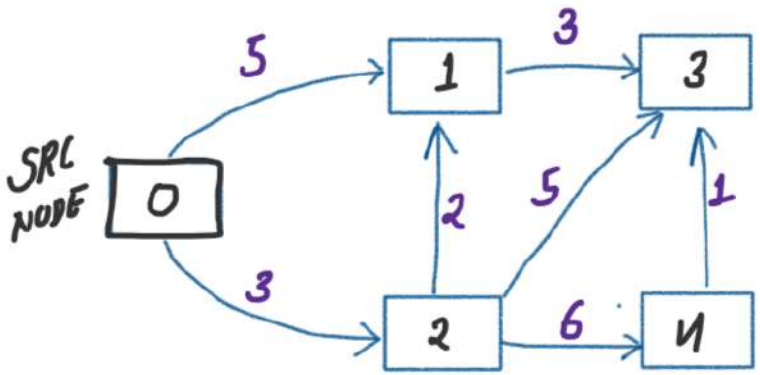
21/02/2024

GRAPHS CLASS - 4

1. Shortest path in a weighted directed graph using DFS

SSSPA: SINGLE SOURCE SHORTEST PATH ALGORITHMS



Output 0 5 3 8 9

Explanation

Paths

I: 0 (dist=0) → 1 (dis=0+5) → 3 (dist=5+3)

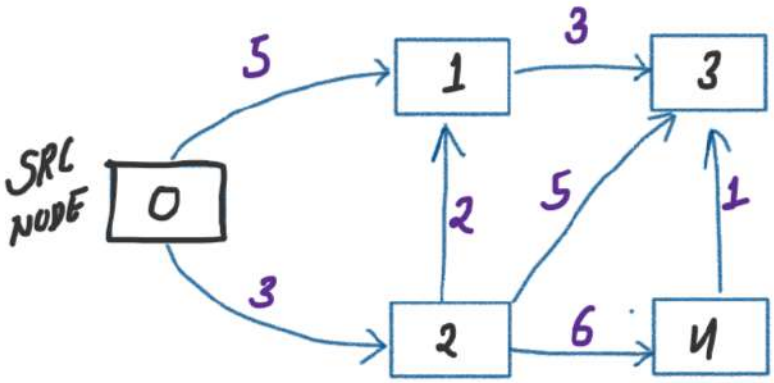
II: 0 → 2 → 4 → 3

III: 0 → 2 → 3

IV: 0 → 2 → 4 → 3

NODE	SHORTEST DISTANCE
0	0
1	5
2	3
3	8
4	9

Find shortest distance from SRC [0] to each nodes [1], [2], [3], [4]



SSSP Algo

Single source shortest path Algorithm

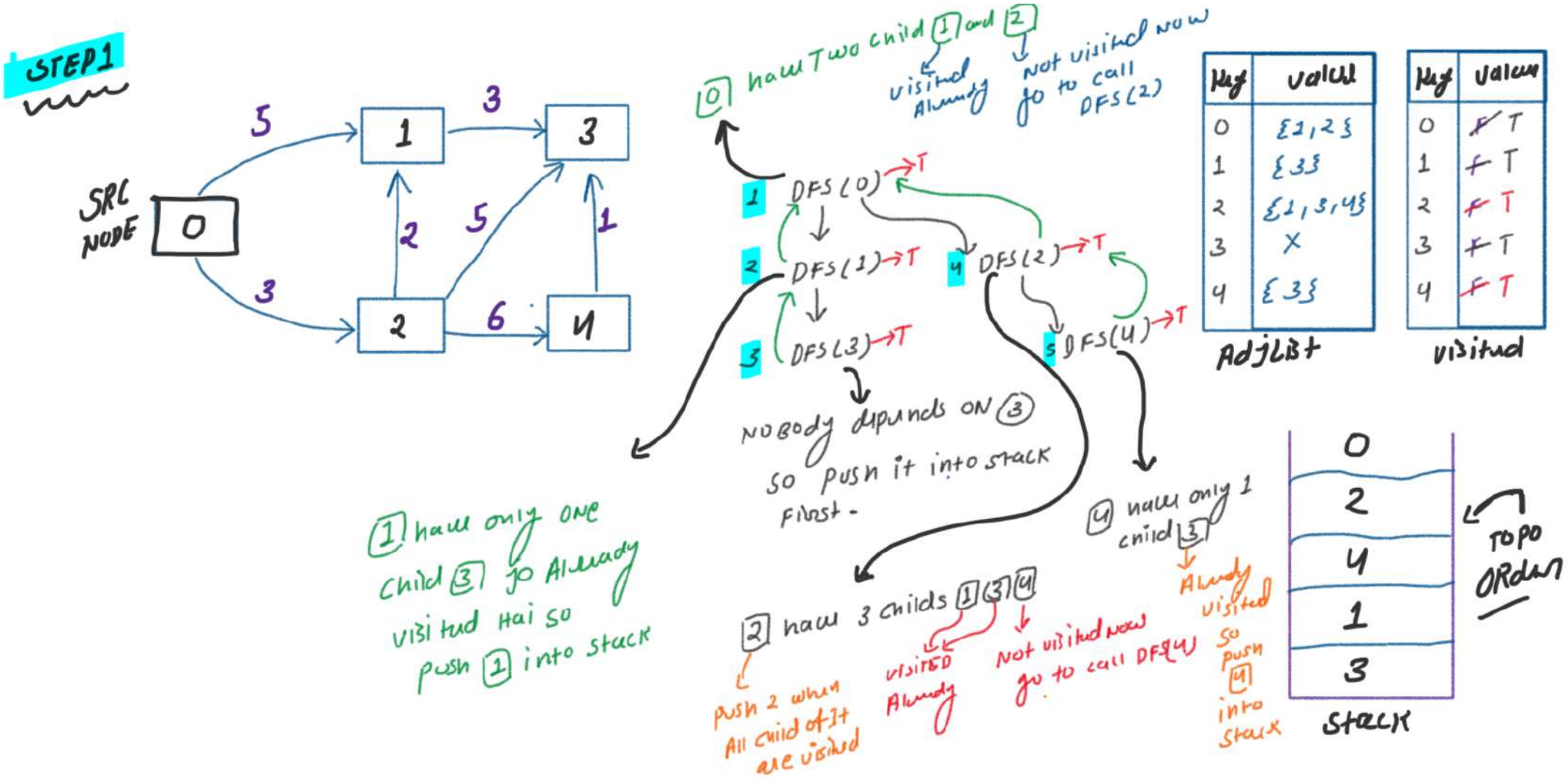
Why TOPOLOGICAL SORT Algorithm Used?

