Algorithm-Analysis-Module-3-Important-Topics

For more notes visit

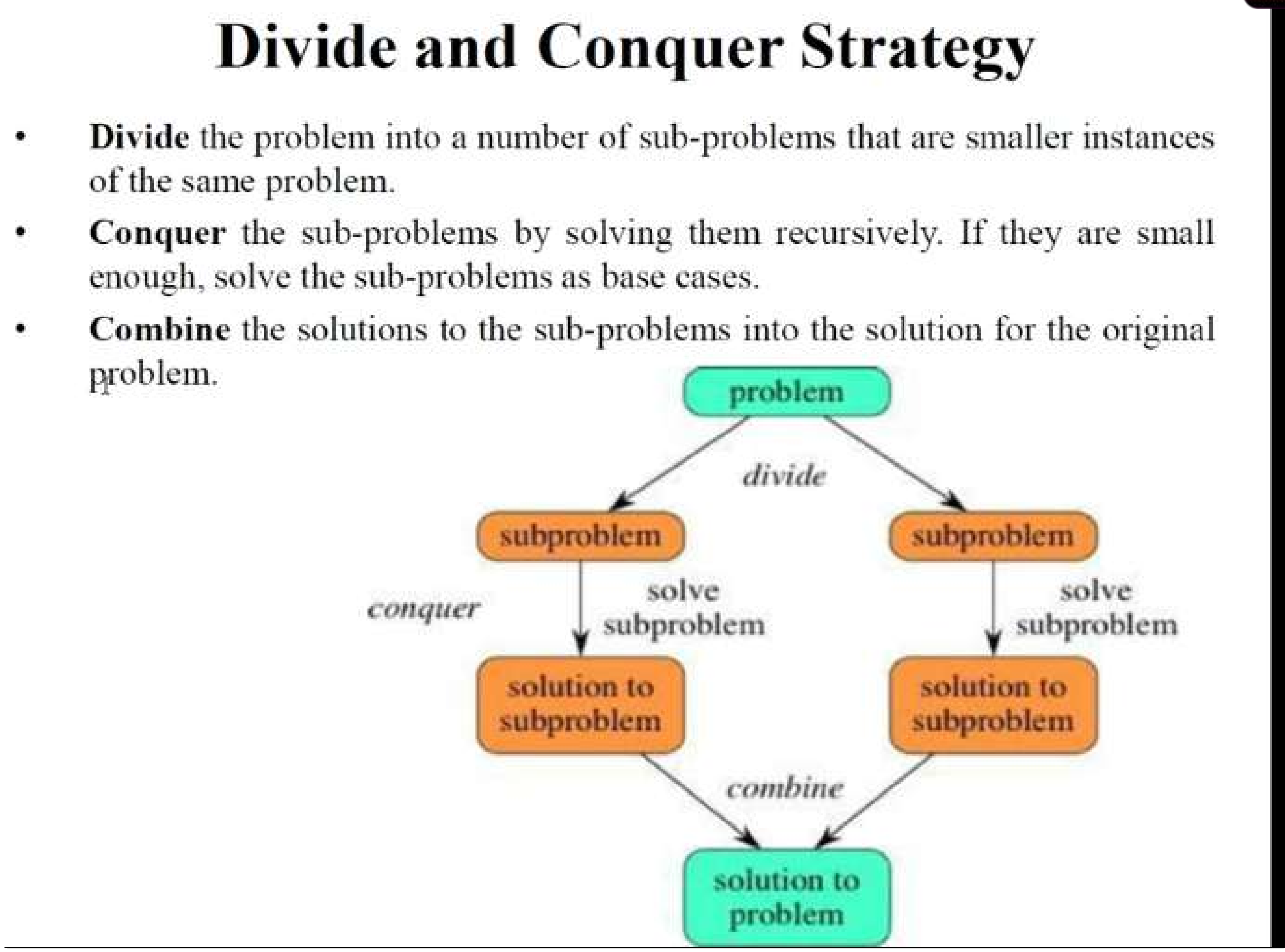
https://rtpnotes.vercel.app

---

1. Divide and conquer-control abstraction

- By control abstraction we mean a procedure whose flow of control is clear but whose primary operations are specified by other procedures whose precise meaning is left undefined

What is divide and conquer



Control abstraction

it involves breaking down a program into smaller, manageable parts while abstracting away the intricate details of how those parts work internally.

Divide and conquer control abstraction

Algorithm DAndC(P)

```
{
    if Small(P) then
            return S(P)
    else
    {
        Divide P into smaller instances P1, P2, . . . . Pk,  k≥1;
        apply DAndC to each of these sub-problems;
        return Combine(DAndC(P1), DAndC(P2), . .  ,DAndC(Pk));
    }
}
```

Divide and Conquer - Time complexity

$$T(n) = \begin{cases} g(n) & \text{n is small} \\ T(n_1) + T(n_2) + \ldots\ldots + T(n_k) + f(n) & \text{Otherwise} \end{cases}$$

T(n): Time for divide and conquer on any input of size n

f(n): Complexity of dividing the problem and combining the results.

Recurrence relation for Divide and Conquer time complexity

Complexity of many divide and conquer algorithms are given by the following recurrence relation

$$T(n) = \begin{cases} T(1) & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

2. Strassen Matrix Multiplication

Strassen's Algorithm for Matrix Multiplication-Analysis

Divide and conquer matrix multiplication

1. We need to compute the product of 2 nxn matrices A and B
2. Assume n is power of 2
3. Partition A and B into 4 square matriices, each of size n/2 x n/2
4. AB can be computed using the formula

$$\underset{C}{\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}} = \underset{A}{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}} \times \underset{B}{\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}}$$

C11 = A11 x B11 + A12 x B21
C12 = A11 x B12 + A12 x B22
C21 = A21 x B11 + A22 x B21
C22 = A21 x B12 + A22 x B22

Divide and conquer Matrix multiplication - Complexity

There are a total of 8 matrix multiplications here

$$C_{11} = A_{11}\,B_{11} + A_{12}\,B_{21}$$
$$C_{12} = A_{11}\,B_{12} + A_{12}\,B_{22}$$
$$C_{21} = A_{21}\,B_{11} + A_{22}\,B_{21}$$
$$C_{22} = A_{21}\,B_{12} + A_{22}\,B_{22}$$

- 8 recursive calls on n/2 matrix
- Addition of matrices take O(n^2) time

$$\text{Time complexity} = 8\,T(n/2) + O(n^2) = \mathbf{O(n^3)}$$
$$\text{[By Master's Theorem]}$$

$$\text{Native matrix multiplication complexity} = O(n^3)$$

Theres no difference in the complexity, so divide and conquer here is of no use

Strassens Matrix Multiplication Algorithm

1. A and B are matrices with dimension nxn
2. If n is not power of 2, add rows and columns of 0s to make the dimensions the power of 2
3. Partition A and B into 4 square matrices n/2 x n/2

   $$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}$$

   1. a,b,c and d are sub matrices of A, size m/2 x n/2
   2. e,f,g,h are sub matrices of B, size n/2 x n/2
4. Compute the 7 matrices P1 to P7

1. $$P_1 = a(f - h)$$
   $$P_2 = h(a + b)$$
   $$P_3 = e(c + d)$$
   $$P_4 = d(g - e)$$
   $$P_5 = (a + d)(e + h)$$
   $$P_6 = (b - d)(g + h)$$
   $$P_7 = (a - c)(e + f)$$
   It requires 7 matrix multiplications and 10 matrix additions/subtractions
2. Then compute $$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix}$$
   $$C_1 = P_4 + P_5 + P_6 - P_2$$
   $$C_2 = P_1 + P_2$$
   $$C_3 = P_3 + P_4$$
   $$C_4 = P_1 - P_3 + P_5 - P_7$$

Trick to learn Strassen Multiplication formula

1. **a Flaming Hawk (P1)**: a(f−h)
2. **Hungry Animal and Berries (P2)**: h(a+b)
3. **Eager Cheetah and Deer (P3)**: e(c+d)
4. **Daring Guard and Elephant (P4)**: d(g−e)
5. **All Dragons Eat Honey (P5)**: (a+d)(e+h)
6. **Bravery Doesn't Go Hungry (P6)**: (b−d)(g+h)
7. **All Cats Eats Fish (P7)**: (a−c)(e+f)

Story Based on the above

Once upon a time, in an enchanted forest, **a Flaming Hawk** soared high above the treetops.
It spotted a **Hungry Animal** munching on **Berries**.
It was an **Eager Cheetah** chasing after a **Deer**.
Nearby, was a **Daring Guard** protecting the forest from a **Elephant** known for causing

trouble.

In this magical world, there was a golden rule: **All Dragons Eat Honey**. These mystical creatures thrived on a sweet diet of honey. They often said, **Bravery Doesn’t Go Hungry**. Those who dared to take risks and face challenges head-on always found themselves well-fed and content, much like the cheetah who cleverly balanced its diet between berries and other forest offerings.

And lastly, everyone knew the simple yet profound truth: **All Cats Eats Fish**. The cats, with their sharp instincts, would always find their way to the freshest fish, symbolizing resourcefulness and the rewards of patience.

Example of Strassens multiplication algorithm

Multiply the following two matrices using Strassen’s Matrix Multiplication Algorithm

$$A = \begin{bmatrix} 6 & 8 \\ 9 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 5 \\ 3 & 6 \end{bmatrix}$$

1. A and B are the matrices with dimension 2x2
2. Here n is a power of 2
3. Partition A and B in to 4 square matrices of size n/2 x n/2 = 1 x 1

$$A = \left[\begin{array}{c|c} 6 & 8 \\ \hline 9 & 7 \end{array}\right] \qquad B = \left[\begin{array}{c|c} 2 & 5 \\ \hline 3 & 6 \end{array}\right]$$

- We get the following values for a to h

  a=6, b=8, c=9, d=7

  e=2, f=5, g=3, h=6

  -

Applying the equations..

Compute 7 n/2 x n/2 matrices

$$P_1 = a(f - h) = 6(5-6) = -6$$
$$P_2 = h(a + b) = 6(6+8) = 84$$
$$P_3 = e(c + d) = 2(9+7) = 32$$
$$P_4 = d(g - e) = 7(3-2) = 7$$
$$P_5 = (a + d)(e + h) = (6+7)(2+6) = 104$$
$$P_6 = (b - d)(g + h) = (8-7)(3+6) = 9$$
$$P_7 = (a - c)(e + f) = (6-9)(2+5) = -21$$

Then compute

$$C_1 = P_4 + P_5 + P_6 - P_2 = 7+104+9-84 = 36$$
$$C_2 = P_1 + P_2 = -6+84 = 78$$
$$C_3 = P_3 + P_4 = 32+7 = 39$$
$$C_4 = P_1 - P_3 + P_5 - P_7 = -6-32+104+21 = 87$$

$$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix} = \begin{bmatrix} 36 & 78 \\ 39 & 87 \end{bmatrix}$$

Strassens matrix multiplication complexity

- As we saw before, there are 7 matrix multiplications and 10 matrix addition/subtractions
- Addition/subtraction takes O(n^2) time
- Multiplication takes T(n/2)

$$\text{Time complexity} = 7\,T(n/2) + O(n^2)$$
$$= O(n^{\log 7})$$
$$\mathbf{= O(n^{2.81})}$$

[By Master's Theorem]

O(n^2.81) is better than O(n^3)

Recurrence relation

$$T(n) = \begin{cases} b & \text{if } n<2 \\ 7\,T(n/2) + c\,n^2 & \text{Otherwise} \end{cases}$$

$$\begin{aligned}
T(n) &= 7\,T(n/2) + c\,n^2 \\
&= 7[7\,T(n/4) + c\,n^2/4\,] + c\,n^2 \\
&= 7^2T(n/2^2) + 7\,c\,n^2/4 + c\,n^2 \\
&= 7^3T(n/2^3) + 7^2\,c\,n^2/4^2 + 7\,c\,n^2/4 + c\,n^2 \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&= 7^kT(n/2^k) + (7^{k-1}/4^{k-1})\,c\,n^2 + \ldots\ldots + (7/4)\,c\,n^2 + c\,n^2 \\
&= 7^kT(n/2^k) + [1+(7/4) + \ldots\ldots + (7^{k-1}/4^{k-1})\,]\,c\,n^2 \\
&\leq 7^kT(n/2^k) + [1+(7/4) + \ldots\ldots\ldots]\,c\,n^2 \\
&= 7^kT(n/2^k) + [1/(1-(7/4))]\,c\,n^2 \\
&\qquad\qquad [\text{Assume that } n/2^k = 1 \Rightarrow k = \log n] \\
&= 7^{\log n}T(1) - [4/3]\,c\,n^2 \\
&= n^{\log 7}O(1) - [4/3]\,c\,n^2 \\
&= \mathbf{O(n^{\log 7})} \\
&= \mathbf{O(n^{2.81})}
\end{aligned}$$

