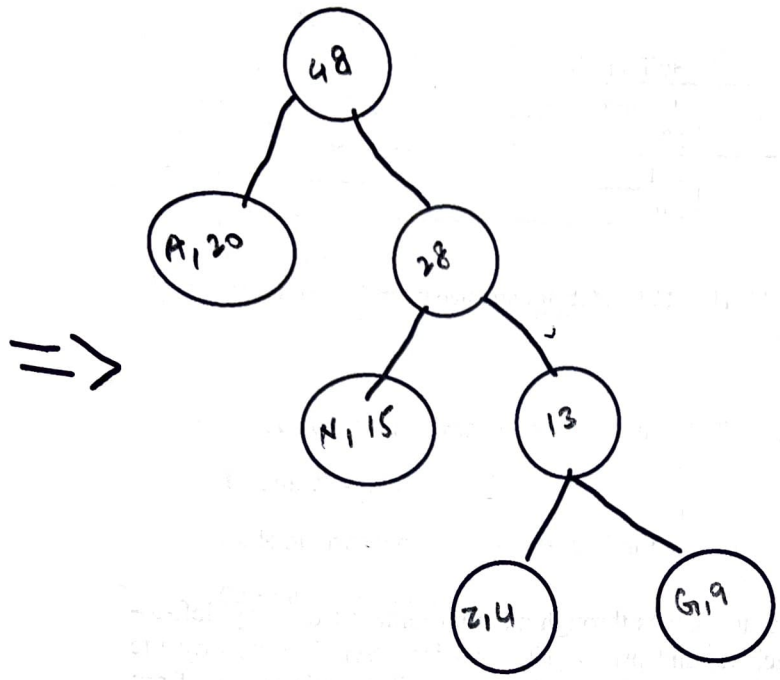
1)Taking the lowest 2 and adding them to the tree

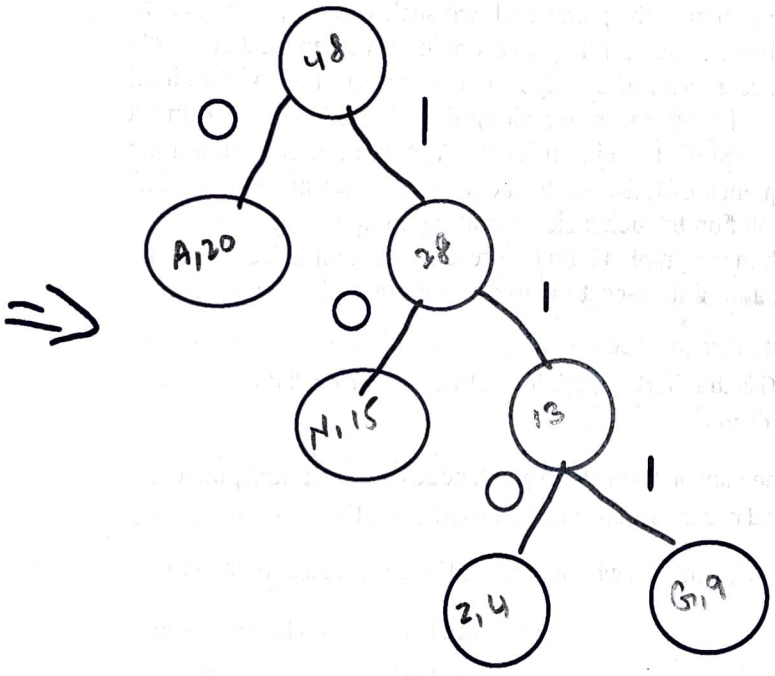
2) Pick the next from the lowest among them

3)Picking the next which is A, which has the highest freq among anybody

4) Apply the values 0 to all the left nodes and 1 to all the right nodes




48
A,20
28
N,15
13
Z,4
G,9


48
0
1
A,20
28
0
1
N,15
13
0
1
Z,4
G,9

The table will have the new codes

Char	Huffman Code
A	0
N	10
G	111
Z	110

All the codes match will the codes expect G which has 111 and it should have been 101. So G can never be coded in 101.