↳ Because yaha par NODES ki BEECH me Dependency create ho rahi hai.

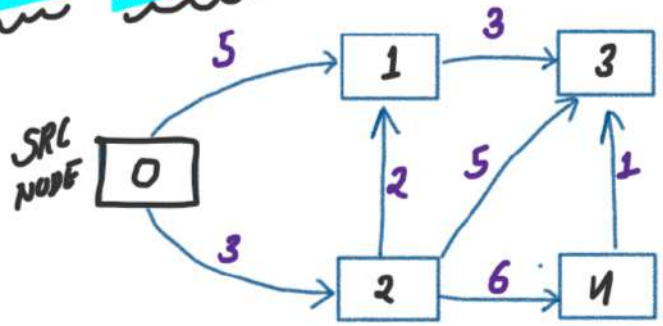
STEP1 Create TOPOLOGICAL ORDER using DFS.

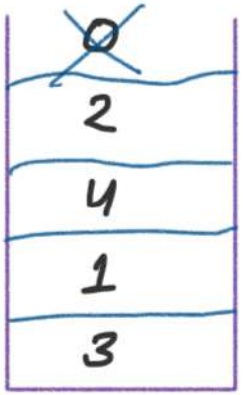
STEP2 STORE All shortest distances using TOPOLOGICAL ORDER.

STEP3 Print All shortest Distances


STEP 1
SRC NODE
0
1
2
3
4
5
3
2
5
1
3
6
0 have Two child 1 and 2
visited Already
Not visited now go to call DFS(2)
DFS(0)→T
DFS(1)→T
DFS(2)→T
DFS(3)→T
DFS(4)→T
NOBody dipends ON 3
So push it into stack First -
1 have only ONE Child 3 jo Already visited Hai so push 1 into stack
2 have 3 childs 1 3 4
visited Already
Not visited now go to call DFS(4)
Push 2 when All child of it are visited
4 have only 1 child 3
Already visited So push 4 into stack
Key
Value
0 {1,2}
1 {3}
2 {1,3,4}
3 X
4 {3}
AdjList
Key
Value
0 F T
1 F T
2 F T
3 F T
4 F T
visited
0
2
4
1
3
TOPO ORder
stack

STEP 1 Situation 1

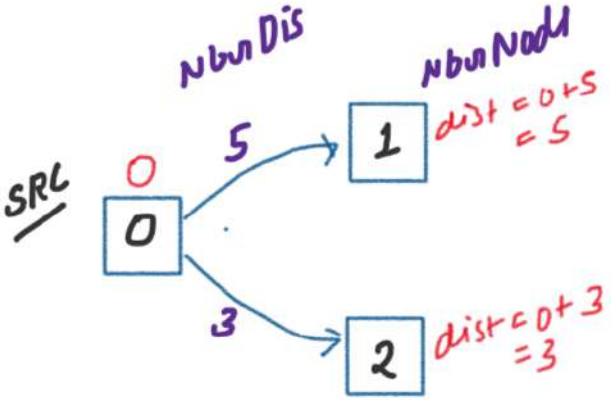




Stack (TopoOrder)

Key	value
0	{{1,5}, {2,3}}
1	{{3,3}}
2	{{1,2}, {3,5}, {4,6}}
3	{}
4	{{3,1}}

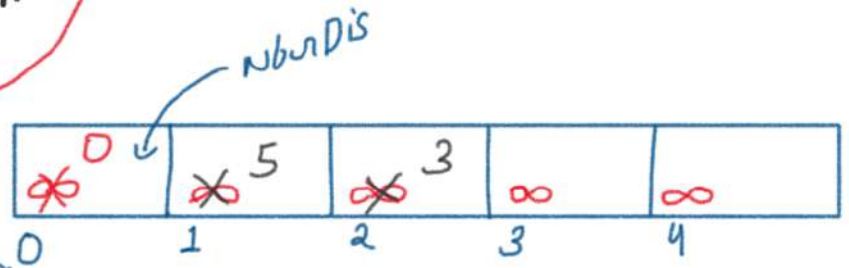
AdjList



(A) goto stack TOP Element which is SRC

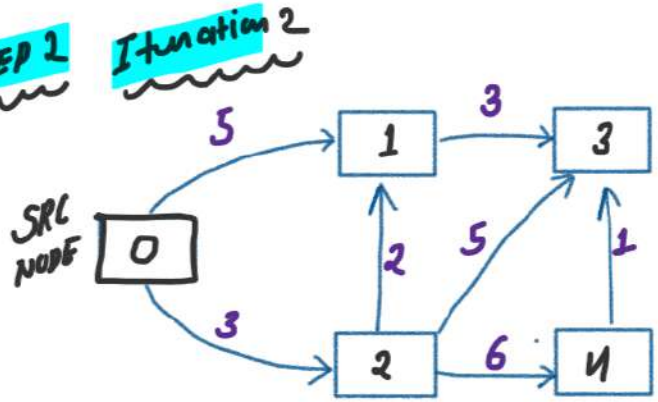
(B) goto AdjList to get All child of SRC