---

3. Two way merge sort

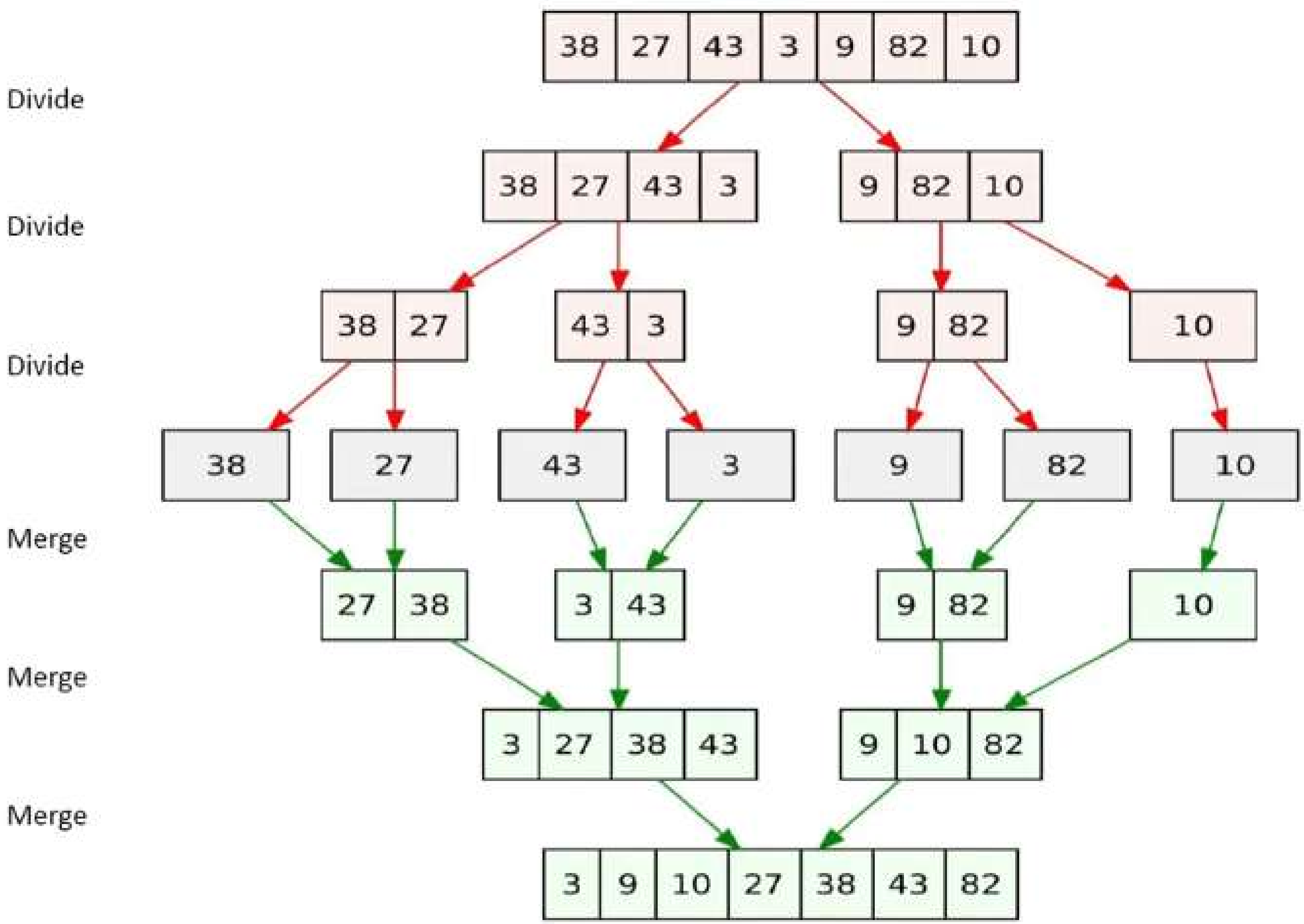
What is merge sort?

Given a sequence of n elements a[l],......a[n]. Split this array into two sets a[l],,.a[n/2] and a[(n/2)+1],...a[n]. Each set is individually sorted, and the resulting sorted sequences are merged to produce a single sorted sequence of n element.

Sorting using merge sort

- Consider the below Example,

- We have an Array 38,27,43,3,9,82,10 and our goal here is to sort it



Divide Operation

- **First we have an unsorted array**
    - 38,27,43,3,9,82,10
- **We divide the unsorted array to almost 2 pieces**
    - Here the two pieces are
        - 38,27,43,3
        - 9,82,10
- We Further divide the array into equal pieces
    - 38,27
    - 43,3
    - 9,82
    - 10
- We divide it again and get individual pieces

    - 38
    - 27
    - 43
    - 3
    - 9
    - 82
    - 10

Merge Operation

- **During merge operation, The Steps we did earlier are reversed**
- **Merge 1**
    - Merging 38 and 27
        - Sorting, we get 27,38
    - Merging 43, and 3
        - 3,43
    - Merging 9,82
        - 9,82
    - 10
        - 10
- **Merge 2**
    - Merging 27,38 and 3,43
        - 2,27,38,43
    - Merging 9,82,10
        - 9,10,82
- **Merge 3**
    - 3,9,10,27,38,43,82
-

Merge sort algorithm

```
Algorithm MergeSort(low, high)
{
        mid = (low + high )/2;
        MergeSort(low, mid);
        MergeSort(mid+1, high);
        Merge(low, mid, high);
}
```

Merge Algorithm

Algorithm

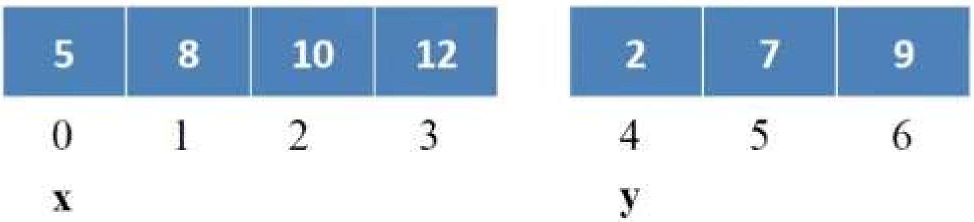
```
Algorithm Merge(low, mid, high)
{       i= low; x= low;  y= mid + 1;
        while((x ≤ mid) and (y ≤ high)) do
        {  if ( a[x] ≤ a[y] ) then
            {       b[i] = a[x];
                    x = x+1;
            }
            else
            {       b[i] = a[y];
                    y = y+1;
            }
            i=i+1;
        }
            if( x ≤ mid) then
            {  for k=x to mid do
                {       b[i] = a[k];
                        i =i+1;
                }
            }
            else
            {  for k=y to high do
                {       b[i] = a[k];
                        i =i+1;
                }
            }
            for k= low to high do
                a[k] = b[k];
}
```

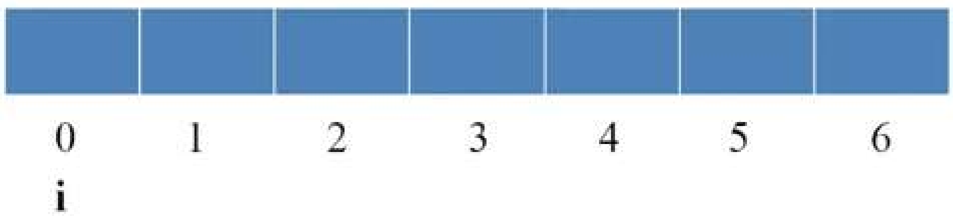
Applying the algorithm

x = first arrays starting index

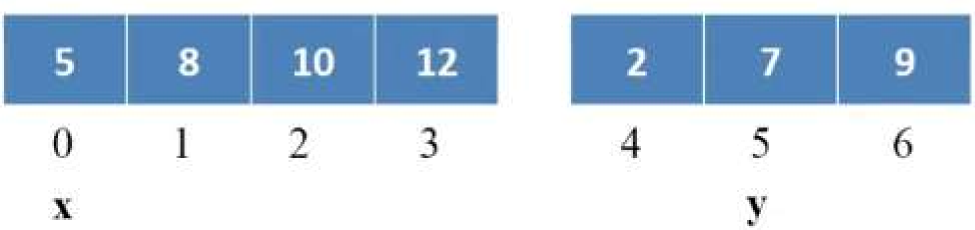
y = second arrays starting index

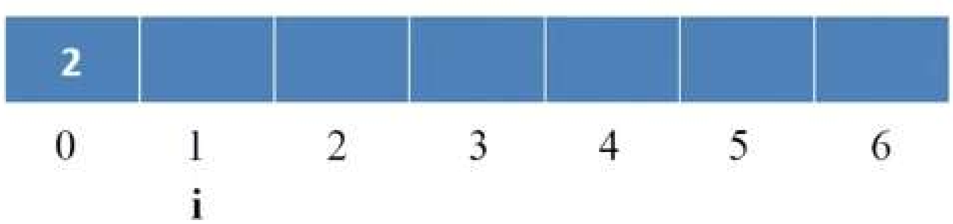
i = final arrays starting index





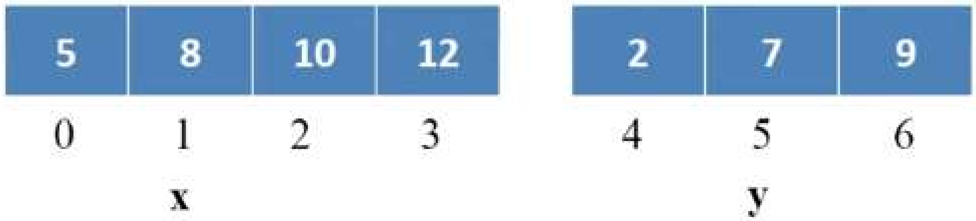
1. Compare x and y
    1. Here x is 5 and y is 2
    2. y < x
    3. Add y (2) to final array
        1. Update i and y

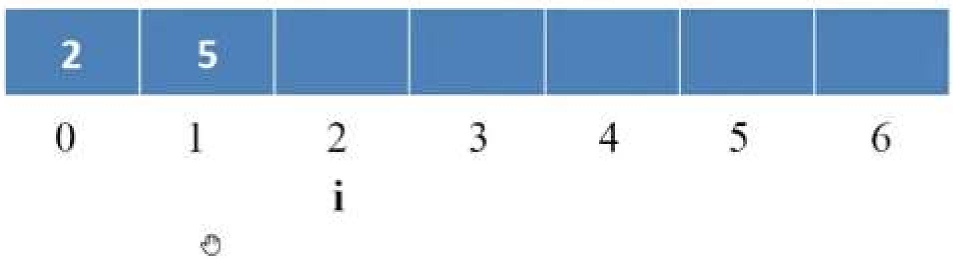




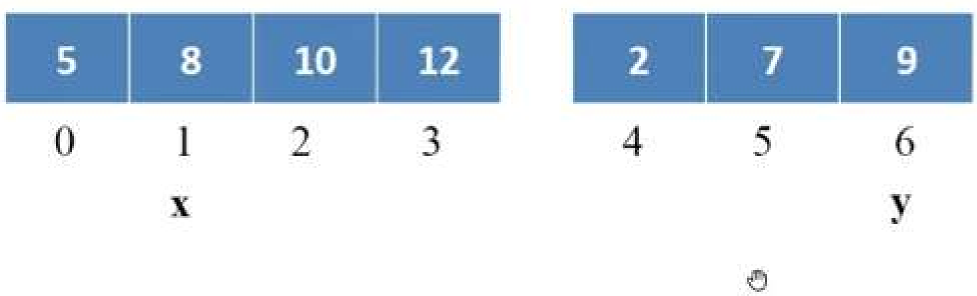
- Compare x and y again
    - x < y
    - Add 5 to final array