Therefore Alena is wrong [X]

5) In a side-scrolling video game, a character moves through an environment from, say, left-to-right, while encountering obstacles, attackers, and prizes. The goal is to avoid or destroy the obstacles, defeat or avoid the attackers, and collect as many prizes as possible while moving from a starting position to an ending position. We can model such a game with a graph, G, where each vertex is a game position, given as an (x, y) point in the plane, and two such vertices, v and w, are connected by an edge, given as a straight line segment, if there is a single movement that connects v and w. Furthermore, we can define the cost, c(e), of an edge to be a combination of the time, health points, prizes, etc., that it costs our character to move along the edge e (where earning a prize on this edge would be modeled as a negative term in this cost). A path, P, in G is monotone if traversing P involves a continuous sequence of left-to-right movements, with no right-to- left moves. Thus, we can model an optimal solution to such a side-scrolling computer game in terms of finding a minimum-cost monotone path in the graph, G, that represents this game. Describe and analyze an efficient algorithm for finding a minimum-cost monotone path in such a graph, G.

Minimum cost monotone path in the graph G is the shortest path forms to every vertex, If it is monotonic, every edge path will increase or decrease.

This can be solved by Dijkstra's algorithm. The minimization of the graph nodes is using priority queue. Where each vertex is pushed in the queue and never altered until it showed up on the top.

The algorithm designed will have the complexity O(nlogn) where n will be the edges assignment in the graph.

Algorithm

Current node= start state, Current distance = 0, Distance node = infinity

Leftmove = true

While(current node != finish node & distance < infinity)

{

For (every current node unvisited)

{ distance from one to the smallest node (current distance + distance between the first node and the next node)

}

Unvisited is visited now

Current = travel from visited node to other node and move to the shortest path from the current node

}

Minimum distance covered from leftmove = current. distance

Left move = false

Node of unvisited and distance to initial 0

Current node = start

Current visit & distance = final (0)

While(current node != finish node & distance < infinity)

{

For (every current node unvisited)

{ distance from one to the smallest node (current distance + distance between the first node and the next node)

}

Unvisited is visited now

Current = travel from visited node to other node and move to the shortest path from the current node