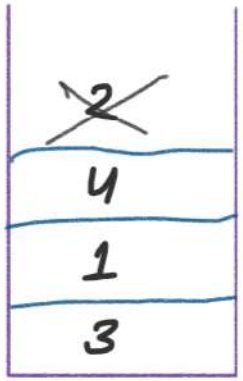
(C) store Distance of NbrNodi when dist[NbrNodi] > NbrDist + Dist[SRC]



NbrNodi Distance Array to store minimum Distance of Each node from SRC NODE

STEP 1 Ituration 2

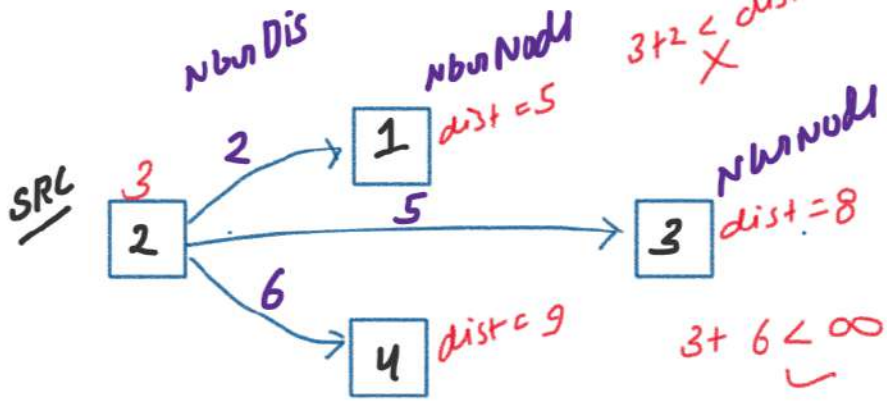


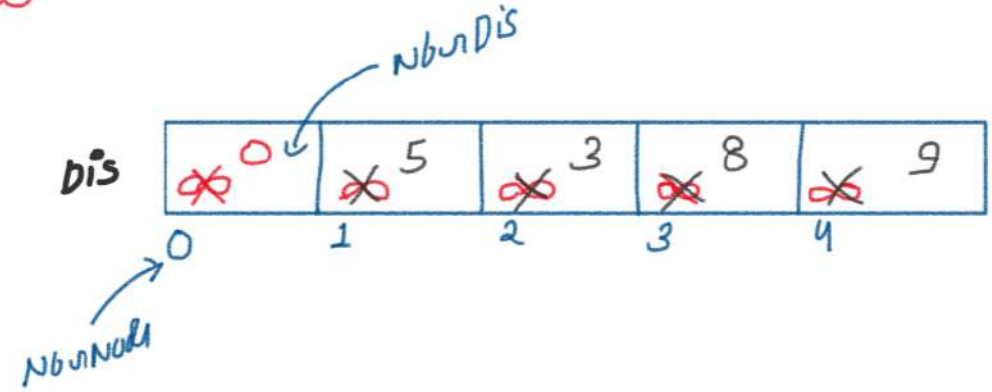


Stack (TopoOrder)

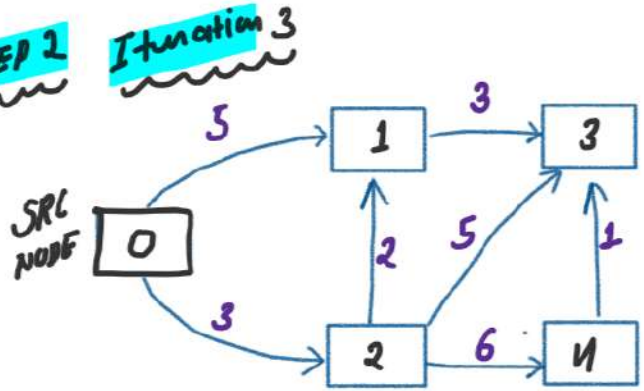
Key	value
0	{{1,5}, {2,3}}
1	{{3,3}}
2	{{1,2}, {3,5}, {4,6}}
3	{}
4	{{3,1}}

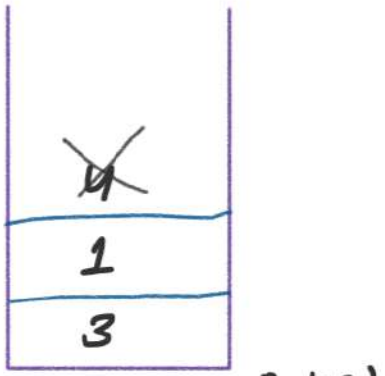
AdjList





STEP 2 Situation 3

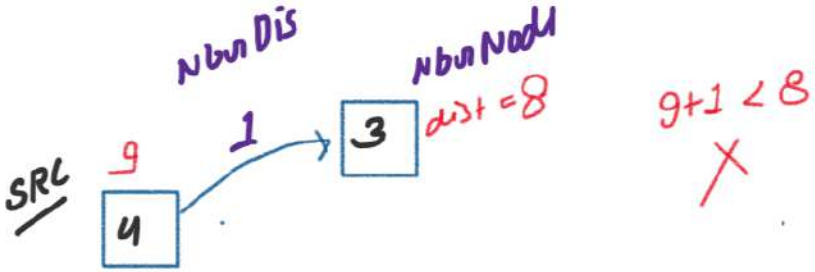




Stack (TopoOrder)

