Greedy Method

General Method :

Greedy Method is a method, in this decision is taken based on the information available. It is used to find optimized solution.

- All these problems have n i/p's & require us to obtain a subset that satisfies some constraints
- Any subset that satisfies these constraints is called a Feasible solution.
- A feasible solution that either maximizes or minimizes a given objective function is called optimal solution.

Algorithm:

```
Algorithm Greedy (a,n)
{
    Solution := ϕ ;
    for i:=1 to n do
    {   x := Select(a);
        if feasible (Solution, x) then
        solution := union (solution, x)
    }
    return solution;
}
```

- The function selects selects an i/p from a[ ] & removes it.
- The selected i/p's value is assigned to x.
- Feasible is a Boolean - valued function that determines whether x can be included into the solution vector
- Function union combines x with the solution & updates the objective function.

Applications

1. Job Sequencing with deadlines
2. Knapsack problem
3. Minimum cost spanning Tree
4. Single Source Shortest path problem

Knapsack Problem:

The knapask (Fractional) problem is defined as:

- Given a list of n objects say $\{I_1, I_2 \ldots I_n\}$ & a knapsack (Bag).
- Capacity of knapsack is m.
- Each object I_i has a weight w_i & a profit of P_i.
- If a fraction x_i (where $x_i \in \{0, \ldots 1\}$) of an object I_i is placed into a knapsack then a profit of $P_i x_i$ is earned.

Mathematically:

$$\text{maximize} \sum_{1 \le i \le n} P_i x_i \qquad \text{— ①}$$

$$\text{subject to} \sum_{1 \le i \le n} w_i x_i \le w \qquad \text{— ②}$$

$$\text{and } 0 \le x_i \le 1, \quad 1 \le i \le n \qquad \text{— ③}$$

- A feasible solution is any set $(x_1 \ldots x_n)$ satisfying ② & ③ above
- An optimal solution is a feasible solution for which ① is maximized.

- The value of x_i is 1, if any object is completely placed into a knapsack
- If we do not pick that object then $x_i = 0$
- If we take a fraction of any object then its value will be b/w 0 & 1.

To solve this problem, Greedy method many apply any one of the following strategies:

- From the remaining objects, Select object with max profit that fit in knapsa
- " " " " " " that has min weight " "
- " " " " " " wit max P_i/w_i that " "

Eg: $n = 3$, $m = 20$, $(P_1, P_2, P_3) = (25, 24, 15)$ and $(w_1, w_2, w_3) = (18, 15, 10)$

Approach-1 (Decreasing order of profit):

In this we select object first with max profit & so on.

- We select 1st object since it has max profit (i.e. 25) & $w_1 = 18$
- After filling this object ($w_1 = 18$), remaining capacity is $20 - 18 = 2$
- we select 2nd object, but its weight is 15, so $x_2 = 2/15$
- Knapsack is full, so $x_3 = 0$.

Solution set = $(1, 2/15, 0)$

$\Sigma w_i x_i = (18 \times 1) + (2/15 \times 15) + (0 \times 10) = 20$

$\Sigma P_i x_i = (25 \times 1) + (2/15 \times 24) + (0 \times 15) = 28.2$

Approach-2 (Selection of order in increasing order of weights):

we select those object first which has minimum weight & so on.

- 1st we select 3rd objects ($w_3 = 10$) & $x_3 = 1$

so remaining capacity $= 20 - 10 = 10$

- 2nd we select 2nd object ($w_2 = 15$) but remaining capacity is 10,

  so $x_2 = 10/15 = 2/3$

- knapsack is full so ignore ~~3rd~~ 1st object

  So $(x_1, x_2, x_3) = (0, 2/3, 1)$

$\Sigma w_i x_i = (18 \times 0) + (2/3 \times 15) + (1 \times 10) = 20$

$\Sigma P_i x_i = (25 \times 0) + (2/3 \times 24) + (1 \times 15) = 31$

Approach-3: (Select of object in Decreasing order of ratio P_i/w_i):

we select objects with maximum P_i/w_i

$(P_1/w_1, P_2/w_2, P_3/w_3) = (25/18, 24/15, 15/10) = (1.3, 1.6, 1.5)$

- we select 2nd object first so $x_2 = 1$ & $w_2 = 15$

  remaining capacity $= 20 - 15 = 5$

- Next 3rd object is selected ($w_3 = 10$) but capacity is 5

  so $x_3 = 5/10 = 1/2$

- knapsack is full so $x_1 = 0$

$(x_1, x_2, x_3) = (0, 1, 1/2)$

$\Sigma w_i x_i = (18 \times 0) + (1 \times 15) + (1/2 \times 10) = 20$

$\Sigma P_i x_i = (25 \times 0) + (24 \times 1) + (15 \times 1/2) = 31.5$

So, finally

Approach	(x_1, x_2, x_3)	$\Sigma w_i x_i$	$\Sigma P_i x_i$
1	$(1, 2/15, 0)$	20	28.2
2	$(0, 2/3, 1)$	20	31.2
3	$(0, 1, 1/2)$	20	31.5 — optimal solution

So optimal solution is $(0, 1, 1/2)$ with Profit $= 31.5$

Eg-2 $n=7,\ m=15$ $(P_1, P_2, \ldots P_7) = (10, 5, 15, 7, 6, 18, 3)$

$(\omega_1, \omega_2 \ldots \omega_7) = (2, 3, 5, 7, 1, 4, 1)$

Approach-1 (Decreasing order of Profit):

we select those object with max profit then next & so on.

- First we select $n=6$ & $P_6 = 18,\ \omega_6 = 4,\ x_6 = 1$, remaining capacity $= 15-4 = 11$
- Next $n=3$, such that $P_3 = 15,\ \omega_3 = 5,\ x_3 = 1$, remaining capacity $= 11-5 = 6$
- Next $n=1$, such that $P_1 = 10,\ \omega_1 = 2,\ x_1 = 1$, " " $= 6-2 = 4$
- Next $n=4$, such that $P_4 = 7,\ \omega_4 = 7$ (can't fit since capacity is 4) so $x_4 = 4/7$

As knapsack is full, ignore remaining objects

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 0, 1, 4/7, 0, 1, 0)$

$\Sigma P_i x_i = (1\times10) + (0\times5) + (1\times15) + (4/7 \times 7) + (0\times6) + (1\times18) + (0\times3) = 47$

$\Sigma \omega_i x_i = 15$

Approach-2 (increasing order of weight):

we select objects in increasing order of weights

- First we select $n=5$, such that $\omega_5 = 1,\ x_5 = 1$, remaining capacity $= 15-1 = 14$
- Next we select $n=7$, such that $\omega_7 = 1,\ x_7 = 1$, remaining capacity $= 14-1 = 13$
- Next $n=1$, such that $\omega_1 = 2,\ x_1 = 1$, remaining capacity $= 13-2 = 11$
- Next $n=2$, such that $\omega_2 = 3,\ x_2 = 1$, " " $= 11-3 = 8$
- Next $n=6$, such that $\omega_6 = 4,\ x_6 = 1$, " " $= 8-4 = 4$
- Next $n=3$, such that $\omega_3 = 5$ (can't fit) so $x_3 = 4/5$
- ignore remaining objects