}

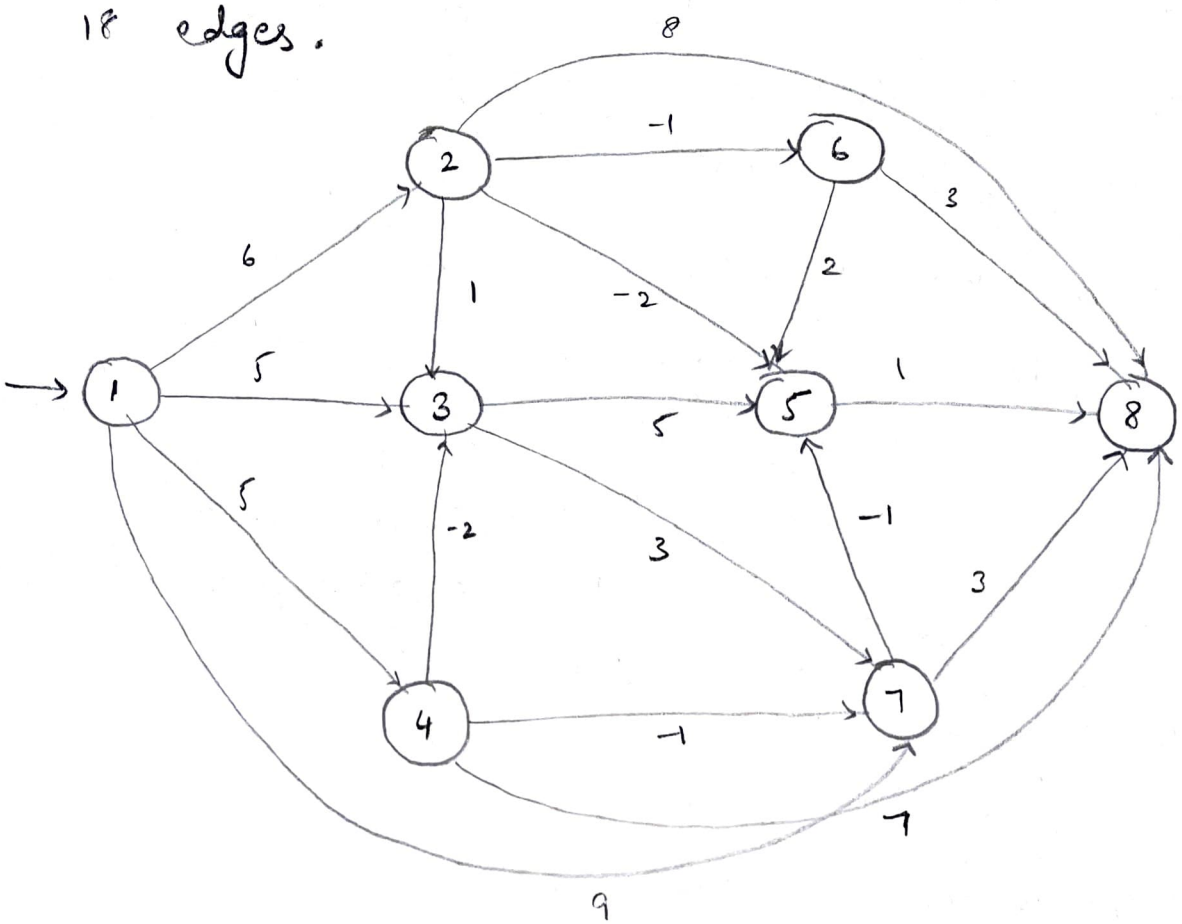
Right move = current.distance

Compare each node from leftmove to the right move and find the minimum distance covered.

Thus the model is an optimal solution to such a side scrolling computer game in terms of finding a minimum cost monotone path in the graph which Dijkstra's algorithm does

(reference Dijkstra's algorithms greekforgreeks)

6) A simple directed graph with 8 vertices and 18 edges.



Bell Ford Algorithm

Consider (u,v)

Relaxation if (d [u] + c (u,v) < d[v])

d [v] = d [u] + c (u,v)

Algorithm to Implement -

Begin

count := 1

max Edge := n*(n-1)/2 // n is number of vertices

for all vertices e of the graph, do

distance [e] := ∞

prediction := ?