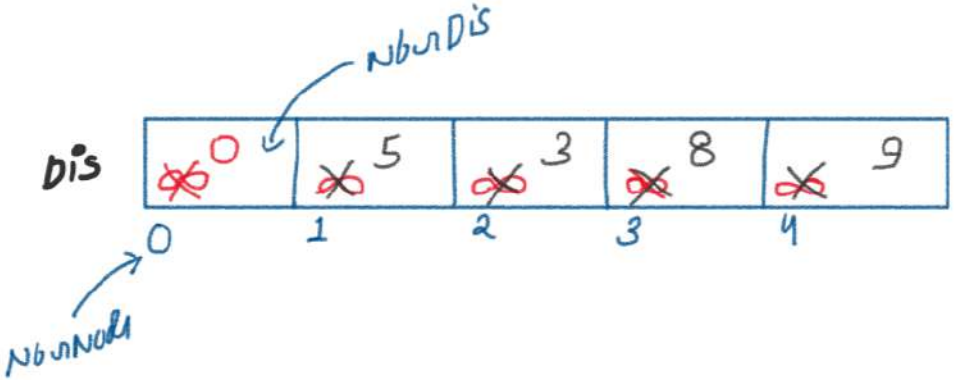
Key	value
0	{ {1,5}, {2,3} }
1	{ {3,3} }
2	{ {1,2}, {3,5}, {4,6} }
3	{ }
4	{ {3,1} }

AdjList

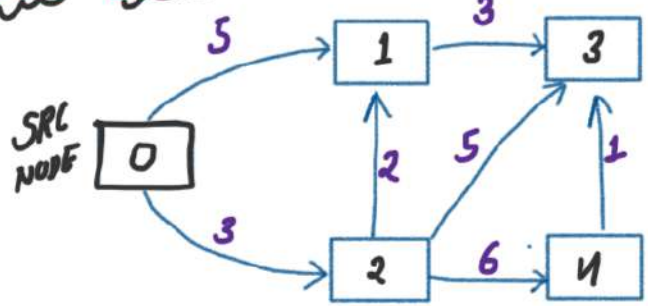


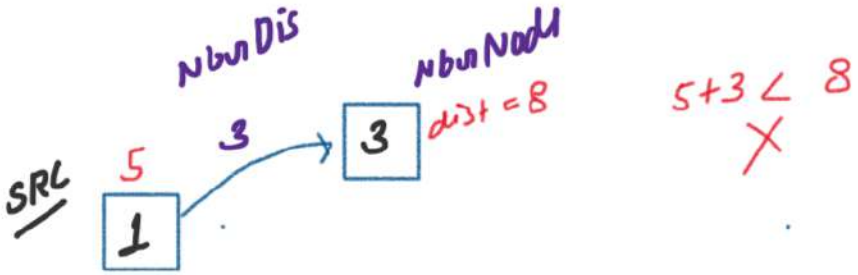
9+1 < 8

X

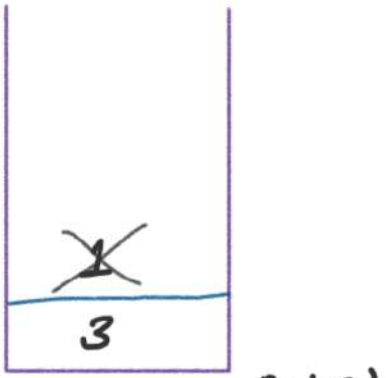


STEP 1 Intuation





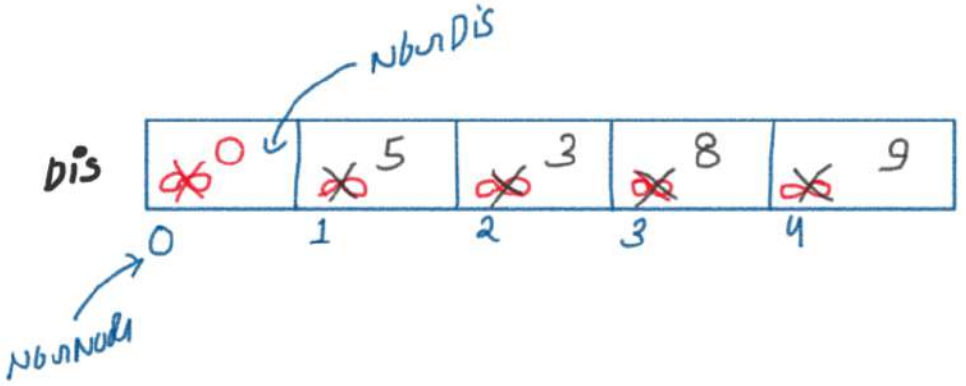
5+3 < 8 ✗



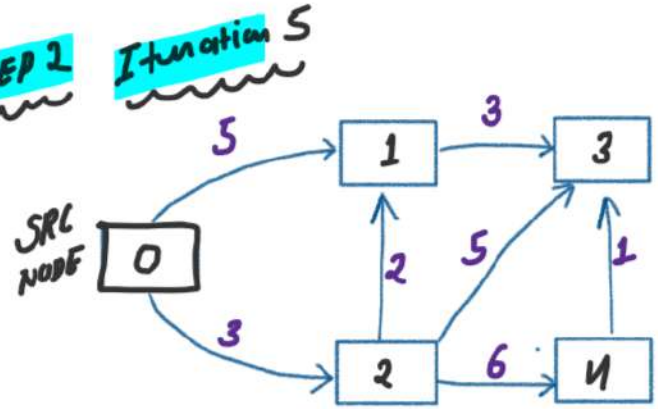
Stack (TopoOrder)