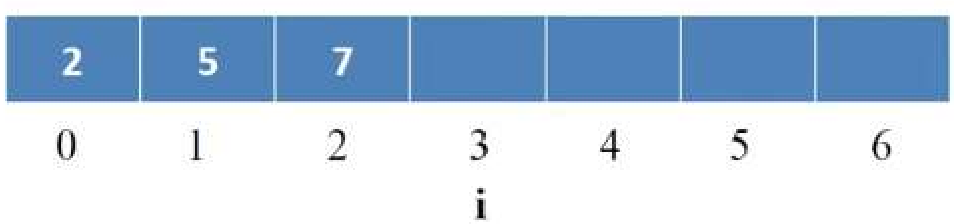
- update i and x by 1 position





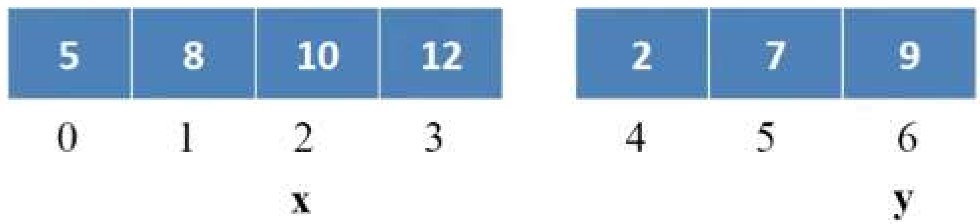
- Comparing x and y
  - y < x
  - Add 7 to the array
  - Update y and i by 1 position

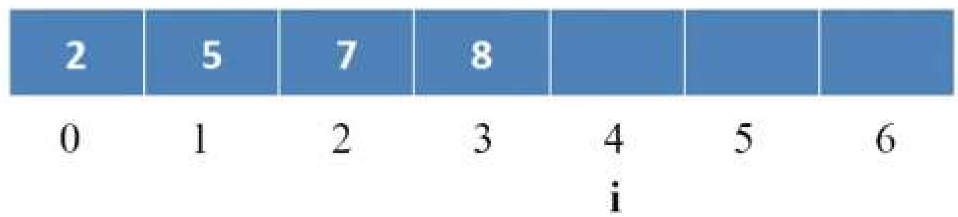




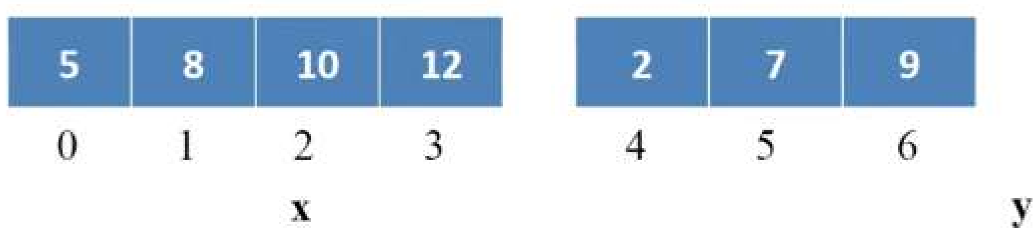
- x and y compared
  - x < y

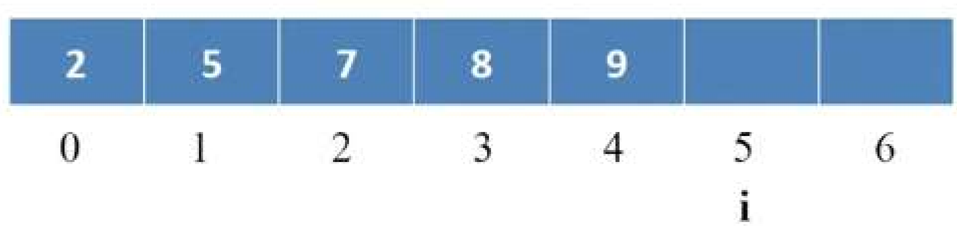
- 8 added, x and i updated





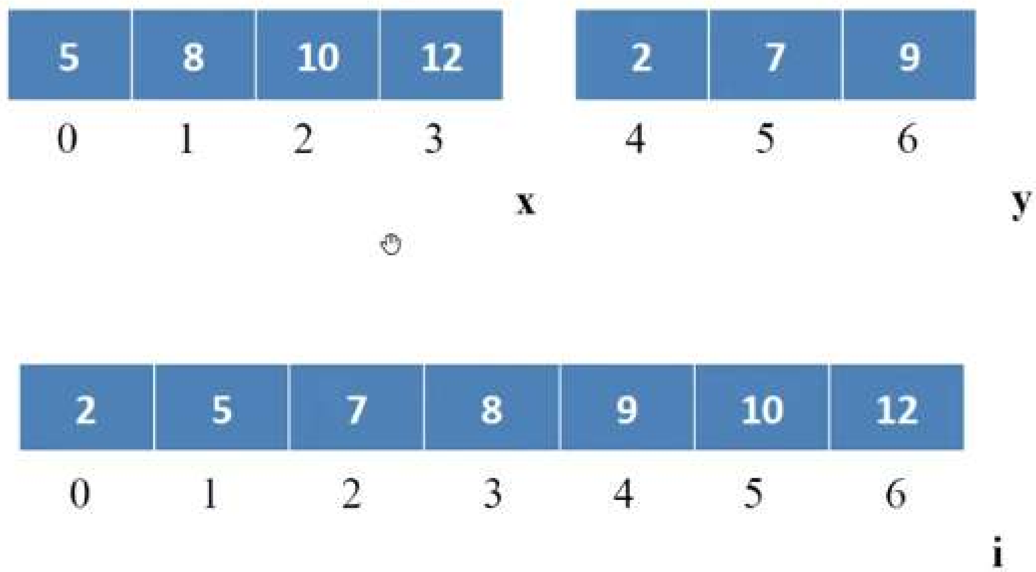
- x and y compared
  - y < x
  - 10 added
  - y and i position updated





- Second array is finished completely

- We can place the remaining things from x into the array
  - Added 10 and 12, Updating positions



Time complexity of merge sort

2 Way Merge Sort – Time Complexity

$$T(n) = \begin{cases} a & \text{if } n=1 \\ 2\,T(n/2) + cn & \text{Otherwise} \end{cases}$$

a is the time to sort an array of size 1

cn is the time to merge two sub-arrays

2 T(n/2) is the complexity of two recursion calls

2 Way Merge Sort – Time Complexity

$$
\begin{aligned}
T(n) &= 2\,T(n/2) + c\,n \\
&= 2(2\,T(n/4)+c(n/2)) + c\,n \\
&= 2^2T(n/2^2) + 2\,c\,n \\
&= 2^3T(n/2^3) + 3\,c\,n \\
&\quad \ldots\ldots\ldots\ldots\ldots\ldots \\
&= 2^kT(n/2^k) + k\,c\,n \qquad [\text{Assume that } n/2^k = 1,\ \ k = \log n] \\
&= n\,T(1) + c\,n \log n \\
&= a\,n + c\,n \log n \\
&= \mathbf{O(n \log n)}
\end{aligned}
$$

Best Case, Average Case and Worst Case Complexity of Merge Sort = O(n log n)

---

4. Greedy approach

The Control Abstraction of Greedy Strategy

```
Greedy(a, n)   //a[1..n] contains n inputs
{
    solution = Φ;
    for i=1 to n do
    {
            x = Select(a);
            if Feasible(solution, x) then
                    solution = Union(solution, x);
    }
    return solution;
}
```

- The argument is array A with n inputs
- Solution is set to null
- Forloop iterates n times
    - Select function is used to select one item from a
    - The item is assigned to x
    - Check if x is feasible
        - If feasible add it to solution, otherwise discard x

---

5. Fractional knapsack problem

What is Fractional Knapsack Problem?

This problem can be solved using greedy strategy

- We have a knapsack or bag of capacity **m**
- **n** is the number of objects
- **Wi** is the weight of object i

- **Pi** is the profit of object i
- **Xi -** Fraction of ith object placed in the knapsack
- **PiXi** - Profit earned from ith object
- The objective is to obtain an optimal solution of the knapsack that maximises the total profit earned
- Total weight of all chosen objects should not be more than m

Fractional Knapsack Problem

- Fractional knapsack problem can be stated as

Maximize $\sum_{i=1}^{n} PiXi$ —— 1

Subject to $\sum_{i=1}^{n} wixi \leq m$ —— 2

$0 \leq Xi \leq 1$ and $1 \leq i \leq n$ —— 3

- A **feasible solution** satisfies equation 2 and 3.
- An **optimal solution** is a feasible solution that satisfies equation 1.

Fractional knapsack problem - Algorithm

- We arrange the objects in descending order of profit/weight

```
Algorithm GreedyKnapsack(m, n)    // m → knapsack capacity
{    for i= 1 to n do
             x[i] = 0.0;                  // x[1:n] → solution vector
     U = m;
     for i=1 to n do
     {        if w[i] > U then
                     break;
              x[i] = 1.0
              U = U – w[i];
     }
     If i ≤ n then
                 x[i] = U / w[i];
}
```

1. Initially x values are set to 0
2. Set knapsack capacity to U
3. if current object weight > Balance capacity in knapsack
    1. Break
4. Else
    1. set x value as 1
    2. Subtract the capacity with the current weight
5. There are 2 cases
    1. When i<n (When loop is broken)
        1. Assign the remaining weight to x by U/w[i]
    2. When i = n+1 (When loop is not broken)
        1. Nothing

Fractional knapsack problem - Time Complexity

- For loop will execute maximum n times
- Time complexiy = O(n)

Fractional knapsack - Problem -1

Find the optimal solution for the following fractional Knapsack problem. Given number of items(n)=4, capacity of sack(m) = 60, W={40,10,20,24} and P={280,100,120,120}

Here, Given

- Maximum capacity m = 60
- Number of items n = 4
- i = {1, 2, 3, 4}
- Profit values are = {280,100,120,120}
- Weight values are = {40,10,20,24}
- Profit/Weight values are = {7, 10, 6, 5}

Now we to sort the Profit/Weight in descending order

- We will get {10,7,6,5}
- Arrange the corresponding values of i, profit and weight

i ➔ { 2, 1, 3, 4 }

P = {100, 280, 120, 120}

W= {10, 40, 20, 24}

- Now place these data in a table, in the descending order
- Here
  - i is item number
  - Pi Profit
  - Wi Weight
  - Xi is the fraction

- U is the knapsack balance capacity

i	Pi	Wi	Xi	U = U-Wi
2	100	10		
1	280	40		
3	120	20		
4	120	24		

Initially our knapsack balance capacity is the total capacity m = 60

- Initially, set all fraction to 0

i	Pi	Wi	Xi	U = U-Wi
2	100	10	0	
1	280	40	0	
3	120	20	0	
4	120	24	0	

i = 1

- U = 60
- Wi = 10
- Completely place the weight
- set Xi = 1
- Balance knapsack capacity is U - Wi = 60 - 10 = 50

i	Pi	Wi	Xi	U = U-Wi
2	100	10	1	50
1	280	40	0	
3	120	20	0	
4	120	24	0	

i = 2

- U = 50
- Wi = 40
- Completely Place the weight

- set Xi = 1
- U = U - Wi = 50 - 40 = 10

i	Pi	Wi	Xi	U = U-Wi
2	100	10	1	50
1	280	40	1	10
3	120	20	0	
4	120	24	0	

i = 3

- U = 10
- Wi = 20
- We cant completely place the weight, Weight is 20, but the capacity is only 10
- We can only place a fraction, its calculated by
    - U/Wi = 10/20 = 1/2
- Xi = 1/2
- U = U-Wi = 10 - 10 = 0

i	Pi	Wi	Xi	U = U-Wi
2	100	10	1	50
1	280	40	1	10
3	120	20	1/2	0
4	120	24	0	

i = 4

- No more capacity, capacity is 0, so stopping here

Solution

- Total Profit = Sum of (profit x Fraction)
- = 100 x 1 + 280 x 1 + 120 x 1/2 = 440
- Solution vector X = {1,1,1/2,0} (Obtained from Xi)

Fractional knapsack - Problem - 2

Find an optimal solution to the fractional knapsack problem for an instance with number of items 7, Capacity of the sack =15, $(p_1,p_2,\ldots,p_7)=(10,5,15,7,6,18,3)$ and $(w_1,w_2,\ldots,w_7)= (2,3,5,7,1,4,1)$.

m = 15

n = 7

i → { 1, 2, 3, 4, 5, 6, 7}

P = {10, 5, 15, 7, 6, 18, 3}

W = { 2, 3, 5, 7, 1, 4, 1}

Sorting in descending order...