done

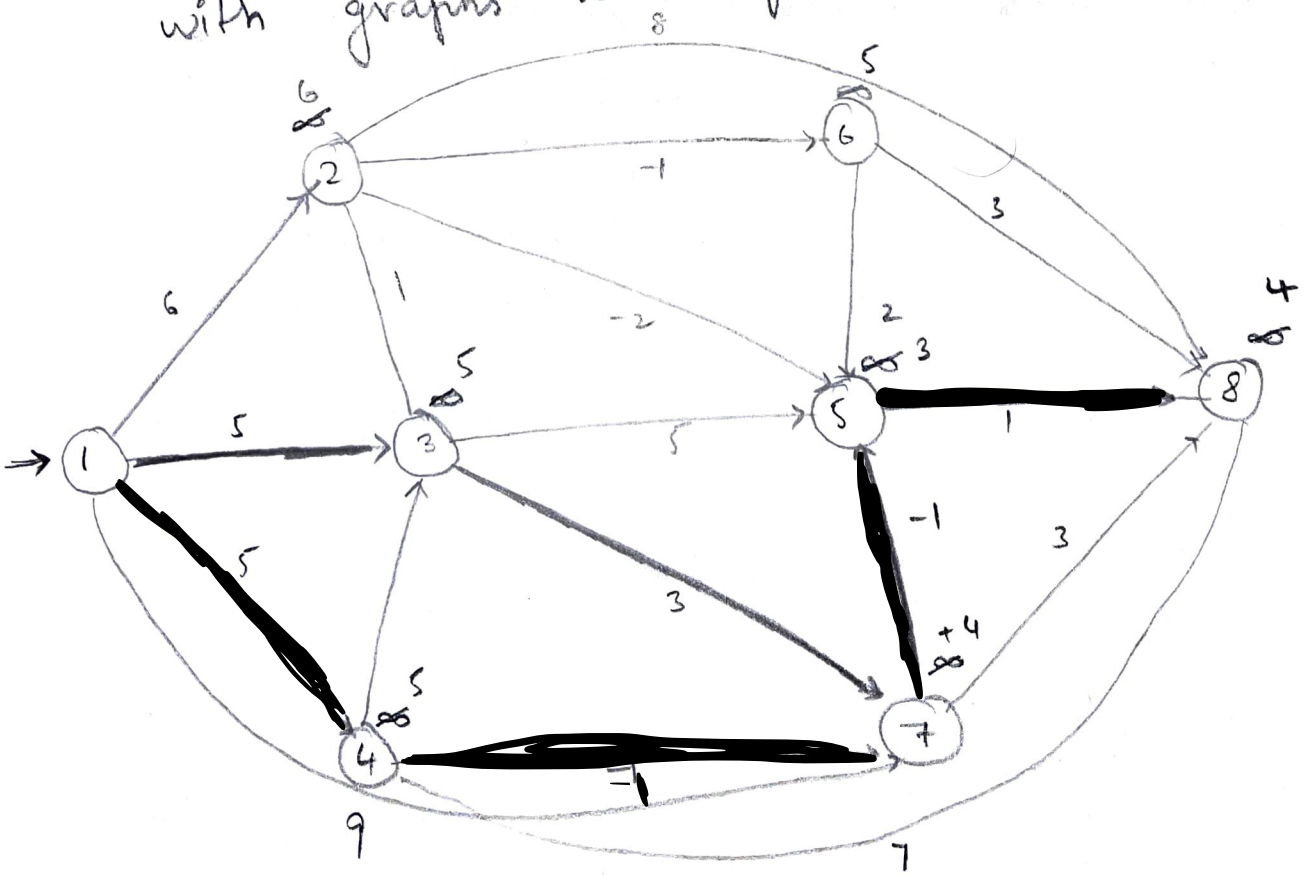
```
Distance [ initial ] := 0
   e count := number of edges

Create edge list as edgel

   while count < n, do

      for k := 0 to e count, do

        if distance [edgel [k]. e] > distance [edgel [k].m
                + (cost [m, e] for edge k) distance [edgel
                [k]. e] > distance [edgel [k].m] +
                (cost [m, e] for edge k) pred [edgel [k] .n]
                := edge l [k].m

        done

      done

  count := count +1
  for all vertices k in the graph, do
     if distance [edge l. [k]. e] > dist [edgel [k].m] +
           (cost [m, e] for edge k),
        then return true

  done

  return false

End.
```

→ Setting the distance to the source vertex to 0 and the distance to all other vertices to infinite. Creating an array of size e

→ Determine the shortest paths from the source vertex. Repeat this procedure N−1 time, where N is the number of vertices in the graph.

→ Even after N−1 of decreasing the path length for each vertex, if we can still relax the path length where distance[e] > distance[m] + weight of edge me, we may conclude that the graph has the negative edge cycle.

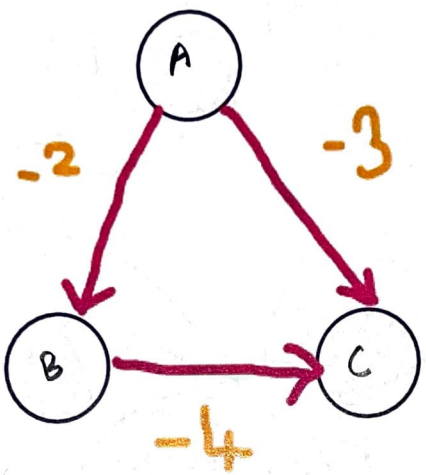
→ As a result, the bellman ford algorithm's main drawbacks is that it does not operate with graphs with negative edge cycles.