Key	value
0	{ {1,5}, {2,3} }
1	{ {3,3} }
2	{ {1,2}, {3,5}, {4,6} }
3	{ }
4	{ {3,1} }

AdjList



STEP 1 Situation 5

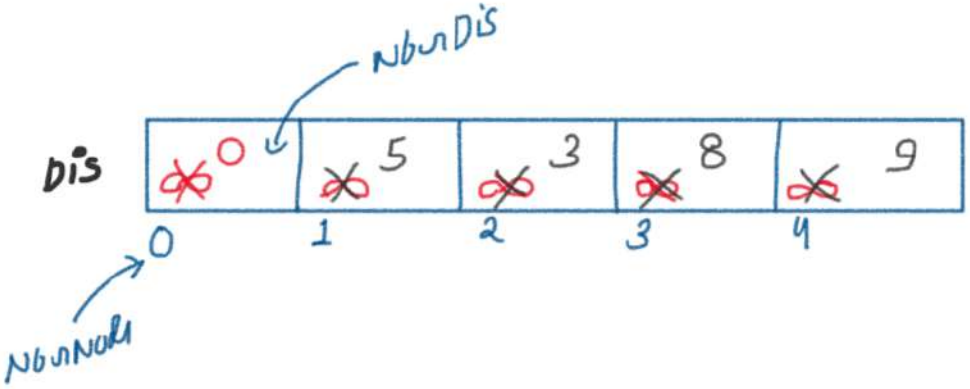


Key	value
0	{ {1,5}, {2,3} }
1	{ {3,3} }
2	{ {1,2}, {3,5}, {4,6} }
3	{ }
4	{ {3,1} }

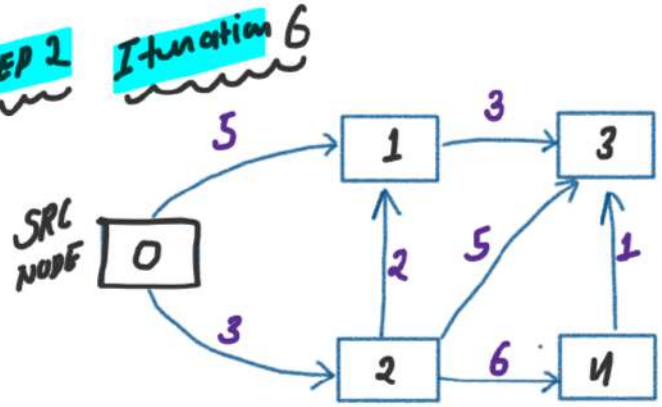
AdjList

SRC 8

[3] does not have any child / Nbr Node



STEP 1 Situation 6



Stack (TopoOrder)