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 1, 4/5, 0, 1, 1, 1)$

$\Sigma \omega_i x_i = 15$

$\Sigma P_i x_i = (1\times10) + (5\times1) + (15\times 4/5) + (7\times0) + (1\times6) + (1\times18) + (1\times3) = 54$

Approach-3 (Decreasing order of P_i/ω_i):

$(P_1/\omega_1, P_2/\omega_2, P_3/\omega_3 \ldots P_7/\omega_7) = (5, 5/3, 3, 1, 6, 18/4, 3) = (5, 1.6, 3, 1, 6, 4.5, 3)$

- First we select $n=5$ so that $x_5 = 1$, remaining capacity $= 15-1 = 14$
- Next $n=1$, $\omega_1 = 2$, so that $x_1 = 1$, remaining capacity $= 14-2 = 12$
- Next $n=6$, $\omega_6 = 4$, so that $x_6 = 1$, remaining capacity $= 12-4 = 8$
- Next $n=3$, $\omega_3 = 5$ so that $x_3 = 1$, " " $= 8-5 = 3$
- Next $n=7$, $\omega_7 = 1$, so that $x_7 = 1$, " " $= 3-1 = 2$
- Next $n=2$, $\omega_2 = 3$ (can't fit) so $x_2 = 2/3$

$(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = (1, 2/3, 1, 0, 1, 1, 1)$

$\Sigma P_i x_i = (1\times10) + (2/3\times5) + (1\times15) + (0\times7) + (1\times6) + (1\times18) + (1\times3) = 55.33$

$\Sigma \omega_i x_i = 15$

So, finally

Approach	(x_1, x_2, x_3)	$\Sigma w_i x_i$	$\Sigma P_i x_i$
1	$(1,0,1,4/7,0,1,0)$	15	47
2	$(1,1,4/5,0,1,1,1)$	15	54
3	$(1,2/3,1,0,1,1,1)$	15	55.33 — optimal solution

The optimal solution is $(1, 2/3, 1, 0, 1, 1, 1)$ with profit = 55.33

Eg-3 $\omega = 60, \ n = 4$

$(\omega_1, \omega_2, \omega_3, \omega_4) = (40, 10, 20, 24)$ $(P_1, P_2, P_3, P_4) = (280, 100, 120, 120)$

Approach-1 (Dec order of profit):

- First select $n = 1, \ \omega_1 = 40, \ x_1 = 1,$ remaining capacity $= 60 - 40 = 20$
- Next $n = 3, \ \omega_3 = 20, \ x_3 = 1,$ remaining capacity $= 20 - 20 = 0$

$(x_1, x_2, x_3, x_4) = (1, 0, 1, 0)$

$\Sigma P_i x_i = 280 + 0 + 120 + 0 = 400$

$\Sigma \omega_i x_i = 60.$

Approach-2 (Increasing order of weights):

- First select $n = 2, \ \omega_2 = 100, \ x_2 = 1,$ remaining capacity $= 60 - 10 = 50$
- Next $n = 3, \ x_3 = 1, \ \omega_3 = 20,$ remaining $= 50 - 20 = 30$
- Next $n = 4, \ x_4 = 1, \ \omega_4 = 24,$ remaining $= 30 - 24 = 6$
- Next $n = 1, \ w_1 = 40$ (can't fit) so $x_1 = 6/40$

$(x_1, x_2, x_3, x_4) = (6/40, 1, 1, 1)$

$\Sigma P_i x_i = (6/40 \times 280) + 100 + 120 + 120 = 382$

$\Sigma \omega_i x_i = 60$

Approach-3 (Dec order of P_i/ω_i):

$(P_1/\omega_2, P_2/\omega_2, P_3/\omega_3, P_4/\omega_4) = (7, 10, 6, 5)$

- First select $n = 2, \ x_2 = 1, \ \omega_2 = 10,$ remaining $= 60 - 10 = 50$
- Next $n = 1, \ x_1 = 1, \ \omega_1 = 40,$ remaining $= 50 - 40 = 10$
- Next $n = 3, \ x_3 = 1, \ \omega_3 = 20$ (can't fit) $x_3 = 10/20 = 1/2$

$(x_1, x_2, x_3) = (1, 1, 1/2, 0)$

$\Sigma P_i x_i = 280 + 100 + (1/2 \times 120) + 0 = 440$ — optimal.

The optimal solution is $(1, 1, 1/2, 0)$ with profit 440

Algorithm:

```
Algorithm knapsack(M,n)
{
  for i:=1 to n do
    x[i]: =0;
    Profit: = weight: =0;
  while( weight <M)
  {
     if (weight + w[i] ≤M)
        x[i]:= 1;
        weight: = weight +w[i] ;
  else
      x[i] : = (M-weight)/ w[i] ;
      weight: = M;
  } }  Profit = Profit + P[i]* x[i];
          i++ ;
```

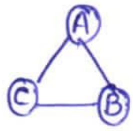
Time complexity:

- Sorting of n items in decreasing of the ratio P_i/w_i takes $O(n\log n)$ time. Since this is the lower bound for any sorting algorithm.
- Here while loop in the algorithm takes $O(n)$ time.
- Therefore total time including sort is $O(n)$.

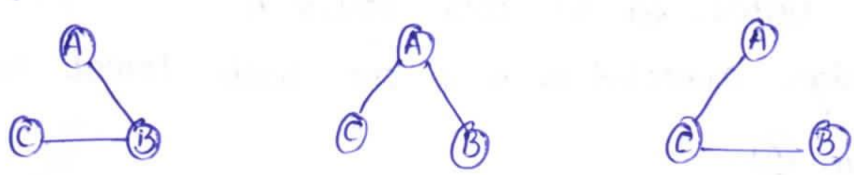
Spanning Tree:

Let $G=(V,E)$ be an undirected connected graph. A subgraph $T=(V,E')$ of G is a spanning Tree if & only if T is a Tree (i.e, no cycle exists in T) & contains all vertices of G.

Eg: Graph G_1,

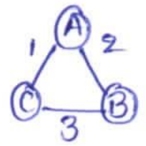


Spanning Tree is,

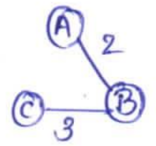


Minimum Cost Spanning Tree: A Minimum cost Spanning Tree (MCST) of a weighted connected graph G is that spanning whose sum of length (or weights) of all its edges is minimum, among all the possible spanning tree of G.