i → {5, 1, 6, 3, 7, 2, 4}

P = {6, 10, 18, 15, 3, 5, 7}

W= {1, 2, 4, 5, 1, 3, 7}

We will get the table as

i	Pi	Wi	Xi	U = U-Wi
5	6	1	0	
1	10	2	0	
6	18	4	0	
3	15	5	0	
7	3	1	0	
2	5	3	0	
4	7	7	0	

U = m = 15

We will get the solution as..

i	Pi	Wi	Xi	U = U-Wi
5	6	1	1	14
1	10	2	1	12
6	18	4	1	8
3	15	5	1	3
7	3	1	1	2
2	5	3	2/3	0
4	7	7	0	

Total Profit = Σ Pi * Xi

$$=6x1 + 10x1 + 18x1 + 15x1 + 3x1 + 5x2/3 = \mathbf{55.33}$$

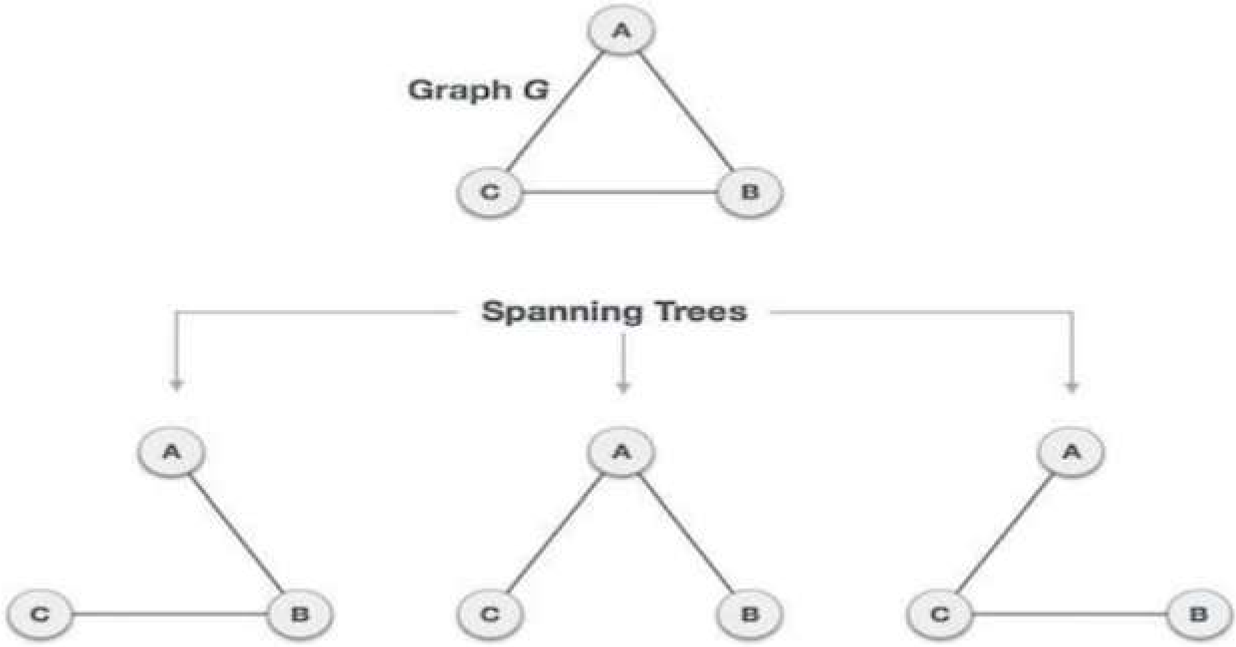
Solution vector **X ={1, 2/3, 1, 0, 1, 1, 1}**

---

6. Minimum cost spanning tree, Kruskals Algorithm

What is a spanning tree?

- Its basically a subset of a graph, which has all vertices covered with minimum number of edges

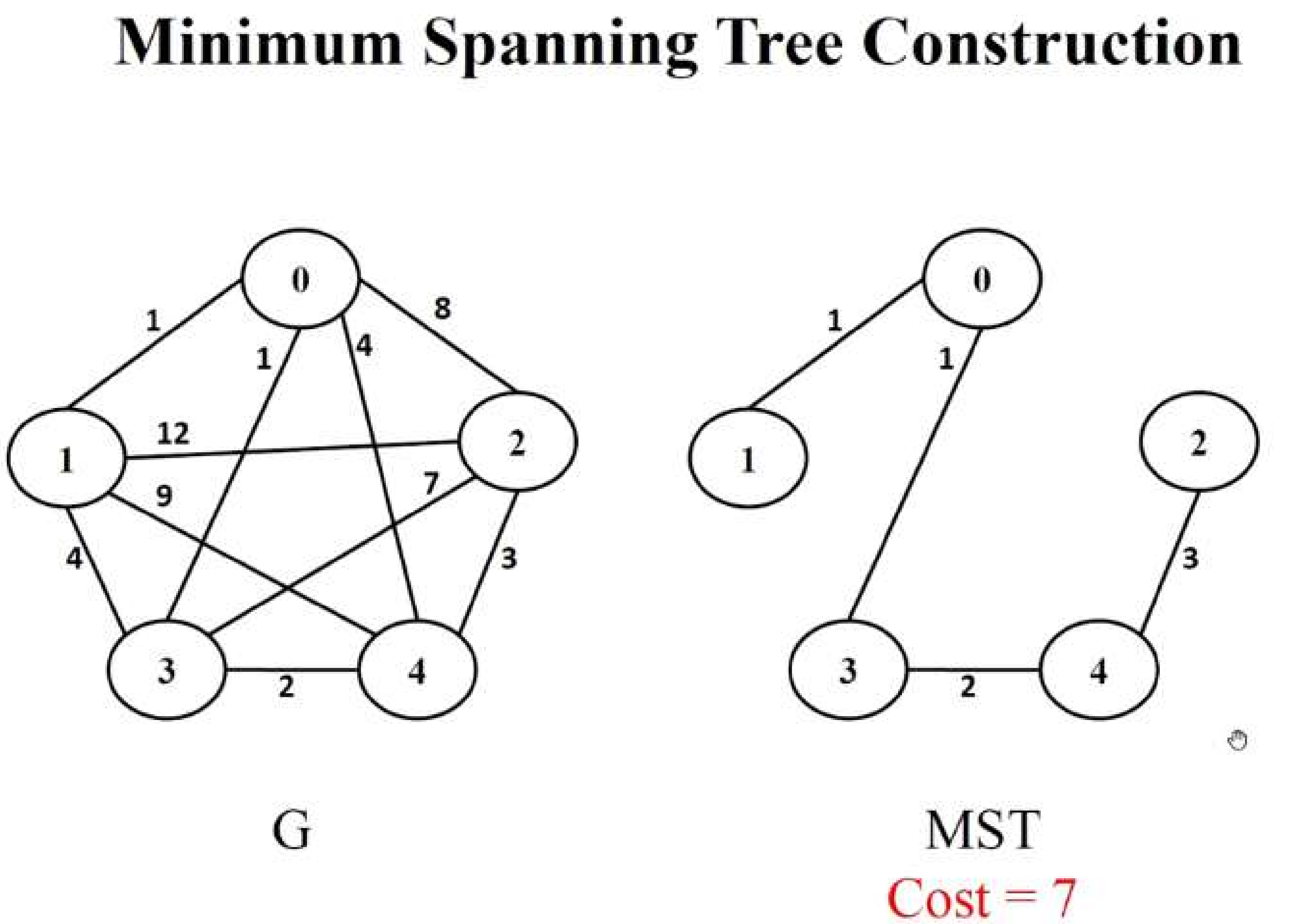


- Here we can see, we got 3 spanning tree from a graph, and all vertices are covered
- Some points
  - Spanning tree has **n-1** edges, where n is the number of nodes
  - Adding one edge to spanning tree will create a loop
  - Removing one edge from spanning tree will make the graph disconnected
  - A Spanning tree doesnt have any loops
  - All spanning tree has same number of edges and vertices
  - A graph can have more than one spanning tree
  - Total number of spanning tree possible for complete graph with n vertices = n^(n-2)

Minimum spanning tree

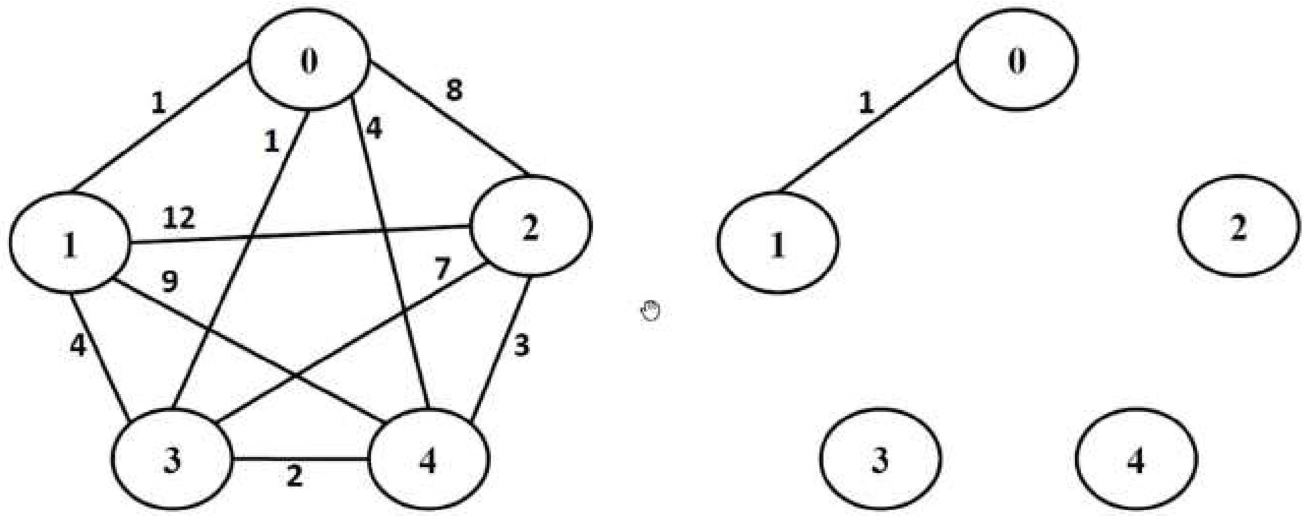
- A minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph

Examples



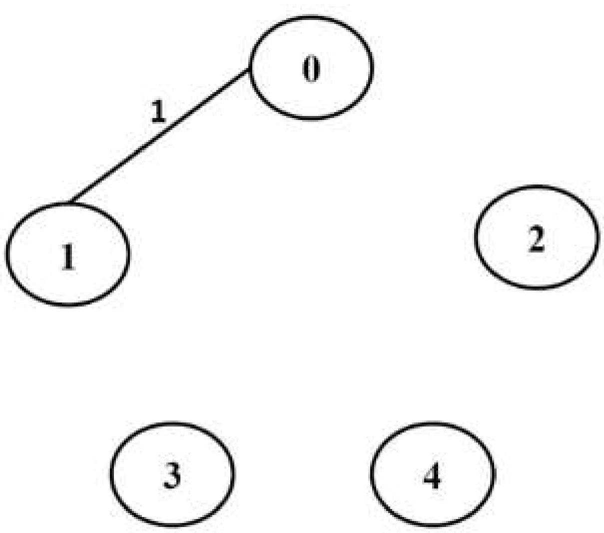
- The cost is the weights added up together

Qtn) What is the minimum possible weight of a spanning tree T in this graph such that vertex 0 is a leaf node in the tree T?