Key	value
0	{{1,5}, {2,3}}
1	{{3,3}}
2	{{1,2}, {3,5}, {4,6}}
3	{}
4	{{3,1}}

AdjList

STEP 3

Print This ARRAY

Output

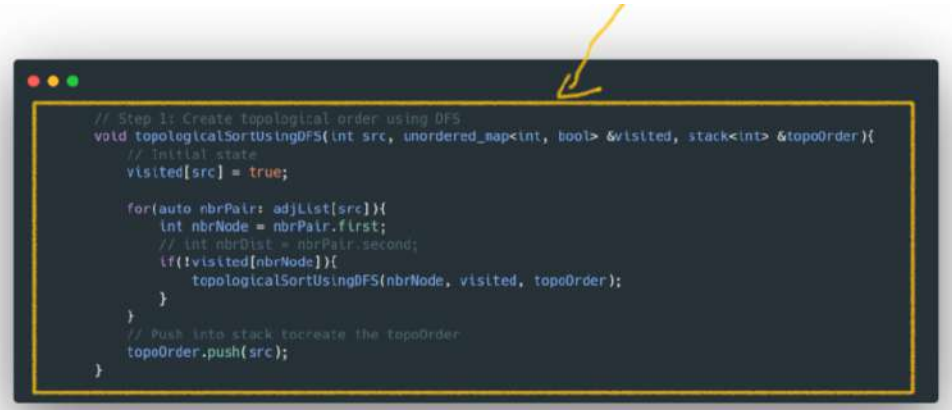
dis

0	5	3	8	9
0	1	2	3	4

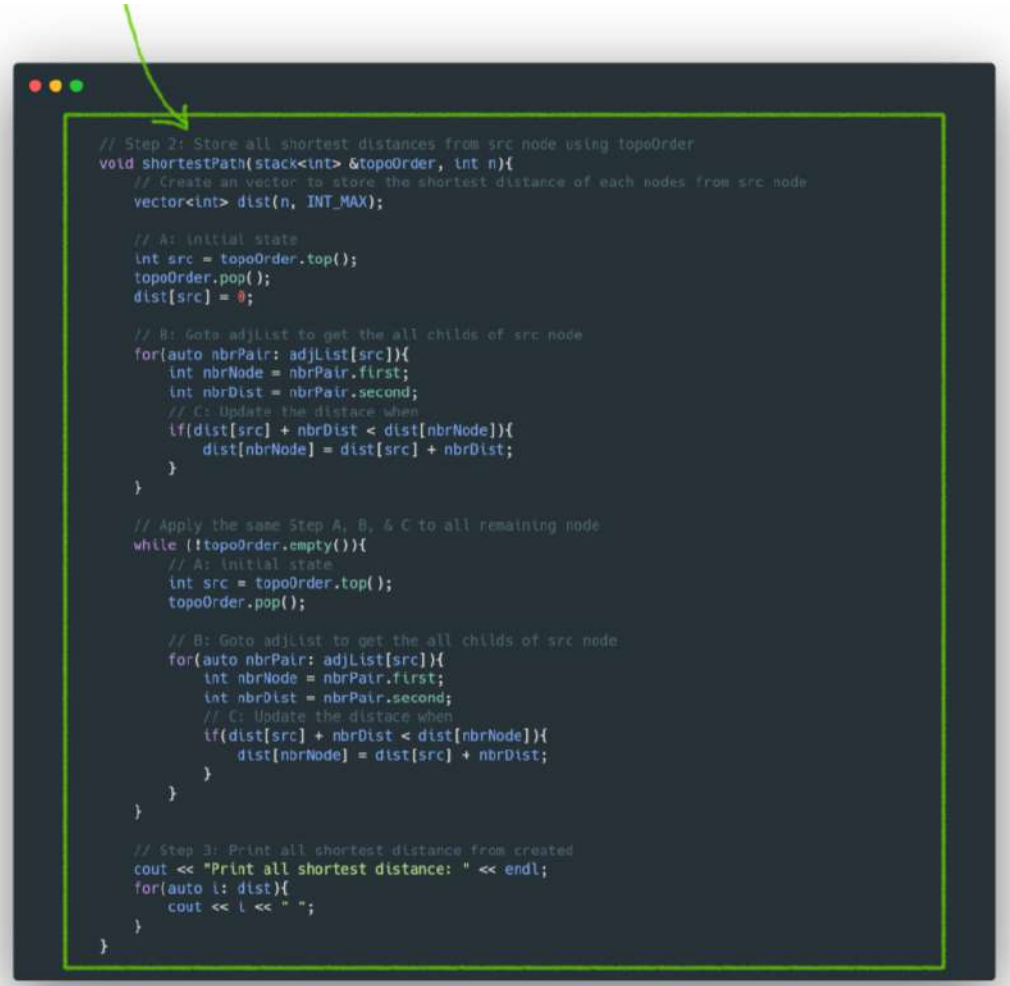
```cpp
// 1. Shortest path in a weighted directed graph using DFS (SSSP Algorithms)

#include<iostream>
#include<stack>
#include<vector>
#include<unordered_map>
#include<limits.h>
#include<list>

using namespace std;

class Graph
{
    public:
        unordered_map<int, list<pair<int, int>>> adjList;

        void addEdges(int u, int v, int wt, int direction){
            if(direction == 1){
                // Directed Graph
                adjList[u].push_back({v,wt});
            }
            else{
                // Undirected Graph
                adjList[u].push_back({v,wt});
                adjList[v].push_back({u,wt});
            }
        }

        // Step 1: Create topological order using DFS
        void topologicalSortUsingDFS(int src, unordered_map<int, bool> &visited, stack<int> &topoOrder){
            ....
        }

        // Step 2: Store all shortest distances from src node using topoOrder
        void shortestPath(stack<int> &topoOrder, int n){
            ....
        }
};

int main(){
    Graph g;
    g.addEdges(0,1,5,1);
    g.addEdges(0,2,3,1);
    g.addEdges(2,1,2,1);
    g.addEdges(1,3,3,1);
    g.addEdges(2,3,5,1);
    g.addEdges(2,4,6,1);
    g.addEdges(4,3,1,1);

    int src = 0;
    unordered_map<int, bool> visited;
    stack<int> topoOrder;
    g.topologicalSortUsingDFS(src, visited, topoOrder);

    int n = 5;
    g.shortestPath(topoOrder, n);

    return 0;
}
```

Time and space complexity

?



```cpp
// Step 1: Create topological order using DFS
void topologicalSortUsingDFS(int src, unordered_map<int, bool> &visited, stack<int> &topoOrder){
    // Initial state
    visited[src] = true;

    for(auto nbrPair: adjList[src]){
        int nbrNode = nbrPair.first;
        // int nbrDist = nbrPair.second;
        if(!visited[nbrNode]){
            topologicalSortUsingDFS(nbrNode, visited, topoOrder);
        }
    }
    // Push into stack tocreate the topoOrder
    topoOrder.push(src);
}
```