Eg: Graph G,

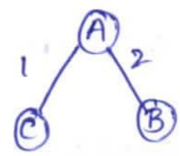


Spanning Tree,



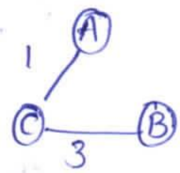
cost $= 2+3 = 5$

(i)



cost $= 1+2 = 3$

(ii)



cost $= 1+3 = 4$

(iii)

(ii) is MCST with cost $= 3$

To find MCST of a graph G, one of following algorithm is used:

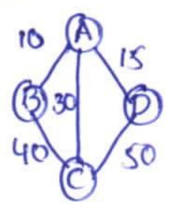
1. Prims Algorithm
2. Krushkals "

Prims Algorithm: In this method,

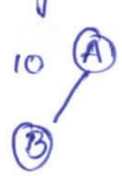
1. Choose any starting vertex. Look at all edges connecting to the vertex & choose the one with the lowest weight & add this to the tree
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keep repeating step 2 until we get MCST

- For a graph G, with n vertices, MCST will have $(n-1)$ edges.

Eg-1:

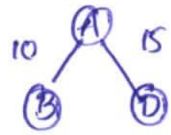


1. Choose any vertex. Let us take vertex A.
   Look at edges connected to A & one with lowest weight is $AB = 10$



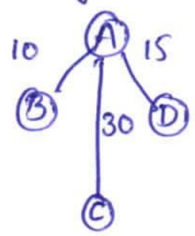
cost $= 10$.

2. Consider edges connected to vertices A 4 B. Minimum is 15 with edge AD.



cost = 10+15 = 25.

3. Next minimum edge weight is 30 (either AC, or CD)

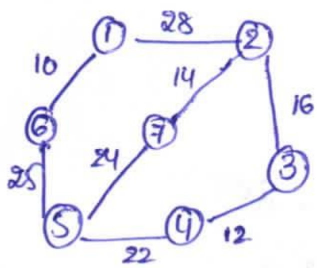


cost = 10+15+30 = 55.

The MCST is having (n-1) edges, so

Total weight = 55

Eg-2:

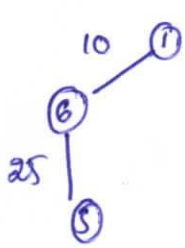


1. Choose any vertex. Let us take vertex 1.

look at edges connected to vertex 1 4 add minimum weight edges 1-6. with cost 10.

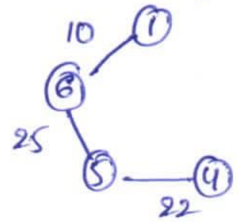
Total weight = 10

2. Next minimum edge weight is 25 for 6-5.



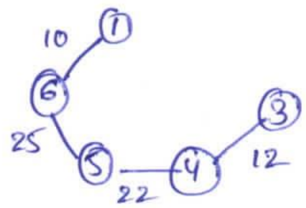
Total weight = 10+25 = 35.

3. Next minimum edge weight is 22 for 5-4



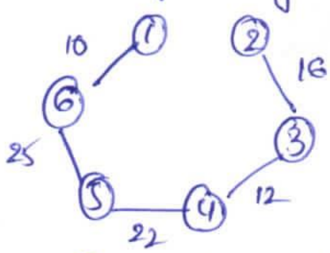
Total cost = 10+25+22 = 57

4. Next minimum edge weight among all vertices connected in Tree is 12 for 4-3



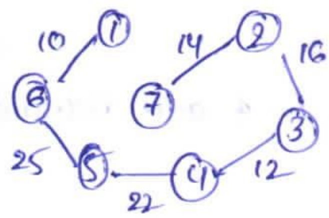
Total cost = 10+25+22+12 = 69

5. Next minimum edge weight is 16 for 3-2



Total cost = 10+25+22+12+16 = 85.

6. Next minimum edge weight is 14 for 2-7



The above is MCST with 6 edges &

Total cost = 10+25+22+12+16+14 = 99

Algorithm :

Algorithm Prims (E, cost, n, t)
{
let (k, l) be an edge of mincost in E;
mincost : = cost [k, l];
t[1,1] : = k;
t[1,2] : = l;

```
for i: = 1 to n do
  if ( cost [i, l] < cost [i, k] ) then
  {  near [i]: = l;
   else
  }  near [i]: = k;
     near [k] : = near[l]: = 0;
   for i: = 2 to n-1 do
{
  let j be an index such that near[j] ≠ 0 and cost[j, near[j]] is minimum;
    t[i,1] : = j ;
    t[i,2]: = near[j];
    mincost : = mincost + cost[j, near[j]];
     near[j]: = 0;
   for k: = 1 to n do
    if ((near[k] ≠ 0)) and (cost(k, near[k]) > cost [k, j]) then
    }     near[k] : = j;
          return mincost ;
    }
```

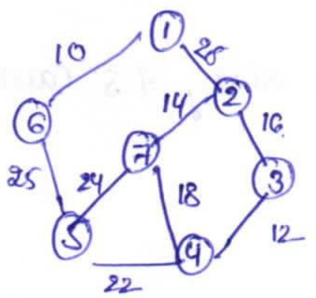
Analysis:

The timecomplexity for Prims algorithm is $O(n^2)$.

Krushkal's Algorithm: In this method,

1. Arrange the edges in their increasing order of weight
2. Add the edge which has least weight. It is not necessary that selected edge is adjacent.
3. Repeat step-2 until we get MCST.

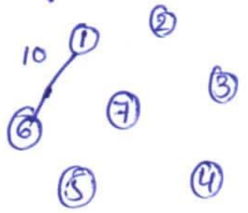
Eg-1:



Arrange edges in increasing order of weights.

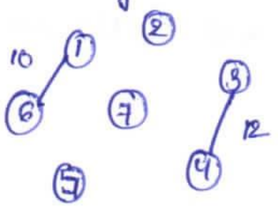
edge	1-6	3-4	2-7	2-3	7-4	4-5	5-7	5-6	1-2
weight	10	12	14	16	18	22	24	25	28

1. Add edge 1-6 to MCST



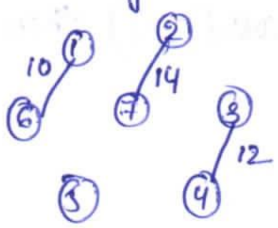
Total weight = 10.

2 Next add edge 3-4 with weight = 12



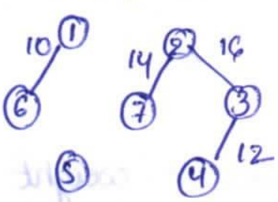
Total weight = 10+12 = 22

3. Next add edge 2-7 with weight 14



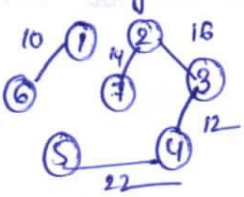
Total weight = 10+12+14 = 36

4. Next add edge 2-3 with weight 16.



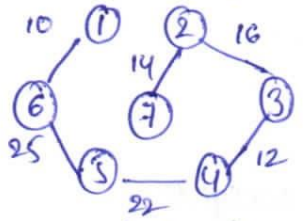
Total weight = 10+12+14+16 = 52

5. Next add edge 4-5 with weight 22 (As adding 4-7 causes cycle)



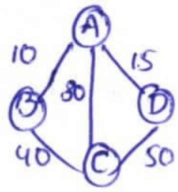
Total weight = 10+12+14+16+22 = 74.