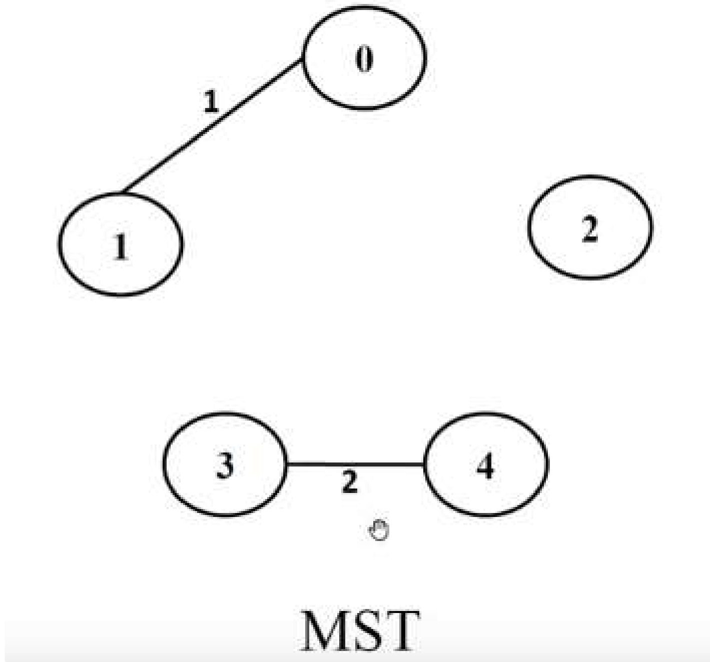


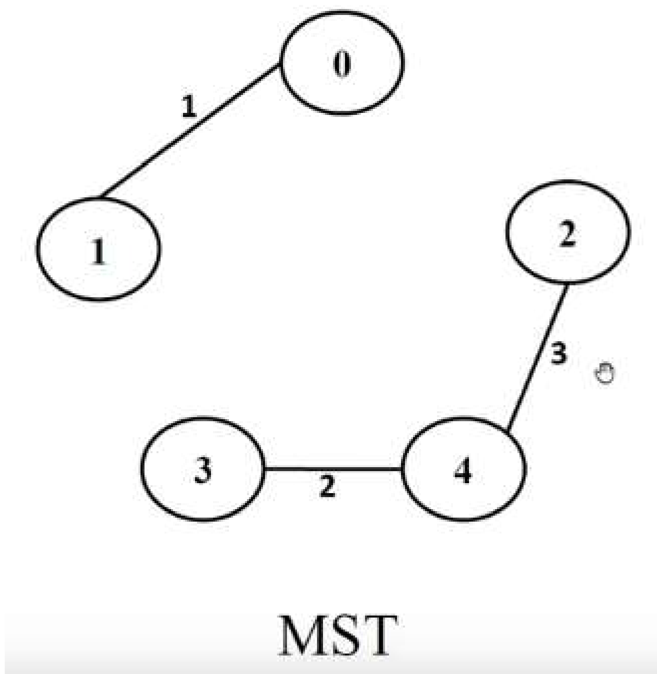
G MST

- We have a condition, 0 should be a leaf node
- So we start with 0

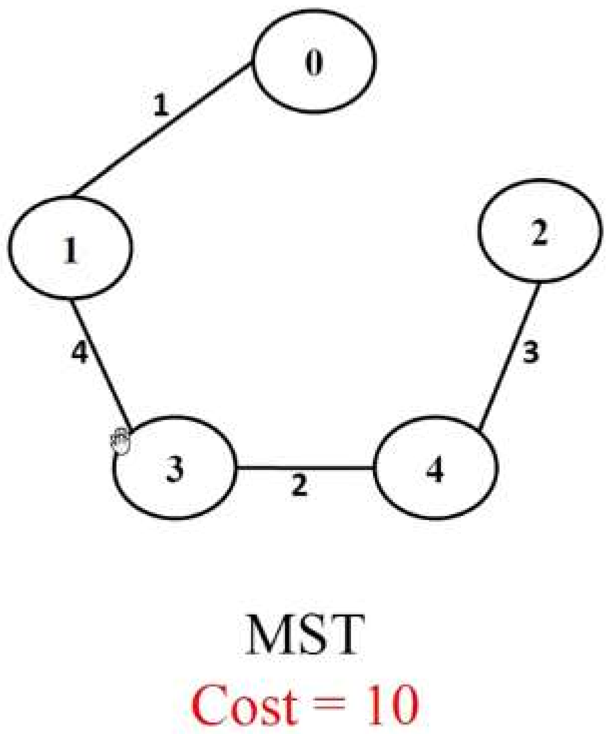


- MST
- Next smallest weight is 2 (from 3 to 4)

- 

- Next smallest is 3 (from 4 to 2)
- 

- Next smallest is 4 (from 1 to 3)

- 

Kruskal’s Algorithm

- Builds tree edge by edge
- Edges are considered in the increasing order of cost
- If the selected edge forms a cycle, discard
- The selection process continues until there are n-1 edges

Algorithm Kruskal_MST (G)

```
1. A = ϕ   // empty set
2. for each vertex v in G.V do
3.        MAKE_SET(v)
4. Arrange all edges in G.E in ascending order
       of cost/weight
5. for each edge (u, v) from the sorted list do
6.        if FIND_SET(u) ≠ FIND_SET(v) then
7.                union(u, v)
8.                A = A ∪ {(u, v)}
9. Return A
```

Analysis

- The for loop in statement no. 2 execute on the order of no. of vertices present in the Graph.
  ie, $O(|V|)$

- Statement no. 4 is an ~~sorting~~ execution of sorting algorithm which sorts all the edges present in the graph. Let us assume we are using the best sorting algorithm, quick sort. then cost will be $O(|E|\log|E|)$

Statement no. 5 executes on the order of no. of edges present in the graph, ie, cost is $O(|E|)$

$\therefore$ Running ~~as~~ time complexity of kruskal's algorithm depends on statement no. 4 and the complexity is $O(|E|\log|E|)$

7. Dijikstras algorithm

Algorithm

- During initialise single source
  - Setting distance of all vertices to infinite

  - Setting parent of all the vertices to null
  - Setting distance of Source to 0
- During Relax(u,v,w)
  - We are updating the newer distance if older distance is larger and updating the parent
- During the main algorithm
  - We call initalize single source
  - Setting Q to be the set of all vertices in G
  - While Q is not empty
    - u is set to the minimum distance in Q
    - Removing u from Q
    - For each vertex v in adjacent(u), call the RELAX function

Initialise single source (G, s)

```
for each vertex v ∈ V(G) do
    d[v] ← ∞
    π[v] ← NIL
    d[s] ← 0
```

Relax (u, v, w)

```
If d[v] > d[u] + w[u,v] then
    d[v] ← d[u] + w[u,v]
    π[v] ← u
```

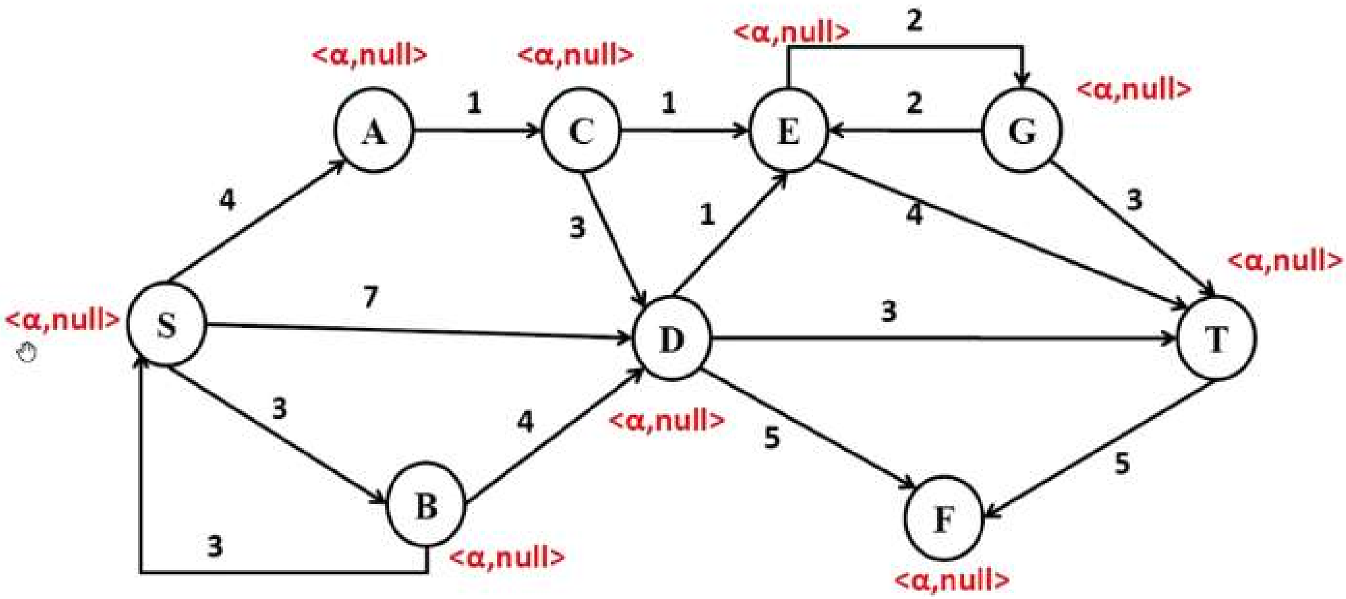
Dijkstra's Algorithm

Dijkstra (G, w, s)

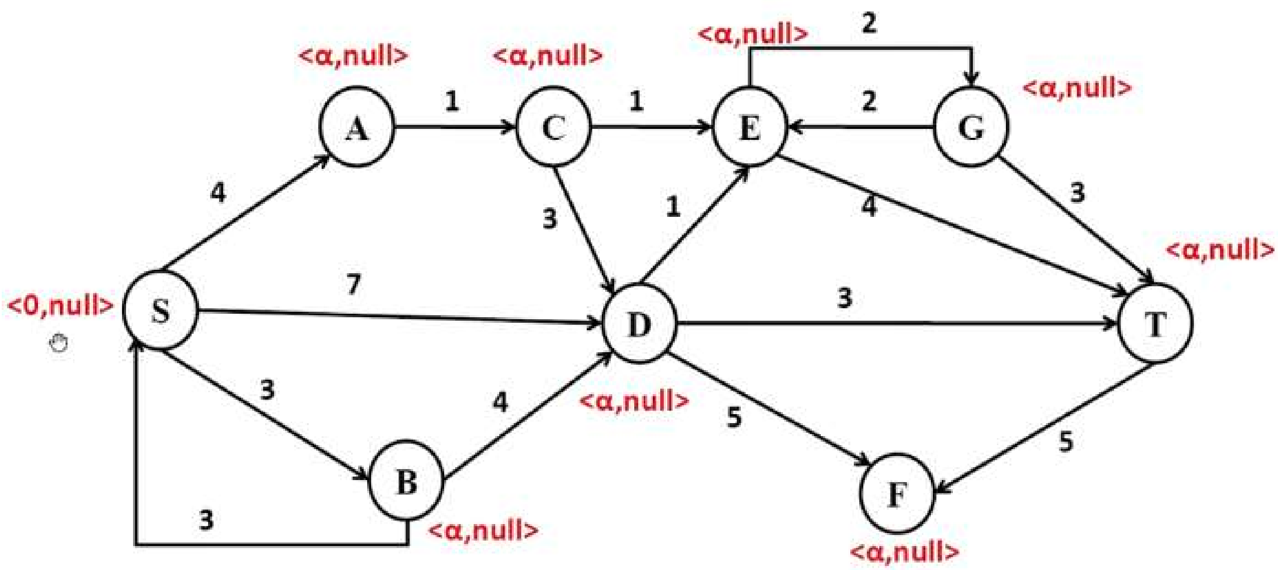
```
Initialize single source (G, s)
S ← ϕ          # set containing visited vertices
Q ← V[G]       # Minimum priority queue
while Q ≠ ϕ do
    u ← EXTRACT_MIN(Q)
    S ← S ∪ {u}
for each vertex v ∈ Adj[u] do
    RELAX (u, v, w)
```