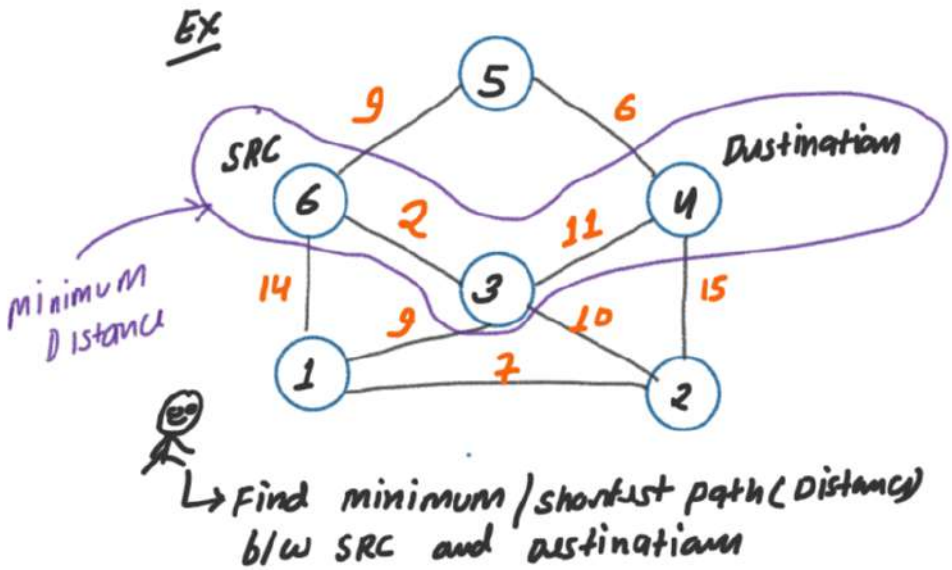
```cpp
// Step 2: Store all shortest distances from src node using topoOrder
void shortestPath(stack<int> &topoOrder, int n){
    // Create an vector to store the shortest distance of each nodes from src node
    vector<int> dist(n, INT_MAX);

    // A: initial state
    int src = topoOrder.top();
    topoOrder.pop();
    dist[src] = 0;

    // B: Goto adjList to get the all childs of src node
    for(auto nbrPair: adjList[src]){
        int nbrNode = nbrPair.first;
        int nbrDist = nbrPair.second;
        // C: Update the distace when
        if(dist[src] + nbrDist < dist[nbrNode]){
            dist[nbrNode] = dist[src] + nbrDist;
        }
    }

    // Apply the same Step A, B, & C to all remaining node
    while (!topoOrder.empty()){
        // A: initial state
        int src = topoOrder.top();
        topoOrder.pop();

        // B: Goto adjList to get the all childs of src node
        for(auto nbrPair: adjList[src]){
            int nbrNode = nbrPair.first;
            int nbrDist = nbrPair.second;
            // C: Update the distace when
            if(dist[src] + nbrDist < dist[nbrNode]){
                dist[nbrNode] = dist[src] + nbrDist;
            }
        }
    }

    // Step 3: Print all shortest distance from created
    cout << "Print all shortest distance: " << endl;
    for(auto i: dist){
        cout << i << " ";
    }
}
```

2. Shortest path/distance in a weighted undirected graph

Dijkstra Algorithm: Source, Destination, Minimum

What is Dijkstra Algorithm and where to use it?

Dijkstra's Algorithm is a popular graph search algorithm used to find the **shortest** path between a **source** node and all other nodes in a **weighted**, **directed** or **undirected graph**.



Dijkstra's Algorithm Applications in real life:

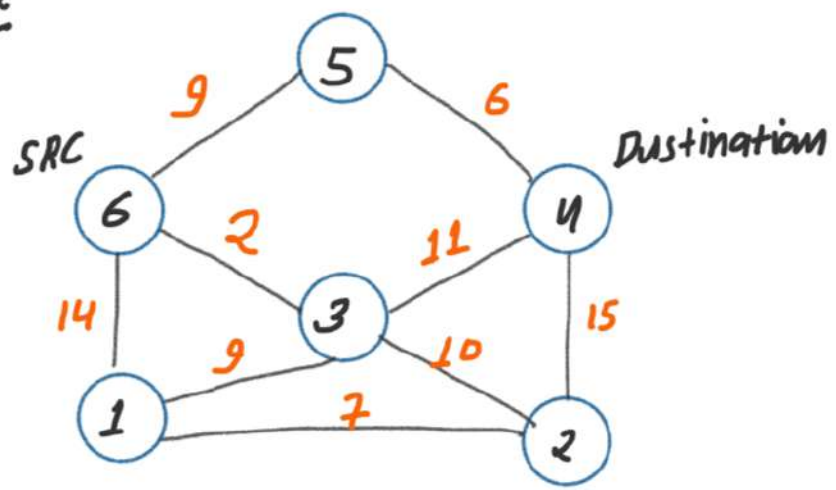
1. To find the shortest path from source to destination
2. In social networking applications
3. In a telephone network
4. To find the locations in the map

Dijkstra's Algorithm Complexity:

Time Complexity: O(E Log V)
Space Complexity: O(V)

where, E is the number of edges and V is the number of vertices

Ex



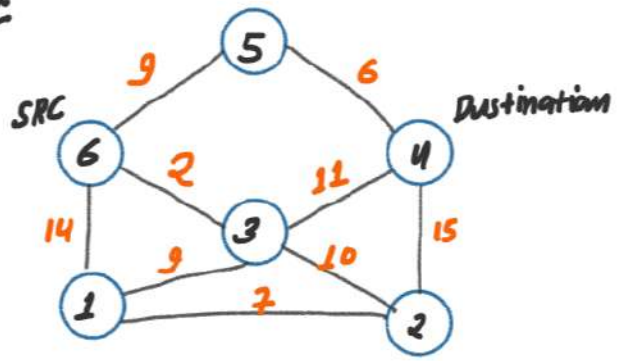
↳ Find minimum / shortest path (distance) b/w SRC and destination

Output = 13

Input ⇒ SRC = 6
Dest = 4

Dijkstra Algo with SET DATA STRUCTURE

Ex



Dist ARRAY

↳ This Array is used to store the total shortest Distance from src to All nodes including src and Destination node.

SET

Set < pair < int, int > > st;

Distance SRC

Why use set?

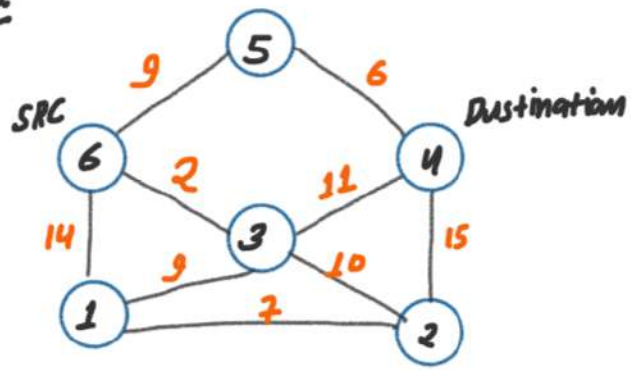
↳ Kyunki Hum shortest distance pahle chahiye from SRC TO DUST.