6. Next add edge 5-6 with weight 25 (As adding 7-5 causes cycle)



Total weight = 10+12+14+16+22+25 = 99

Eg - 2 :

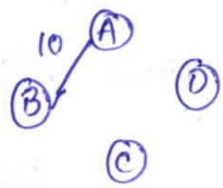


1 Arrange the edges in increasing order of weights.

edge	AB	AD	CA	BC	CD
weight	10	15	30	40	50

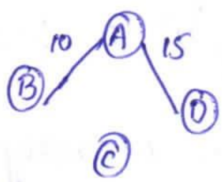
2. Initially,

Now add edge AB with weight 10.



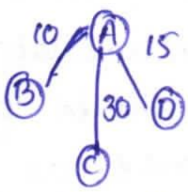
Total weight = 10

3. Now add edge AD with weight 15.



Total weight = 10+15 = 25

4. Now edge CA is to be added with weight 30



Above is MCST with (n-1) edges &

Total weight = 10+15+30 = 55

Algorithm;

```
Algorithm kruskal (E, cost, n, t)
{
    Construct a heap out of the edges;
    for i:=1 to n do parent[i] := -1;
    i:=0;
    mincost := 0.0;
```

```
while ((i<n-1) and (heap not empty)) do
{
    Delete a minimum cost edge (u,v) from heap & reheapify;
    j: =find(u);
    k: =find(v);
    if (j ≠ k) then
    {
        i: = i+1;
        t[i,1]: = u;
        t[i,2]: = v;
        mincost = mincost + cost[u,v];
        union(j, k);
    }
}
return mincost;
```

Analysis:

The computing time of krushkal's algorithm is $O(E \log n)$

Differences between krushcals & Prim's Algorithm:

Krushkal's Algorithm	Prim's Algorithm
1. Always selects an edge (u,v) of minimum weight to find MCST	1. Always selects a vertex (say v) to find MCST.
2. Not necessary to choose adjacent vertices for getting MCST	2. It is necessary to select an adjacent vertex to get MCST.
3. At intermediate step of algorithm, there are many be more than one connected components are possible	3. At intermediate step of algorithm, there will be only one connected components are possible
4. Time complexity $O(E \log n)$	4. Time complexity $O(n^2)$

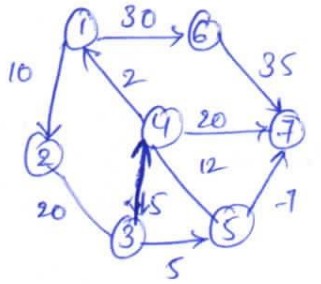
Single Source Shortest path Algorithm - Dijkstra's Algorithm:

In a single source shortest path problem the shortest distance from a single vertex called source & the last vertex is called Destination

In this algorithm,

1. Set the distance to source vertex as zero
2. Relax all vertices adjacent to current vertex.
3. Choose the closest vertex as current vertex
4. Repeat Step 2 & 3.

Eg:



1. We will start at vertex 1.

   Hense $S = \{1\}$

2. From 1,

   $d\{1,2\} = 10$
   $d\{1,6\} = 30$
   $d\{1,3\} = \infty$
   $d\{1,5\} = \infty$
   $d\{1,7\} = \infty$

3. Select closest vertex. So select vertex 2.

   Hence $S = \{1, 2\}$

4. Relax all vertices adjacent to current vertex.

   $d\{1,2,3\} = 30$
   $d\{1,2,4\} = \infty$
   $d\{1,2,5\} = \infty$
   $d\{1,2,7\} = \infty$
   $d\{1,2,6\} = \infty$

5. Hence $d = \{1, 2, 3\}$ is selected.

6. Relax all vertices adjacent to current vertex.

   $d\{1,2,3,4\} = 45$
   $d\{1,2,3,5\} = 35$
   $d\{1,2,3,6\} = \infty$
   $d\{1,2,3,7\} = \infty$

7 Hence d{1,2,3,5} is selected.

8. Now, d{1,2,3,5,7} = 42

d{1,2,3,5,6} = ∞

d{1,2,3,5,4} = ∞

∴ d{1,2,3,5,7} is selected

So single source shortest path is,

$1 \xrightarrow{10} 2 \xrightarrow{20} 3 \xrightarrow{5} 5 \xrightarrow{7} 7$

= 42

Analysis

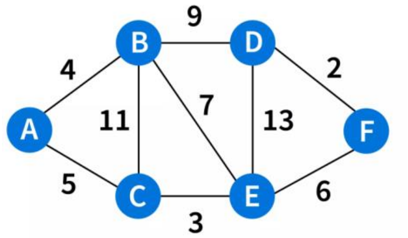
Running time is $O(n + |E| \log n)$

Algorithm :

```
Algorithm Shortestpath(v, cost, dist, n)
{
   for i:=1 to n do
   {  S[i]: =false;
   }  dist[i]: =cost[v,i];
     S[v] = true;
     dist[v]: =0.0;
   for num:= 2 to n-1 do
   {
     Choose u from those vertices not in S such that dist[u] is min;
     S[u]: = true;
     for (each w adjacent to u with S[w] = false) do
       if (dist[w] > dist[u] + cost [u,w])) then
         dist[w] : = dist[u] + cost[u,w];
   }
}
```

Dijkstra Algorithm Example

Lets take an example to understand the algortihm better.



Optimal Merge Patterns:

- Merge a set of sorted files of different length into a single sorted file. We need to find an optimal solution, where the resultant file will be generated in minimum time.
- If the number of sorted files are given, there are many ways to merge them into a single sorted file. This merge can be performed pair wise. Hence, this type of merging is called as **2-way merge patterns**.
- As, different pairings require different amounts of time, in this strategy we want to determine an optimal way of merging many files together. At each step, two shortest sequences are merged.
- To merge a **p-record file** and a **q-record file** requires possibly **p + q** record moves, the obvious choice being, merge the two smallest files together at each step.

Two-way merge patterns can be represented by binary merge trees. Let us consider a set of **n** sorted files {f_1, f_2, f_3, …, f_n}. Initially, each element of this is considered as a single node binary tree. To find this optimal solution, the following algorithm is used.

Algorithm: TREE (n)

```
for i := 1 to n – 1 do
   declare new node
   node.leftchild := least (list)
   node.rightchild := least (list)
   node.weight) := ((node.leftchild).weight) + ((node.rightchild).weight)
   insert (list, node);
return least (list);
```

At the end of this algorithm, the weight of the root node represents the optimal cost.