Example of Dijkstra's algorithm

Set all vertices to infinity distance and null previous

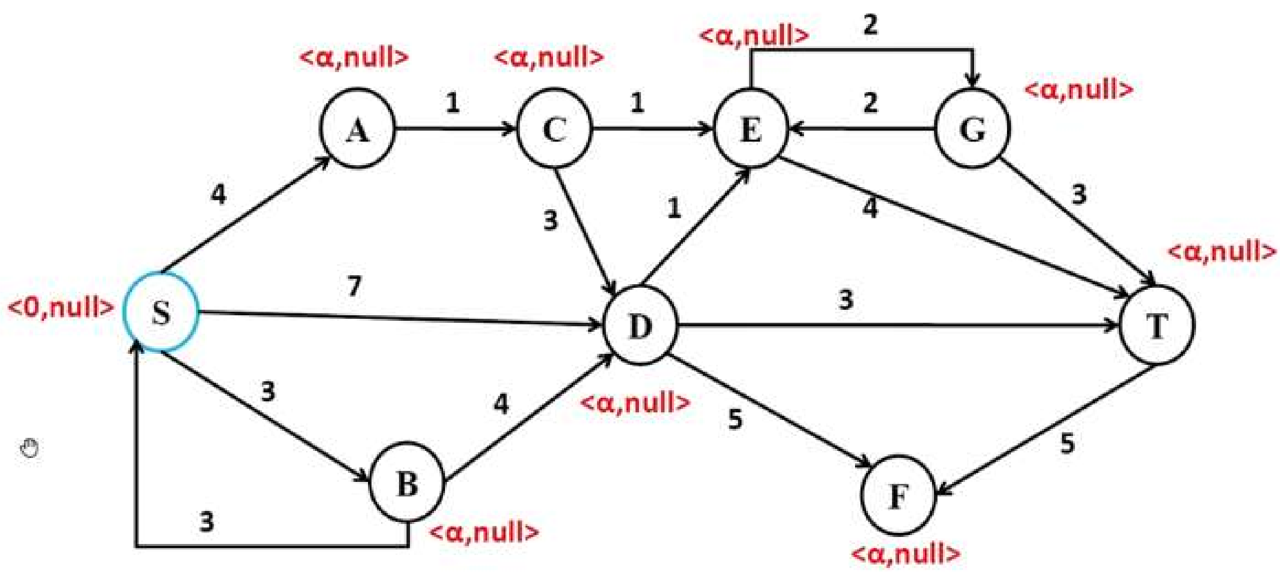


- Set distance of S to 0



	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α

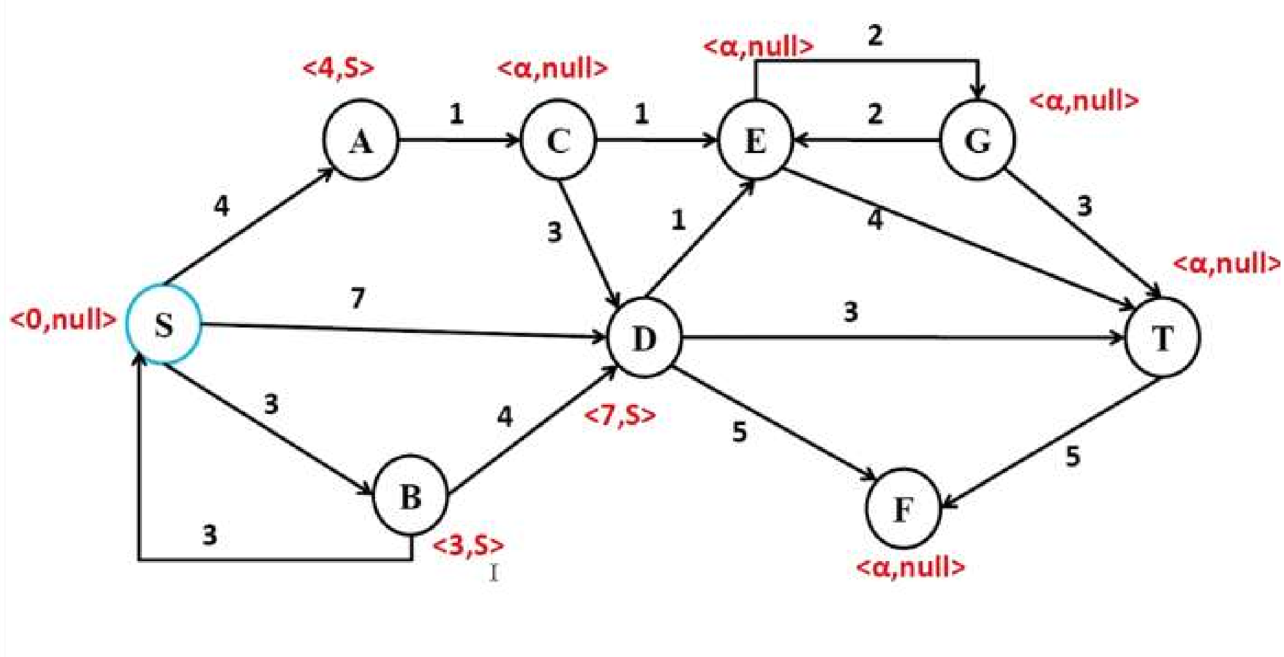
- The minimum distance vertex now is S
- So, remove it, marking it as blue



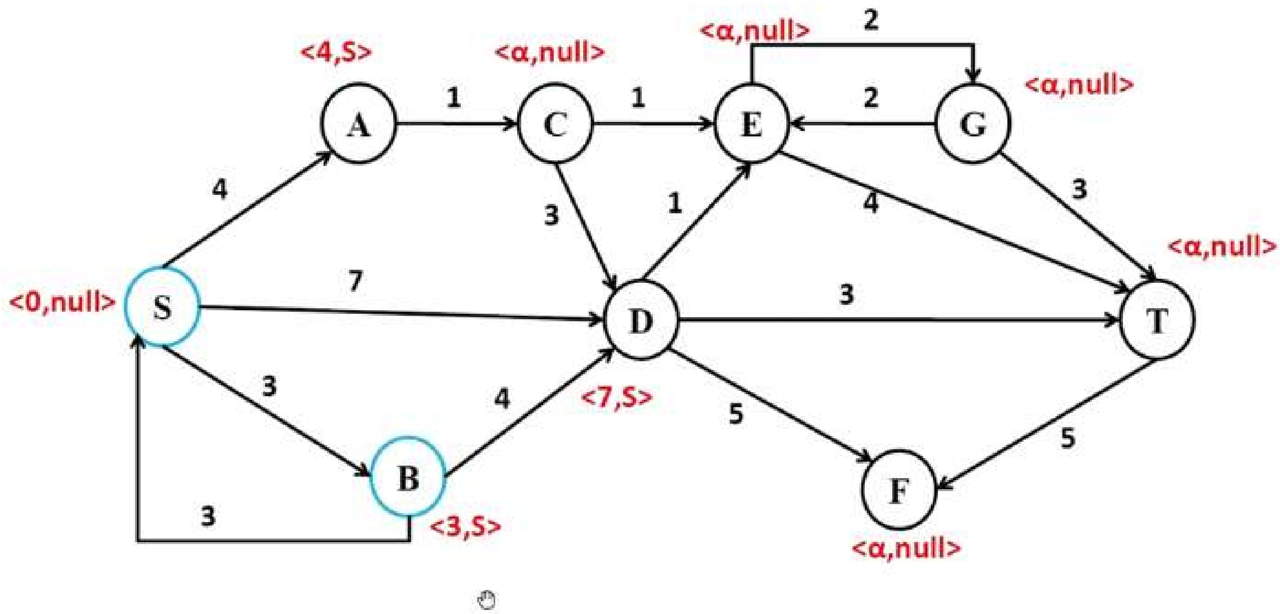
- Get the neighbour nodes of S
  - A, D and B
- Distance from A = 4 < Infinity
  - Setting Distance of A = 4, previous node = S
- Distance from D = 7 < infinity
  - Setting Distance of D = 7, previous node = S

- Distance from B = 3 < infinity

  \- Setting Distance of B = 3, previous node = S



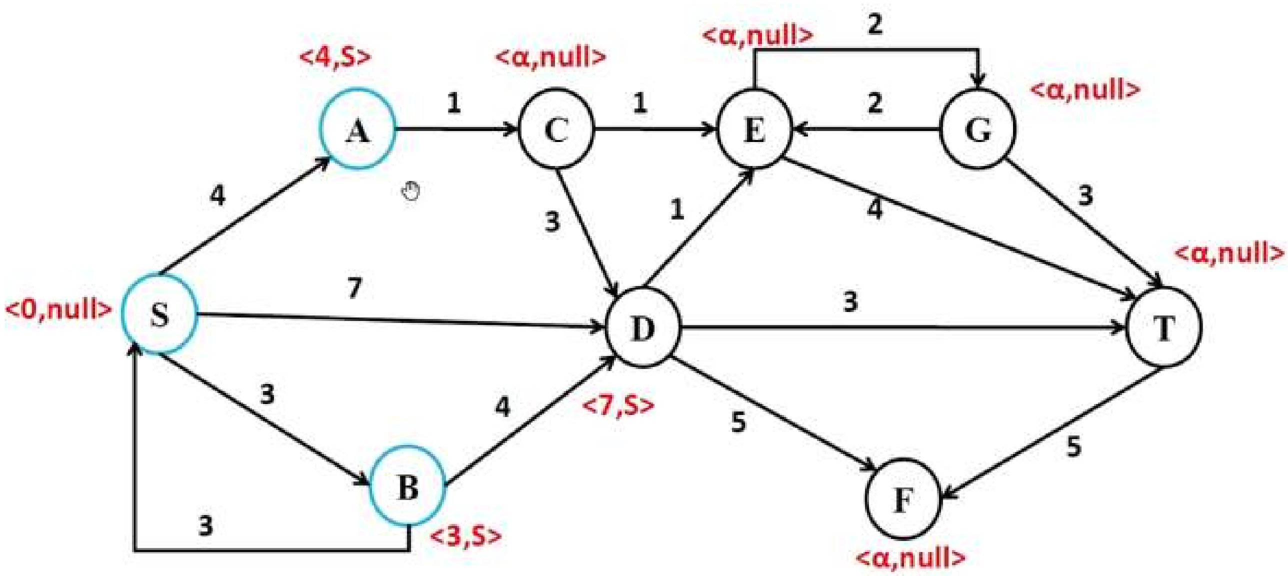
- Among the nodes, the node with minimum distance is Node B with 3.
- Removing Node B



-
- Neighbours of B
  - D
  - Cost from D = 3 + 4 = 7, Which is equal to previous distance 7, not updating

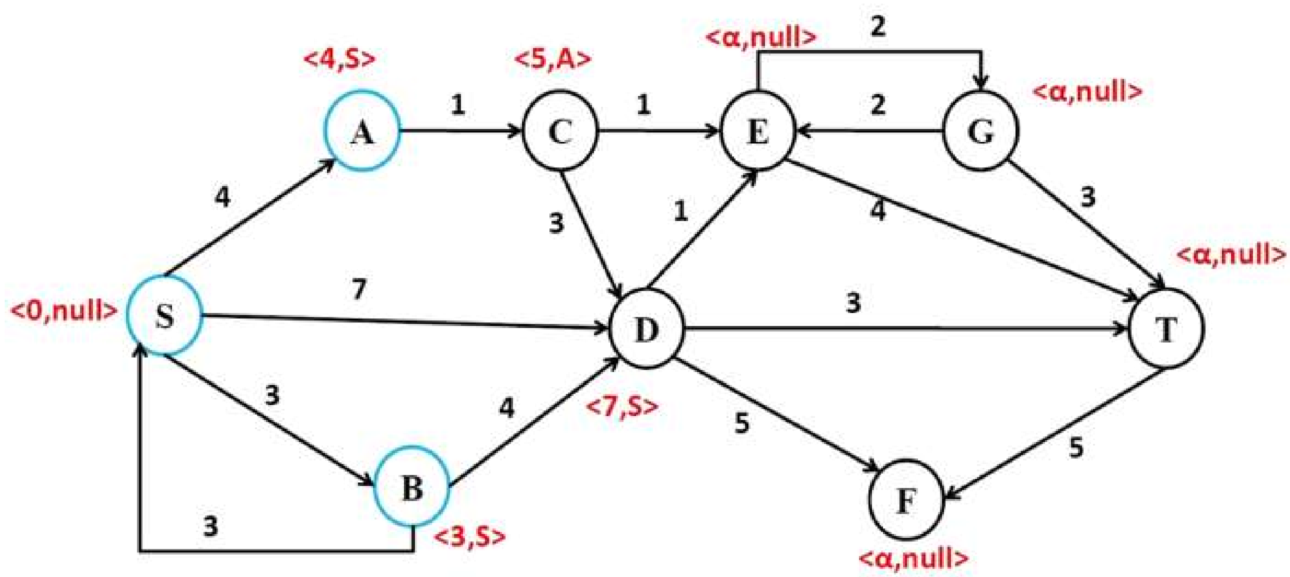
	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
		S	S		S				

- Bottom row denotes the previous node
- Among the nodes, A is the node with minimum distance 7
- Removing Node A