The shortest path after evaluating the algorithm we get [redacted] and the weight is 4 {1-4-7-5-8}

7) Give an example of a weighted directed graph, G , with negative-weight edges, but no negative-weight cycle, such that Dijkstra's algorithm incorrectly computes the shortest-path distances from some start vertex v.

Example :-



The Following graph has negative weights and has no cycle

Firstly the algorithm starts checking from the initial step where the node is unvisited ad visits the nodes.

So the distance from A->B is -2

the distance from A->C is -3

the distance from B->C is -2

Calculating the paths from A->B->C which is -6

Calculating the path from A->c which is -3

Where A->B->c path is is lesser compared to the other path

So Dijkstra's algorithm would not be able to find this and fail.

8) Describe the meaning of the graphical conventions used in Figures 15.9 and 15.10, illustrating Prim-Jarnik algorithm. What do thick lines and dashed lines signify?

Figure 15.10 is similar showing the remaining steps

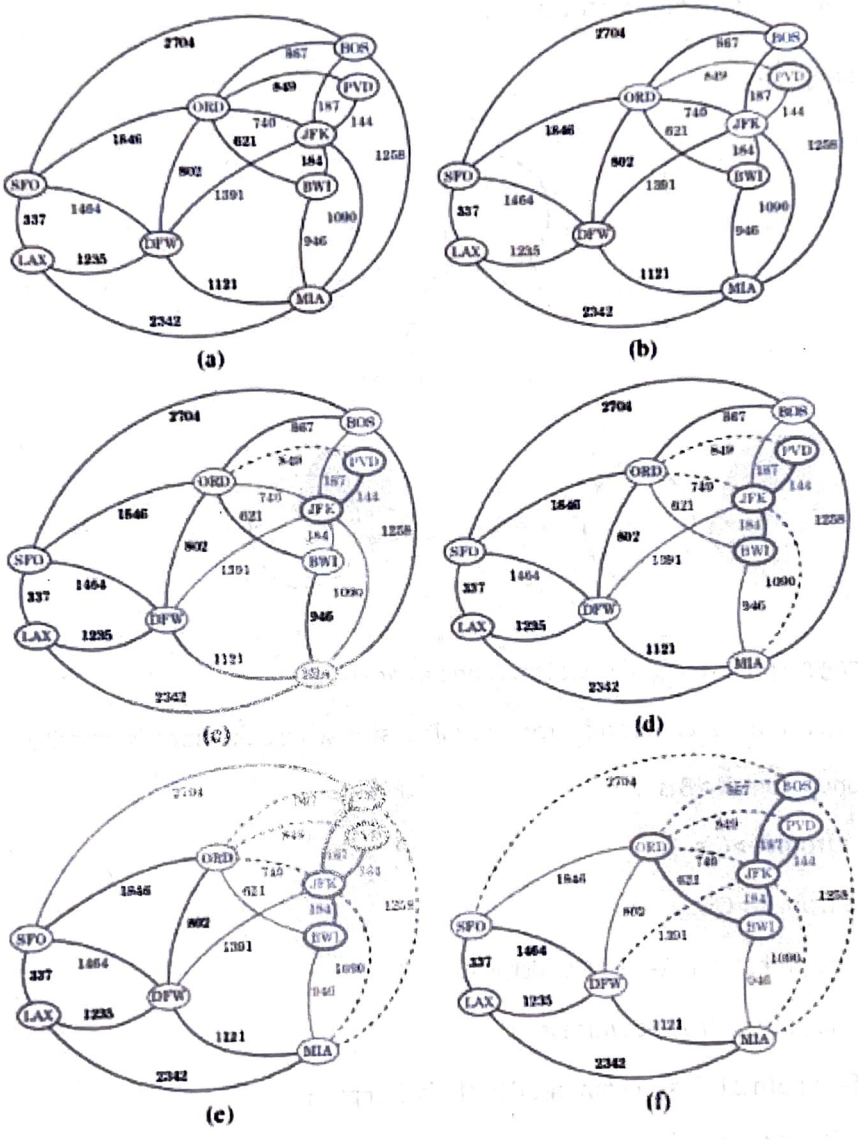


Figure 15.9: Visualizing the Prim-Jarník algorithm (continued in Figure 15.10).