Example

Let us consider the given files, f_1, f_2, f_3, f_4 and f_5 with 20, 30, 10, 5 and 30 number of elements respectively.

If merge operations are performed according to the provided sequence, then

M_1 = merge f_1 and f_2 => 20 + 30 = 50

M_2 = merge M_1 and f_3 => 50 + 10 = 60

M_3 = merge M_2 and f_4 => 60 + 5 = 65

M_4 = merge M_3 and f_5 => 65 + 30 = 95

Hence, the total number of operations is

50 + 60 + 65 + 95 = 270

Now, the question arises is there any better solution?

Sorting the numbers according to their size in an ascending order, we get the following sequence –

f_4, f_3, f_1, f_2, f_5

Hence, merge operations can be performed on this sequence

M_1 = merge f_4 and f_3 => 5 + 10 = 15

M_2 = merge M_1 and f_1 => 15 + 20 = 35

M_3 = merge M_2 and f_2 => 35 + 30 = 65

M_4 = merge M_3 and f_5 => 65 + 30 = 95

Therefore, the total number of operations is

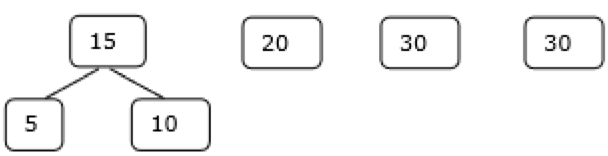
15 + 35 + 65 + 95 = 210

Obviously, this is better than the previous one.

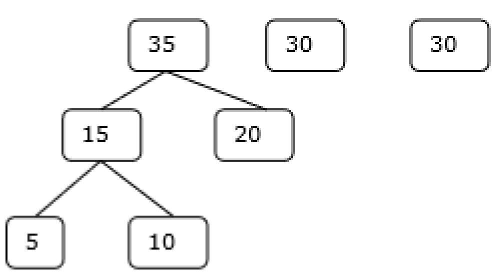
In this context, we are now going to solve the problem using this algorithm.

Initial Set

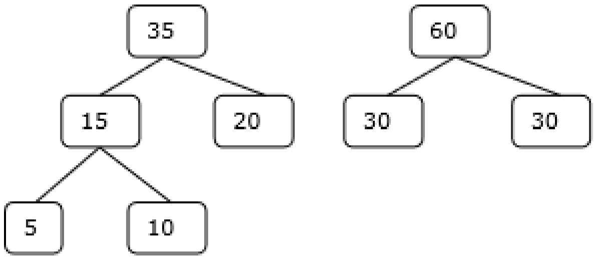
Step-1



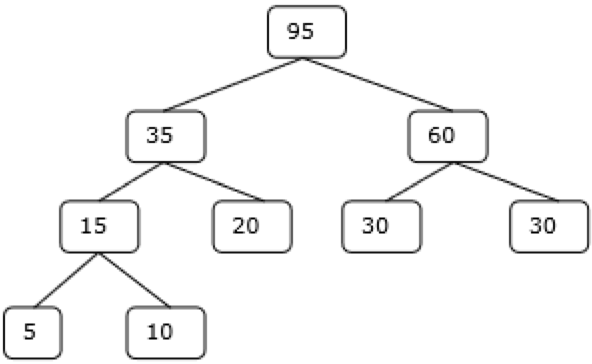
Step-2



Step-3



Step-4



Hence, the solution takes 15 + 35 + 60 + 95 = 205 number of comparisons.

Huffman Coding

It is also known as **data compression encoding.** It is widely used in image (JPEG or JPG) compression. In this section, we will discuss the **Huffman encoding** and **decoding,** and also implement its algorithm.

We know that each character is a sequence of 0's and 1's and stores using 8-bits

variable-length encoding : we exploit some characters that occur more frequently in comparison to other characters. In this encoding technique, we can represent the same piece of text or string by reducing the number of bits.

Huffman Encoding

Huffman encoding implements the following steps.

- It assigns a variable-length code to all the given characters.
- The code length of a character depends on how frequently it occurs in the given text or string.
- A character gets the smallest code if it frequently occurs.
- A character gets the largest code if it least occurs.

There are the following two major steps involved in Huffman coding:

- First, construct a **Huffman tree** from the given input string or characters or text.
- Assign, a Huffman code to each character by traversing over the tree.

Let's brief the above two steps.

Huffman Tree

Step 1: For each character of the node, create a leaf node. The leaf node of a character contains the frequency of that character.

Step 2: Set all the nodes in sorted order according to their frequency.

Step 3: There may exist a condition in which two nodes may have the same frequency. In such a case, do the following:

1. Create a new internal node.
2. The frequency of the node will be the sum of the frequency of those two nodes that have the same frequency.
3. Mark the first node as the left child and another node as the right child of the newly created internal node.

Step 4: Repeat step 2 and 3 until all the node forms a single tree. Thus, we get a Huffman tree.

Huffman Encoding Example

Suppose, we have to encode string **abracadabra.** Determine the following:

i. Huffman code for All the characters
ii. Average code length for the given String
iii. Length of the encoded string

(i) Huffman Code for All the Characters

In order to determine the code for each character, first, we construct a **Huffman tree.**

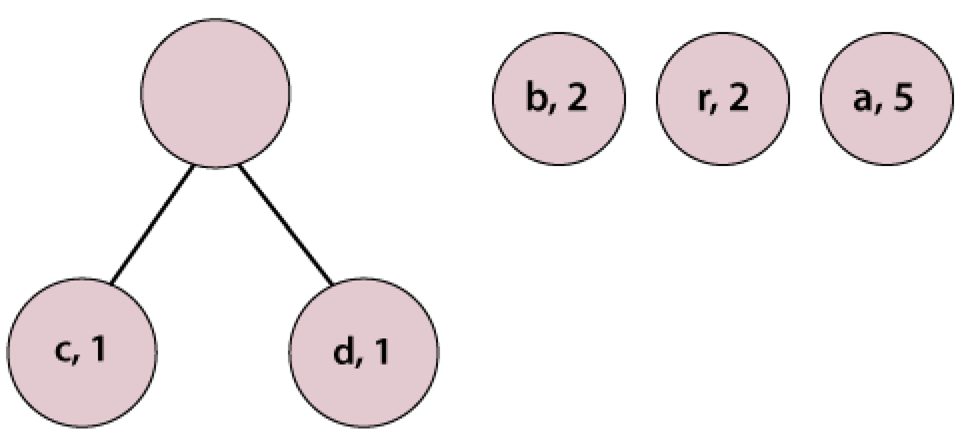
Step 1: Make pairs of characters and their frequencies.