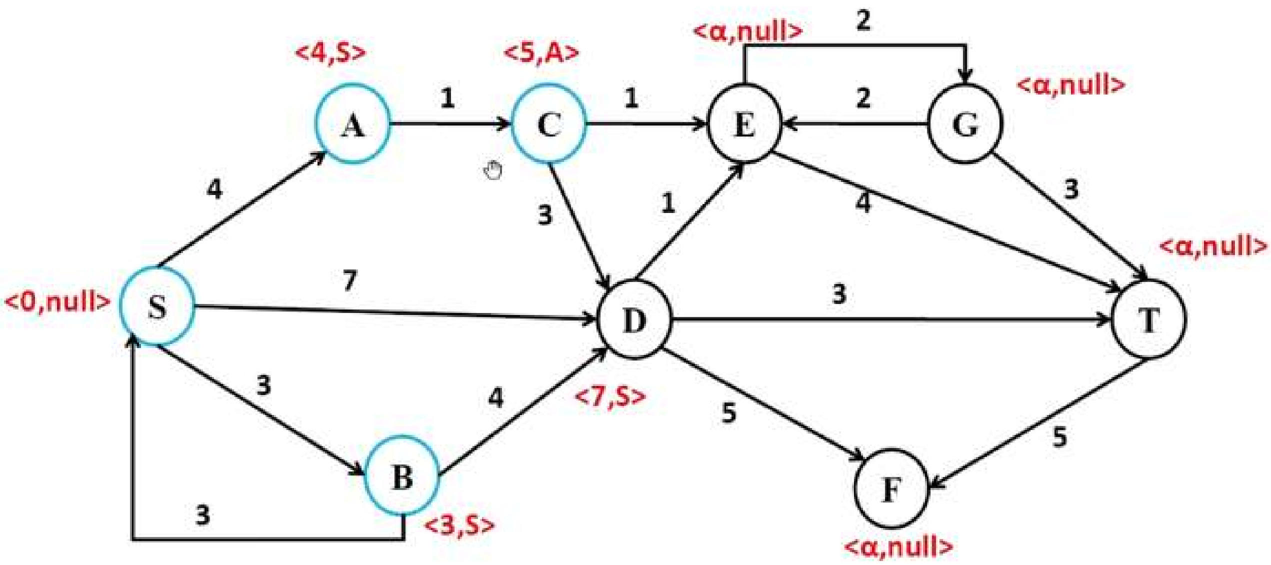


	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
		S	S		S				

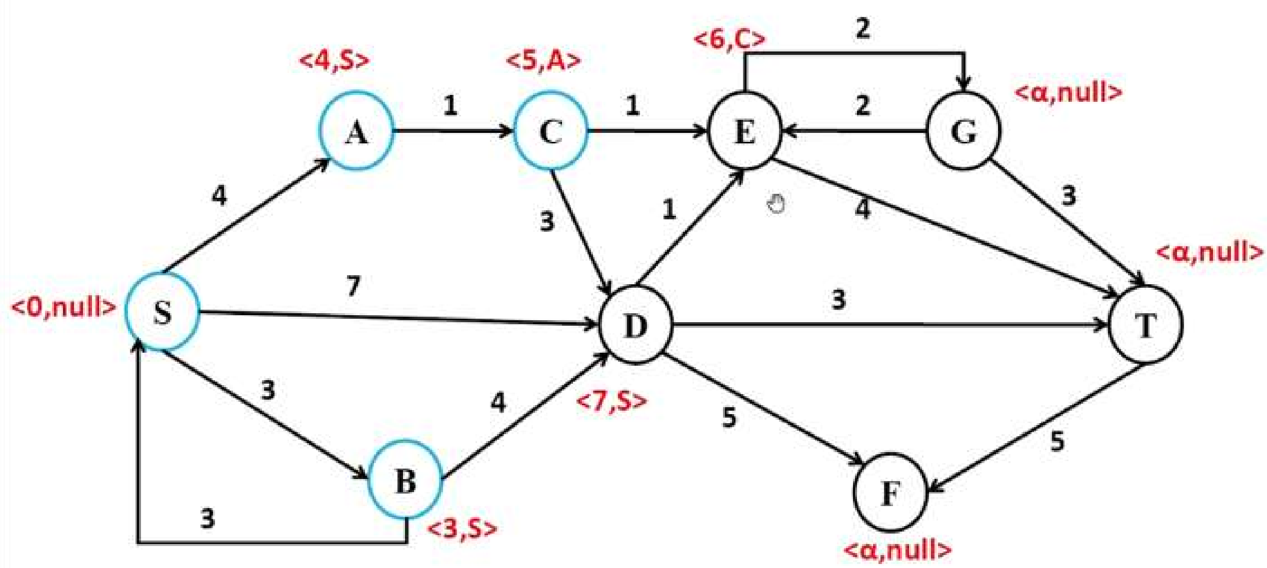
- Neighbour of A = C
  - Distance from S = 1 +4 = 5 < infinity
  - Setting Distance A = 1, and previous = A



- The node with minimum distance is C with 5
- Removing node C



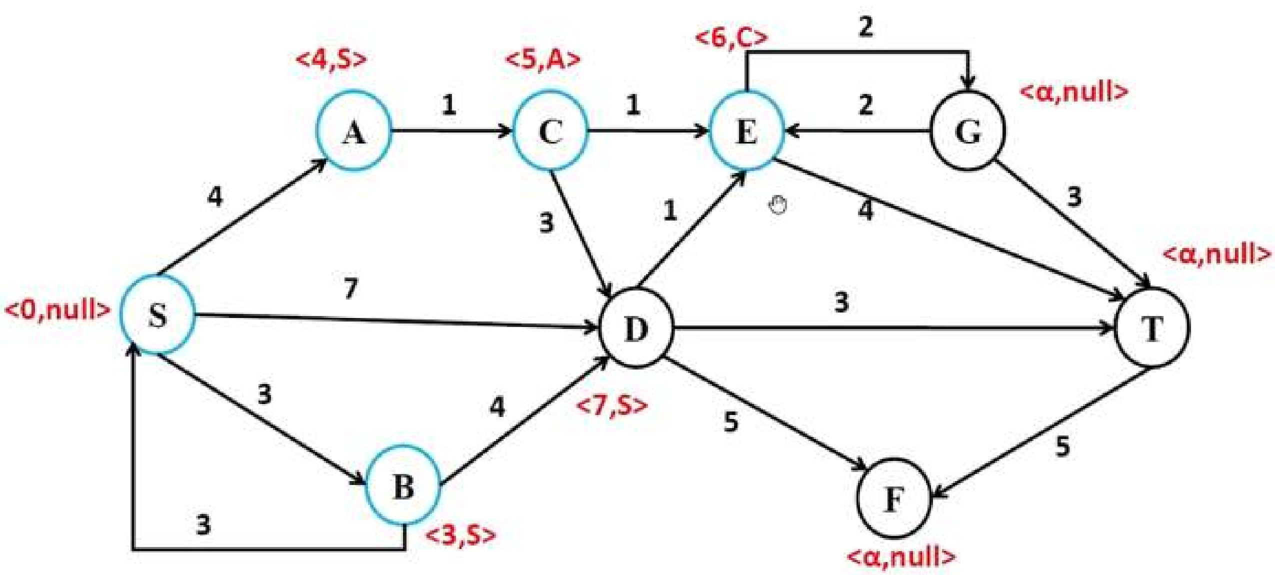
-
- Neighbours of C are
  - E and D
  - Distance of E = 5+1 = 6 < Infinity
  - Setting distance = 6 and previous = C
  - Distance of D = 7, same as previous value, not changing



-

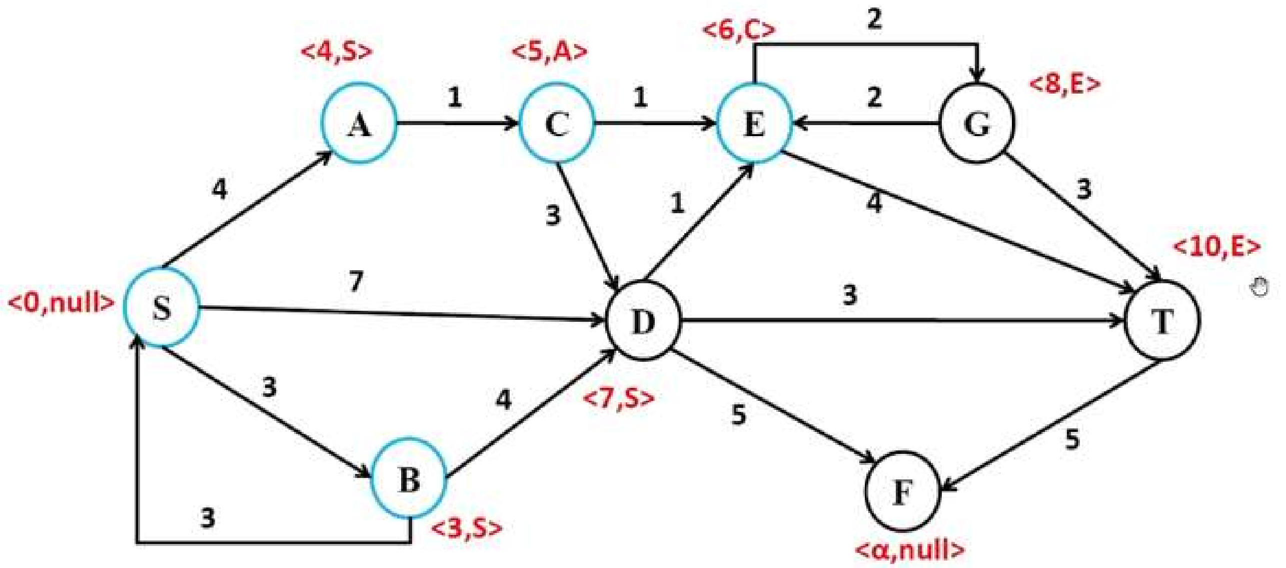
	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
C				5	7	α	α	α	α
					7	6	α	α	α
		S	S	A	S	C			

- The minimum distance among E and D is E with distance 6
- Removing E



- Neighbours of E are G and T
  - Distance of G = 6 + 2 = 8 < infinity
    - Setting distance and previous to 8, E
  - Distance of T = 6+4 < infinity
    - Setting distance and previous = 10,E