A·8) Graph conventions are mathematical Pairwise relations between objects. A graph is made up of vertices which are connected by edge. A distinction is made between undirected graphs where edges link two vertices symmetrically and the directed graphs where edges link two vertices asymmetrically. A Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that include every vertex, at each step adding the cheapest possible connection from the tree to another vertex.

Algorithm :-

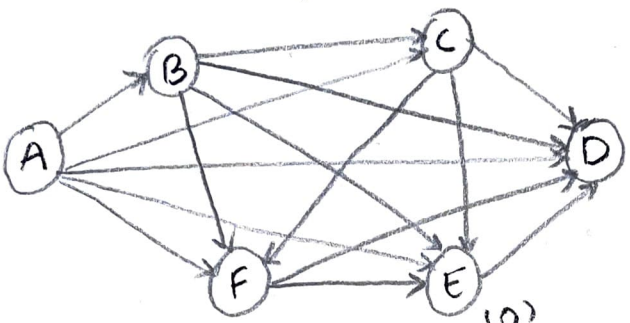
1) Create a set mst set that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as infinite. Assign key value as 0 for the first vertex so that it is picked first.
3) while mst set doesn't include all the vertices
   a) Pick a vertex u which is not there in mstSet and has minimum key value.
   b) include u to mst set.
   c) Update key value of all adjacent vertices of u. Every vertex, weight of edge u-v < prev key value v, then update key value as weight u-v.

→ So from the algorithm and seeing the figure 15.9 & 15.10 we see that from (a) to (b) we compare and find the smallest weight possible in the graph take the next one which is connected to the min cost edge. By that we visit the nodes in a tree like structure and dashed lines are forming a connected graph which has minimized weights forming a tree. We go on applying the algorithm in (c), (d), (e) and till (f) we get to know that the tree of the dashed lines should not form a loop or a circle and should be acyclic in every possible outcome. The thick lines are the edges which are not visited by the algorithm because they did not have the min cost while constructing a tree.

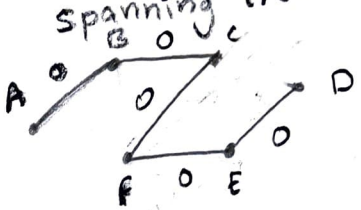
9) The total number of vertices in the graph is 6 which says that the maximum number of edges that we can compute is $\frac{n(n-1)}{2}$, where n is total number of vertices which is 6

$\therefore {}^{6}C_{2}$ or $\frac{6(6-1)}{2} = \frac{^{3}6 \times 5}{\not 2} = 15$

→ The maximum edges are 15. Given weight of the graph can be calculated as $|i-j-1|$ where $i >= 1$ & $j <= 6$ for the two vertices i and j.



spanning tree



→ The edges are respectively $(A \xrightarrow{(0)} B)$ $(A \xrightarrow{(1)} C)$ $(A \xrightarrow{(2)} F)$ $(A \xrightarrow{(3)} E)$ $(A \xrightarrow{(4)} D)$ $(B \xrightarrow{(0)} C)$ $(B \xrightarrow{(1)} D)$ $(B \xrightarrow{(2)} F)$ $(B \xrightarrow{(3)} E)$ $(C \xrightarrow{(0)} F)$ $(C \xrightarrow{(1)} E)$ $(C \xrightarrow{(2)} D)$ $(F \xrightarrow{(0)} E)$ $(F \xrightarrow{(1)} D)$ $(E \xrightarrow{(0)} D)$. with the weights above.

Kruskal's Algorithm for minimum cost spanning tree :-

Step 1 - Travers the graph 'G' and sort all the edges 'E' in increasing orders of their weight.

Step 2 - Sort orders of graph edges, remove the number of minimum weight and add it to the minimum spanning tree M in the same place as in the input graph G.