(a, 5), (b, 2), (c, 1), (d, 1), (r, 2)

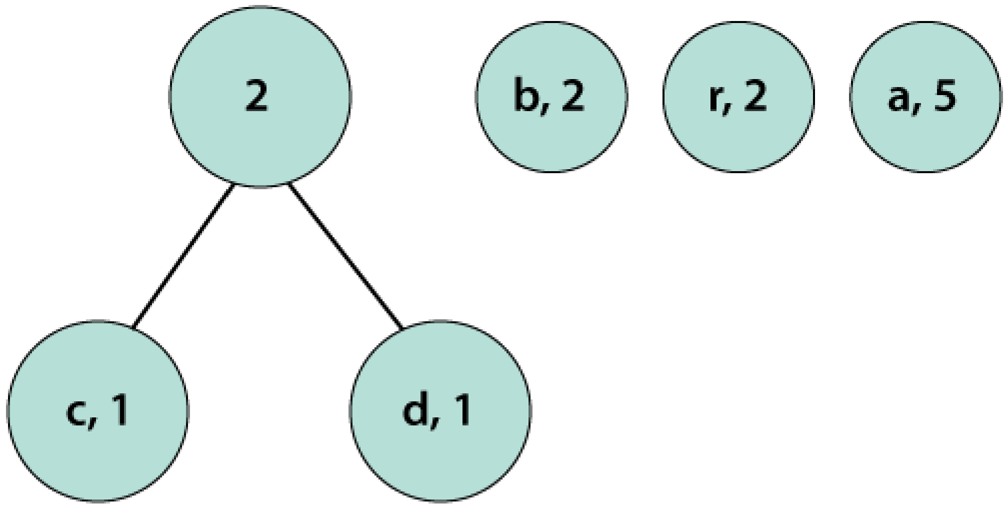
Step 2: Sort pairs with respect to frequency, we get:

(c, 1), (d, 1), (b, 2) (r, 2), (a, 5)

Step 3: Pick the first two characters and join them under a parent node.

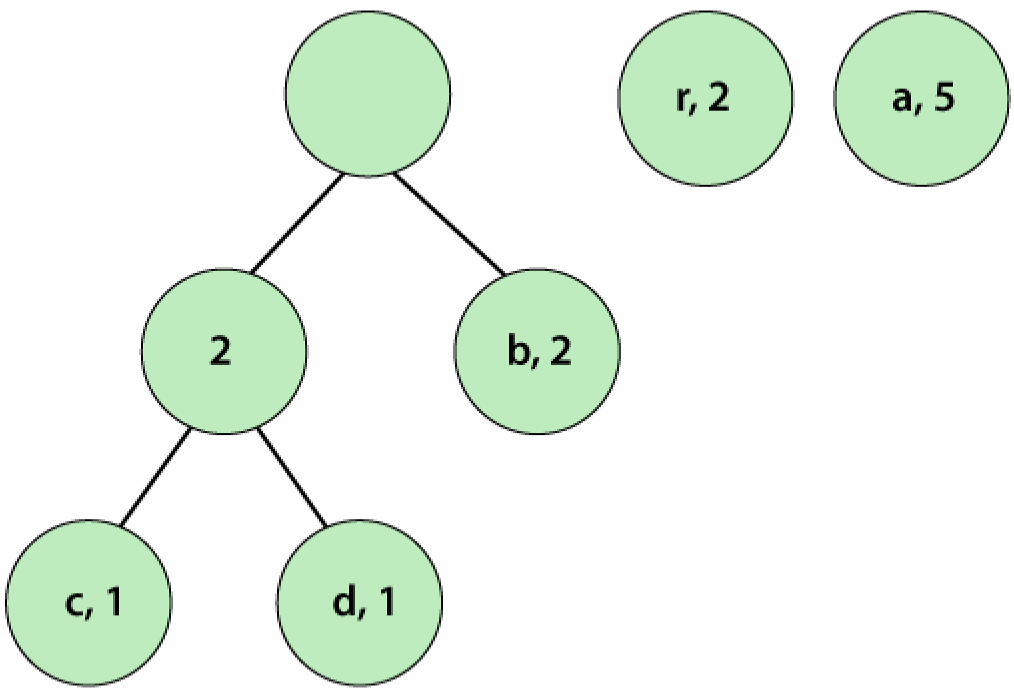


We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 1+1=**2.**

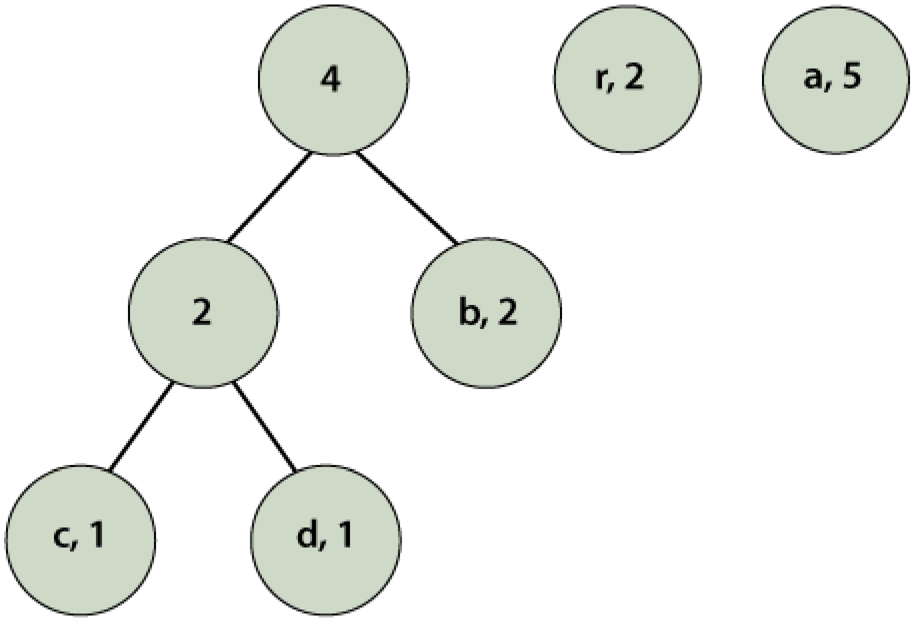


Step 4: Repeat Steps 2 and 3 until, we get a single tree.