Dijkstra's Algorithm : Example

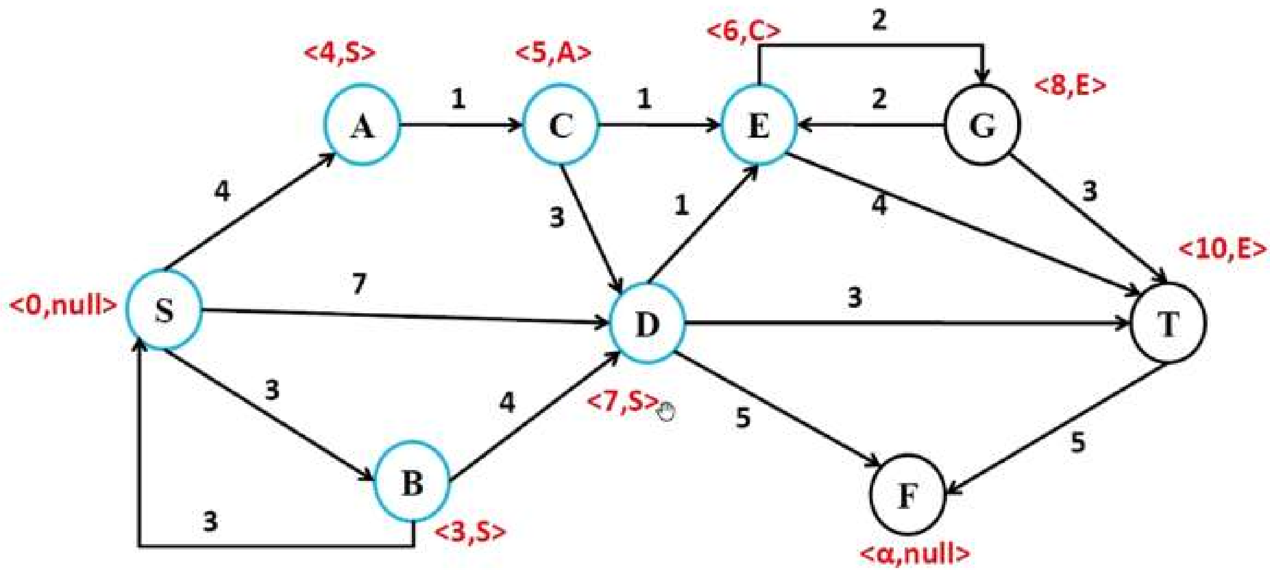


-

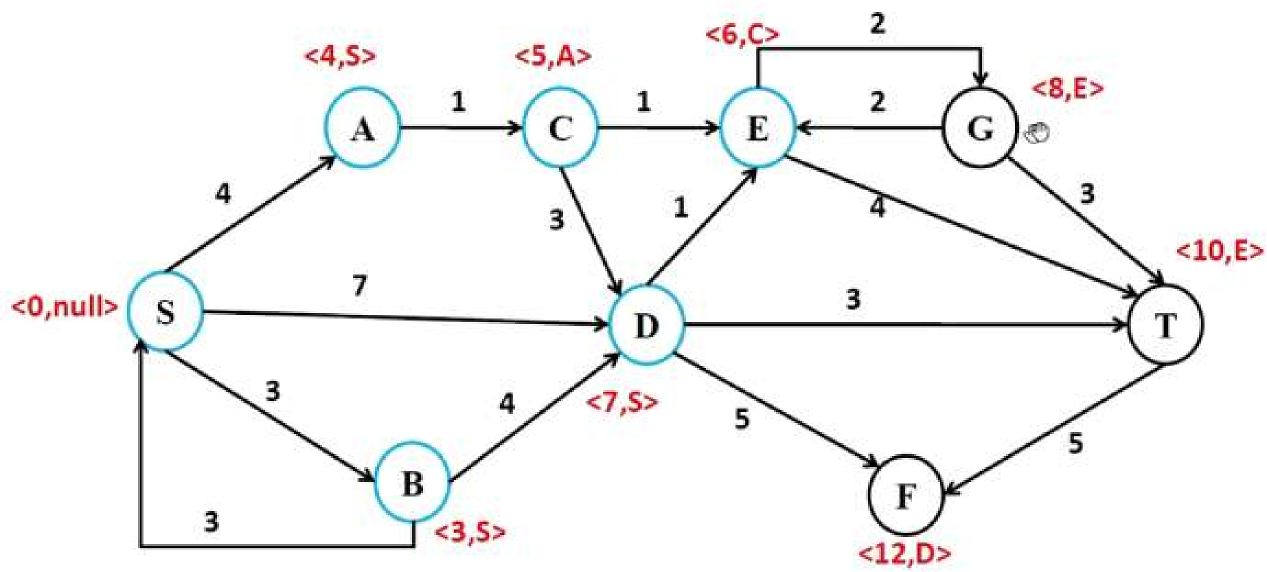
	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
C				5	7	α	α	α	α
E					7	6	α	α	α
					7		α	8	10
		S	S	A	S	C		E	E

-
- Among these nodes, D is the minimum with 7
- Removing Node D

Dijkstra's Algorithm : Example



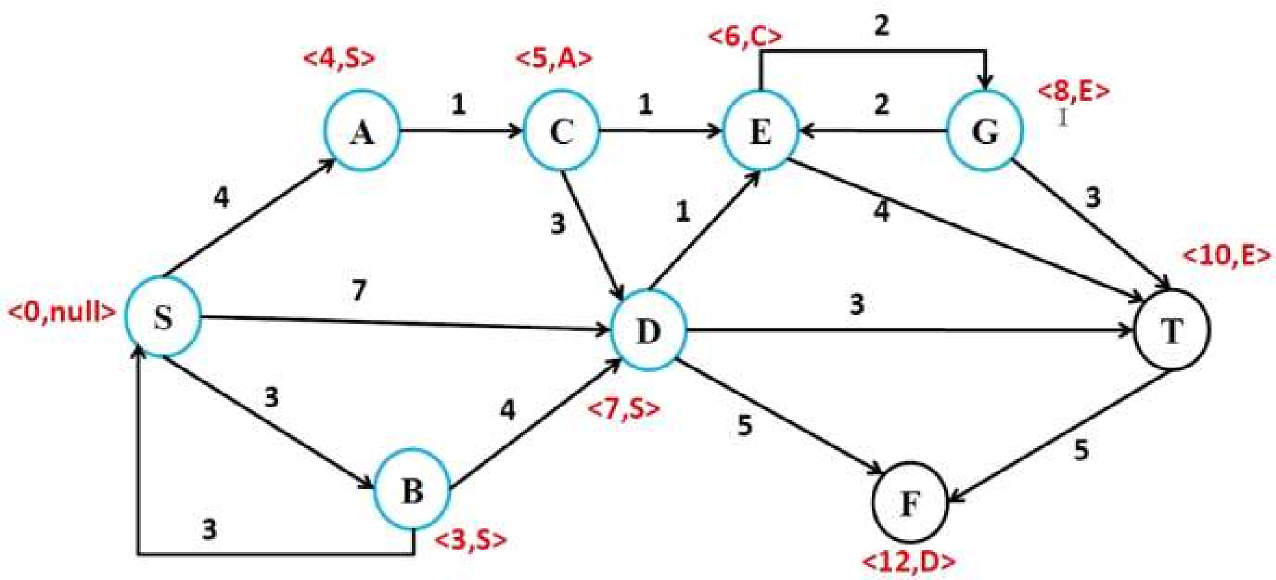
-
- Neighbours of D are
  - T and F
  - Distance of T is 7 + 3 = 10, Which is same as previous distance
  - Distance of F is 7+5 = 12 < infinity
    - Setting distance of F to 12, and previous to D



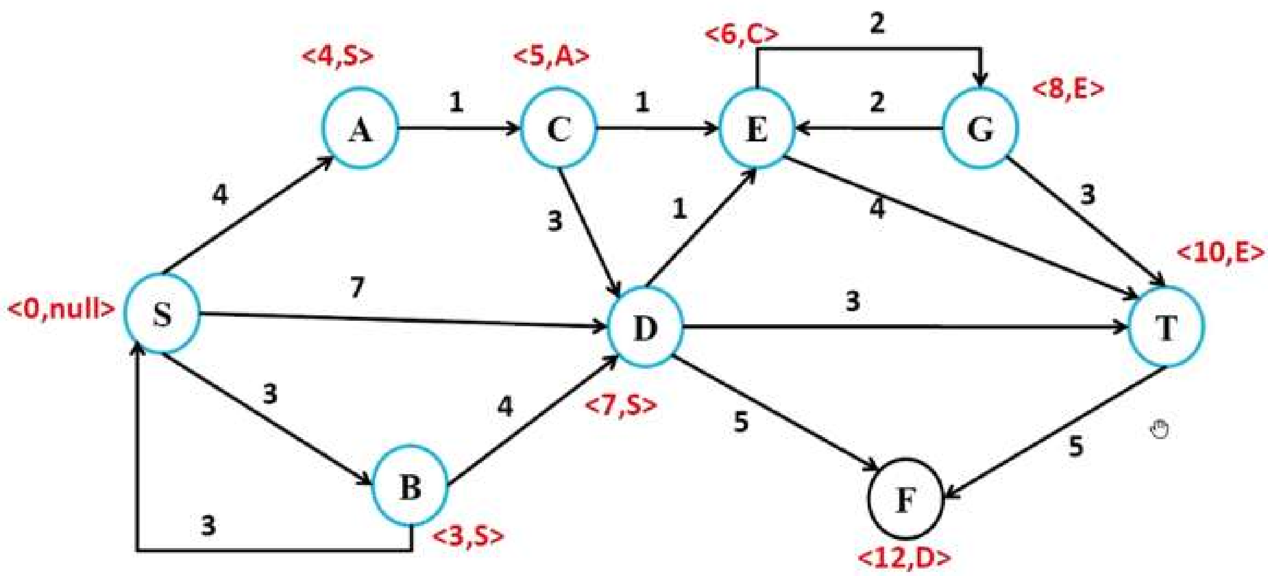
-

	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
C				5	7	α	α	α	α
E					7	6	α	α	α
D					7		α	8	10
							12	8	10
		S	S	A	S	C	D	E	E

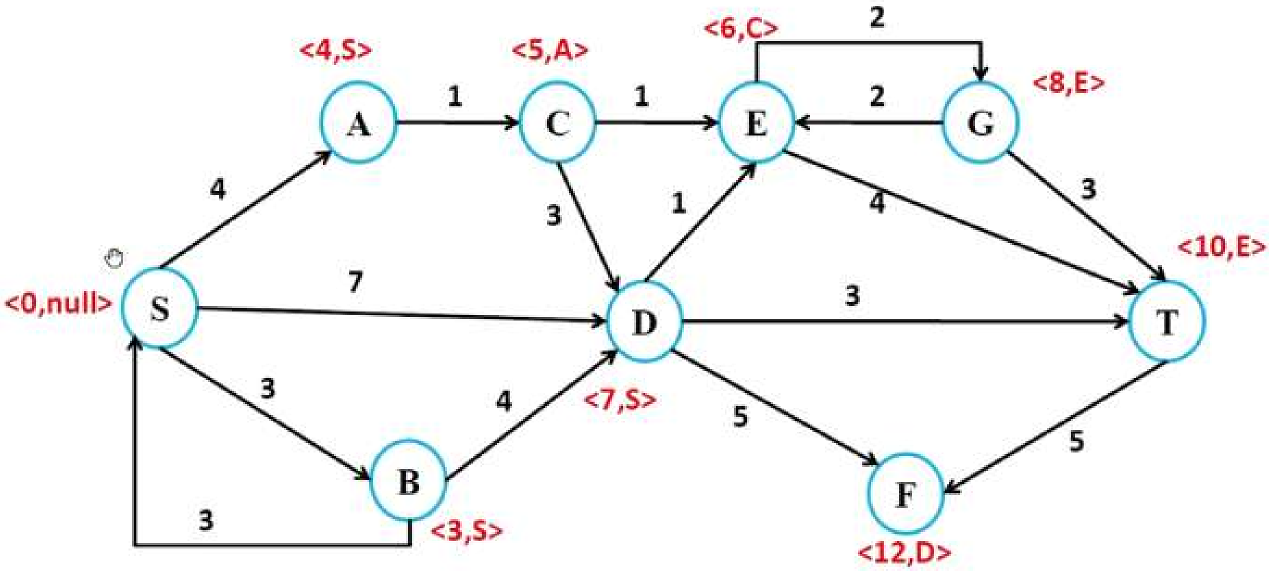
- Among the nodes, G has the least distance with 8
- Removing G



- Neighbours of G are T
  - Distance of T = 8 + 3 =11, Which is more than existing distance, no change required

	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
C				5	7	α	α	α	α
E					7	6	α	α	α
D					7		α	8	10
G							12	8	10
							12		10
		S	S	A	S	C	D	E	E
- Smallest Node is T with 10
- Removing T
- 

- Neighbour of T is F
  - Distance of F is 10 + 5 = 15 which is more than existing distance, so not changing
- Last node is F, Removing the node
- F has no neighbouring nodes



	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
C				5	7	α	α	α	α
E					7	6	α	α	α
D					7		α	8	10
G							12	8	10
T							12		10
F							12		
		S	S	A	S	C	D	E	E

Calculating path and distance from the table

	S	A	B	C	D	E	F	G	T
S	0	α	α	α	α	α	α	α	α
B		4	3	α	7	α	α	α	α
A		4		α	7	α	α	α	α
C				5	7	α	α	α	α
E					7	6	α	α	α
D					7		α	8	10
G							12	8	10
T							12		10
F							12		
		S	S	A	S	C	D	E	E

	Shortest Path	Shortest Distance
S-A	S-A	4
S-B	S-B	3
S-C	S-A-C	5
S-D	S-D	7
S-E	S-A-C-E	6
S-F	S-D-F	12
S-G	S-A-C-E-G	8
S-T	S-A-C-E-T	10

- To get path from S to E
  - Previous of E = C
  - Previous of C = A
  - Previous of A = S
- We get the path = S-A-C-E

Djikstras Algorithm - Analysis

Analysis

Suppose the priority queue is an ordered (by d) linked list.

~~1 building the queue (sorting)~~ ~~$O(V)\log V$~~

1. building the queue (sorting) $O(V\log V)$
2. Each ~~extract~~ EXTRACT_MIN $O(V)$
3. This is done V times so $\Omega(V^2)$
4. Each edge is Relaxed one time $O(E)$
5. Total time : $O(V^2+E) = O(V^2)$

Or.

Q is a min-priority queue

INSERT (line 3) : $|V|$ times.

EXTRACT_MIN (line 5): $|V|$ times

DECREASE_KEY (Implicit in RELAX) : at most $|E|$ times.

Binary min_heap: $O(V)$ for building it, each DECREASE_KEY.

EXTRACT_MIN take time $O(\lg V)$

Total running time is $O((V+E)\lg V)$

Fibonacci heap: running time is $O(V\lg V+E)$.

The ~~all~~ amortized cost of each of the $|V|$ EXTRACT_MIN operations is $O(\lg V)$ and each DECREASE_KEY call takes only $O(1)$ amortized heap