Step 3 - After inserting every edge check the new edge making the cycle the minimum spanning tree. If so remove the edge which is in the tree which means reject the edge.

Step 4 - Recursively repeat 2 & 3 untill every node is sorted and empty and the graph formed should not contain a cycle.

Step 5 - The tree which we get will be the minimum cost spanning tree M of given graph G.

Also - The minimum spanning tree consists of 6 vertices in the question where $V' = V$ & edges count will be less than the original edge $E' < E$

10) Describe an efficient greedy algorithm for making change for a specified value using a minimum number of coins, assuming there are four denominations of coins (called quarters, dimes, nickels, and pennies), with values 25, 10, 5, and 1, respectively. Argue why your algorithm is correct. Now, give an example set of denominations of coins so that a greedy change making algorithm will not use the minimum number of coins.

Given denominations – 25,10,5 and 1

So, greedy algorithms will always pick the decreasing order and the denominations are used accordingly

- ➔ An array implementation can be used containing the values of the denominations C[]
- ➔ Pick in decreasing order and sort it
- ➔ N number of variables that have the size of C [] array
- ➔ Initial count 0
- ➔ From the initial i-0 to n-1
- ➔ If the value <=0,
- ➔ If else set count = count + int (value / C[i})
- ➔ Set value = value % C[i]
- ➔ Return count

The time complexity is O(n) times where the denominations are run in the loop n number.

If the following array is run C[] = {25,10,5,1},

Then the value will be 40 by adding the sum .

Loop 1 :- i=0, value=40, count = 0

Value > 0,

Count = count + value/ C [0]

⇨ 0 + 40/25 = 1

Value = value % 25

⇨ 40 % 25 = 15

Loop 2: i=1, value = 15, count = 1

Value > 0,

Count = count + value/ C [1]

⇨ 1 + 15/10 = 2

Value = value % 10

⇨ 15 % 10 = 5

Loop 3: i=2, value = 5, count = 2

Value > 0,

Count = count + value/ C [2]

⇨ 2 + 5/5 = 3

Value = value % 5

⇨ 5 % 5 = 0

Loop 4: i=3, value = 0, count = 3

Value = 0,

Return count = 3

The value remaining is 3 as the value became null

Another Possibility can be where the denomination, C[] where the greedy algorithms cannot provide an optimized solution for every denomination

So, if the case Is when the value = 40 where 25+10+5 but there can also be 20+20 which is an optimized sum of using only 2 coins.

Therefore the algorithm of a dynamic approach can be

Algorithm DynamicCoin

- ➔ An array implementation can be used containing the values of the denominations C[]
- ➔ N number of variables that have the size of C [] array
- ➔ Create an array table [] of the size. Value +1 to calculate all the results
- ➔ Initial Loop i=0 starts
- ➔ Where table [i] = infinity
- ➔ Variable value i+1
- ➔ Set table value [0]
- ➔ Start loop from i=1. To the value
- ➔ Start j loop where j=0 to n-1
- ➔ If C[j]<=I,
- ➔ Then create the variable temp = table [i-C[j]
- ➔ Set the table value [i] = temp+1, if the value of temp +1< table [i]
- ➔ Then increase the value of the variable j+1
- ➔ Then increase the value of the variable i+1
- ➔

  Return table[value]

Therefore the algorithms stores all the different possible outcomes of the amount of the coin and then find the optimal solution for the sum of the coins.

References [Wikipedia , GeekforGeek]

I give permission to the instructor to share my solutions.

PLAGIARISM DECLARATION

1. I know that plagiarism means taking and using the ideas, writings, programs, code, works, or inventions of another as if they were one's own. I know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgement (which includes the proper use of quotation marks). I know that plagiarism covers this sort of use of material found in textual sources and from the Internet.

2. I acknowledge and understand that plagiarism is wrong.

3. This assignment is my own work, or my group's own unique group assignment. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.

4. I have not allowed, nor will I in the future allow, anyone to copy my work with the intention of passing it off as their own work.

Name..... Affan moyeed ..... Student #.....

Signed ..... Date ..... 03/29/22 .....