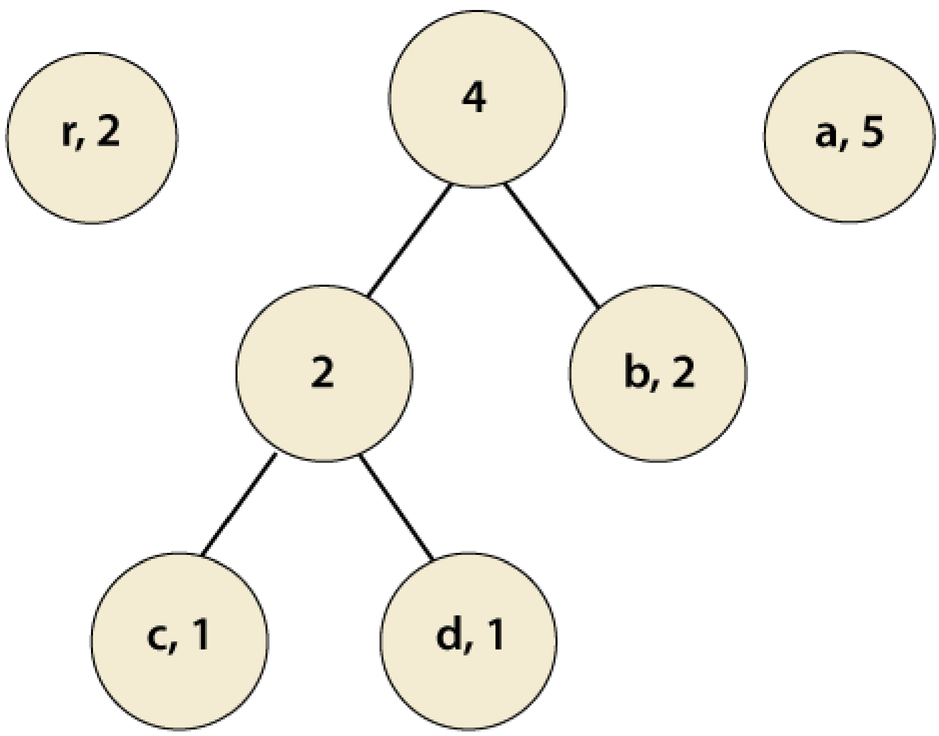
We observe that the pairs are already in a sorted (by step 2) manner. Again, pick the first two pairs and join them.



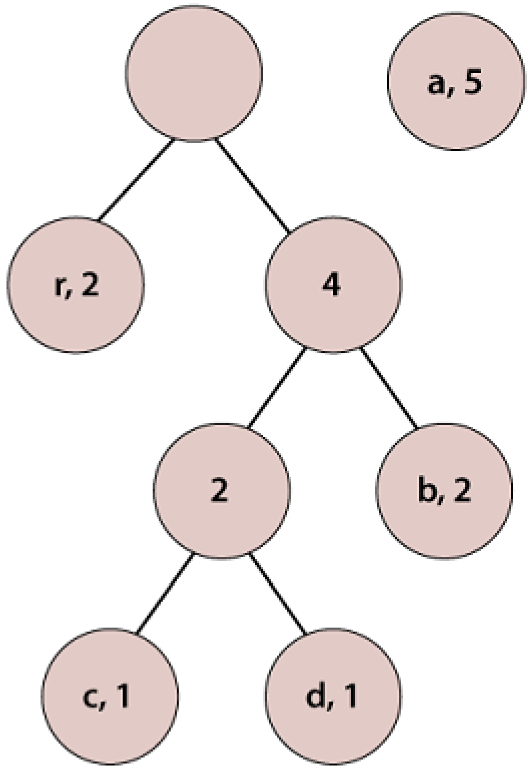
We observe that a parent node does not has a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 2+2=**4.**



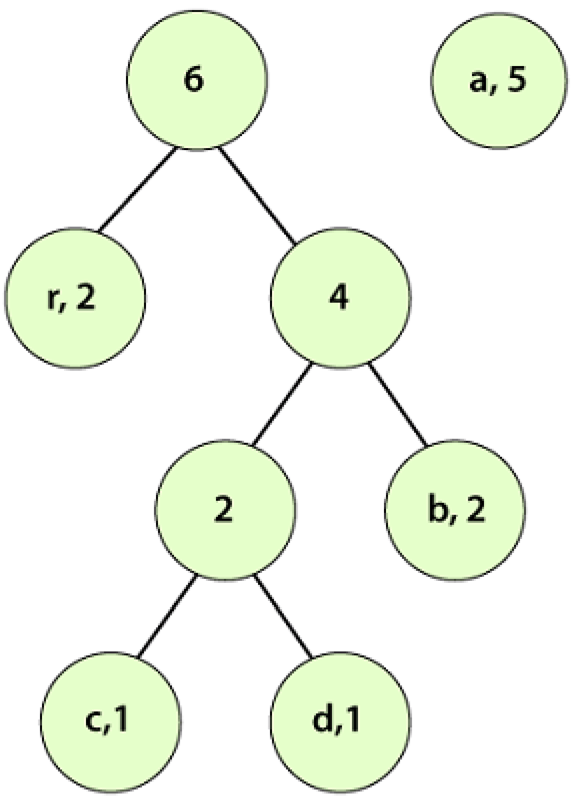
Again, we check if the pairs are in a sorted manner or not. At this step, we need to sort the pairs.



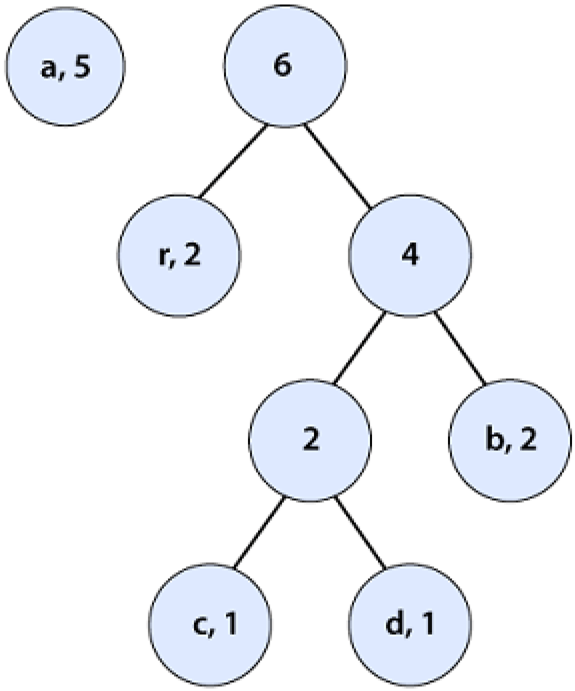
According to step 3, pick the first two pairs and join them, we get:



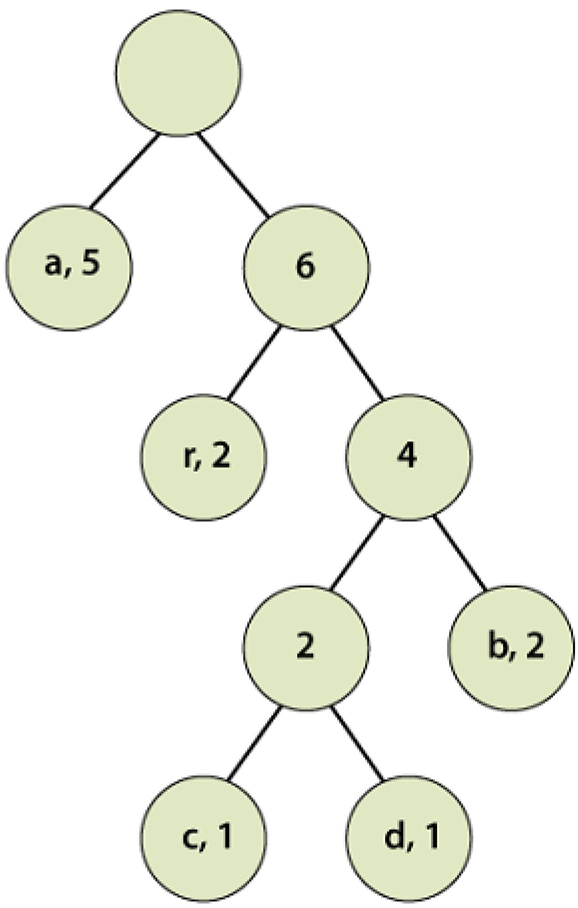
We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 2+4=**6.**



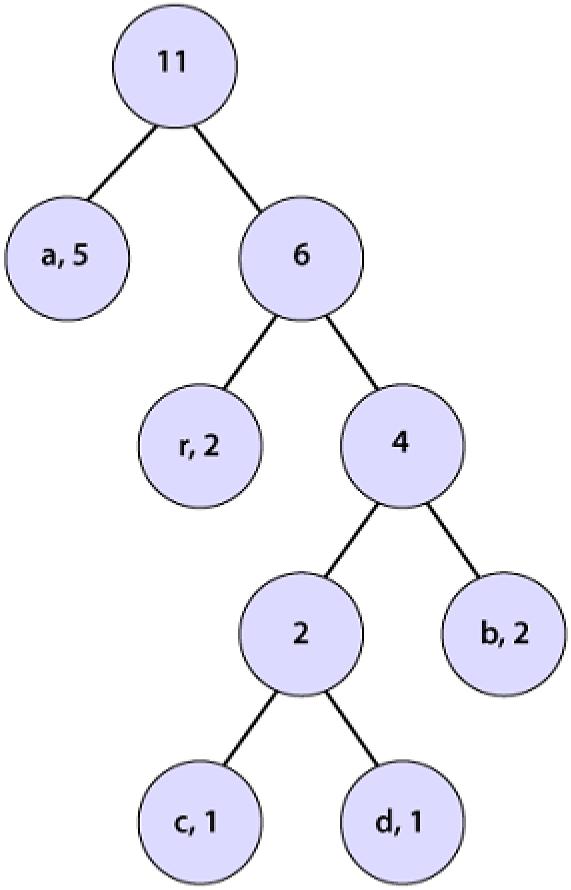
Again, we check if the pairs are in a sorted manner or not. At this step, we need to sort the pairs. After sorting the tree looks like the following:



According to step 3, pick the first two pairs and join them, we get:

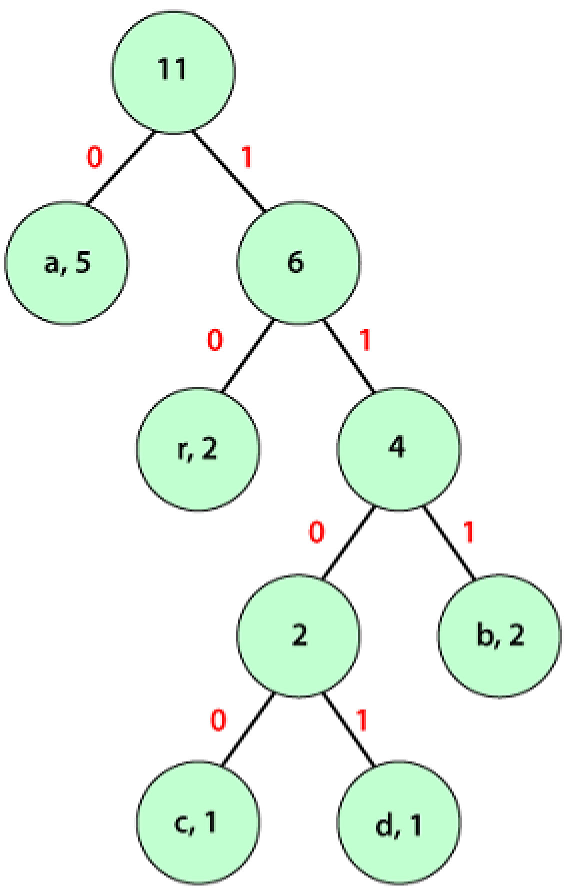


We observe that a parent node does not have a frequency so, we must assign a frequency to it. The parent node frequency will be the sum of its child nodes (left and right) i.e. 5+6=**11.**



Therefore, we get a single tree.

At last, we will find the code for each character with the help of the above tree. Assign a weight to each edge. Note that each **left edge-weighted is 0** and the **right edge-weighted is 1.**



We observe that input characters are only presented in the leave nodes and the internal nodes have null values. In order to find the Huffman code for each character, traverse over the Huffman tree from the root node to the leaf node of that particular character for which we want to find code. The table describes the code and code length for each character.

Character	Frequency	Code	Code Length
A	5	0	1
B	2	111	3
C	1	1100	4
D	1	1101	4
R	2	10	2

We observe that the most frequent character gets the shortest code length and the less frequent character gets the largest code length.

Now we can encode the string **(abracadabra)** that we have taken above.

1. 0 111 10 0 1100 0 1101 0 111 10 0

(ii) Average Code Length for the String

The average code length of the Huffman tree can be determined by using the formula given below:

1. Average Code Length = ∑ (frequency × code length) / ∑ (frequency)

= { (5 x 1) + (2 x 3) + (1 x 4) + (1 x 4) + (2 x 2) } / (5+2+1+1+2)

= 2.09090909

(iii) Length of the Encoded String

The length of the encoded message can be determined by using the following formula:

1. length= Total number of characters in the text x Average code length per character

= 11 x 2.09090909

= 23 bits

fixed-length encoding : each character uses the same number of fixed-bit storage.

Huffman Encoding

Huffman encoding implements the following steps.

- It assigns a Fixed-length code to all the given characters.
- The code length of a character depends on how frequently it occurs in the given text or string.

- A character gets the smallest code if it frequently occurs.
- A character gets the largest code if it least occurs.

There are the following two major steps involved in Huffman coding:

- First, Analyze the how many characters from the given input string or characters or text.
- Assign, a Huffman code to each character by the depending on the possibilities of (2^n) .
-

Character	Frequency	Code	Code Length
A	5	000	15
B	2	001	6
C	1	010	3
D	1	011	3
R	2	100	6

Now we can encode the string **(abracadabra)** that we have taken above.

2. 000 001 100 000 010 000 011 000 001 100 000

(ii) Average Code Length for the String

The average code length of the Huffman tree can be determined by using the formula given below:

2. Average Code Length = ∑ (frequency × code length) / ∑ (frequency)

= { (5 x 3) + (2 x 3) + (1 x 3) + (1 x 3) + (2 x 3) } / (5+2+1+1+2)

= 3

(iii) Length of the Encoded String

The length of the encoded message can be determined by using the following formula:

2. length= Total number of characters in the text x Average code length per character

= 11 x 3

= 33 bits