Why Distance is first in the set's pair?

↳ Kyunki Hum set ko Distance wise sorted order me chahte hai

jiske dwara Hume shortest Distance pahle mil jayegga from src to dust.

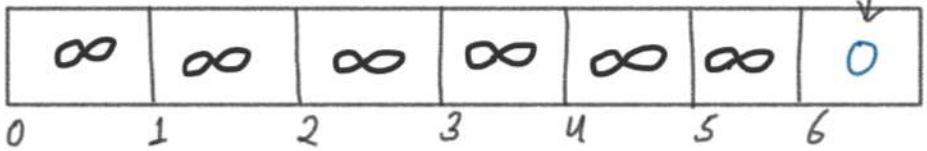
Ex

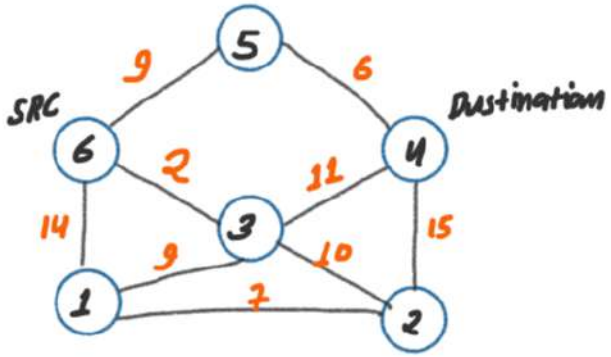


initial State

St.push({ dist, src });
0 6

Dist[src] = 0;
6





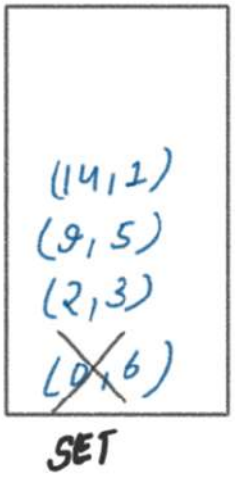
Situation1

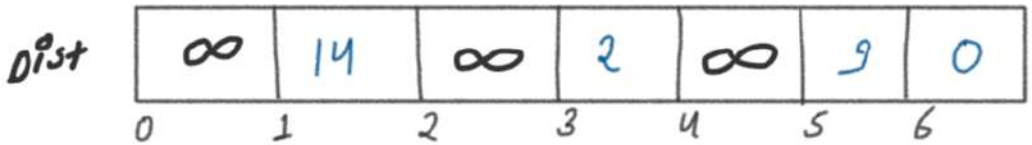
(A) Select shortest distance Node from the set.

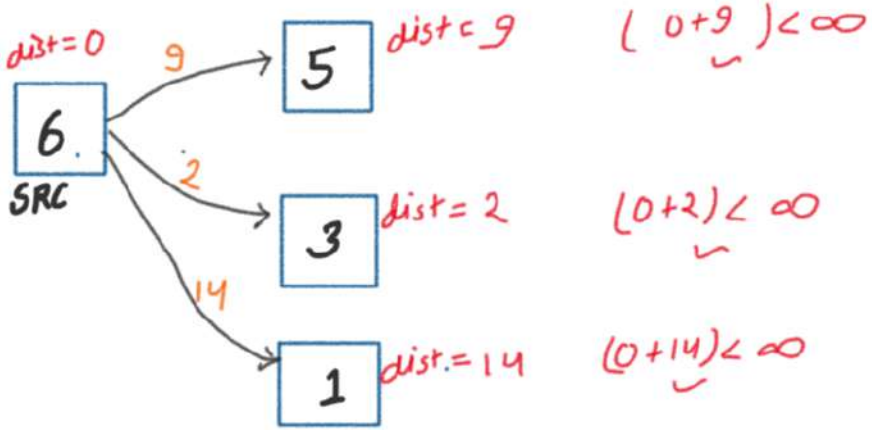
(0,6)

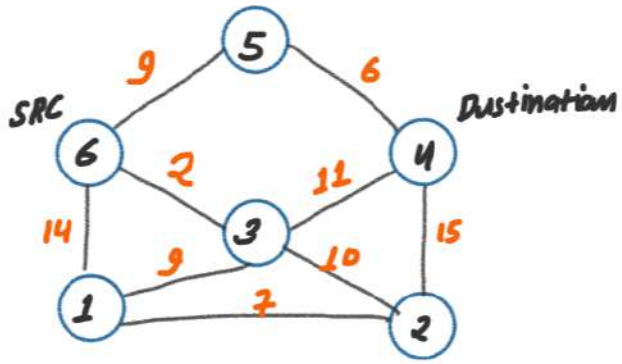
(B) Delete the selected node from set

(C) Update New Distance for New Node and Create New entry for New Distance in set.









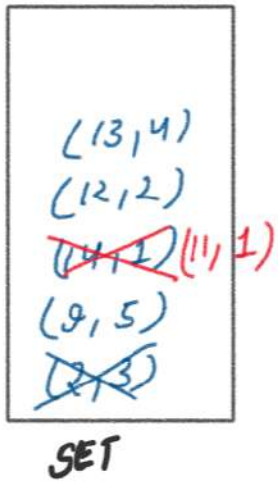
Situation 2

(A) Select shortest distance Node from the set.

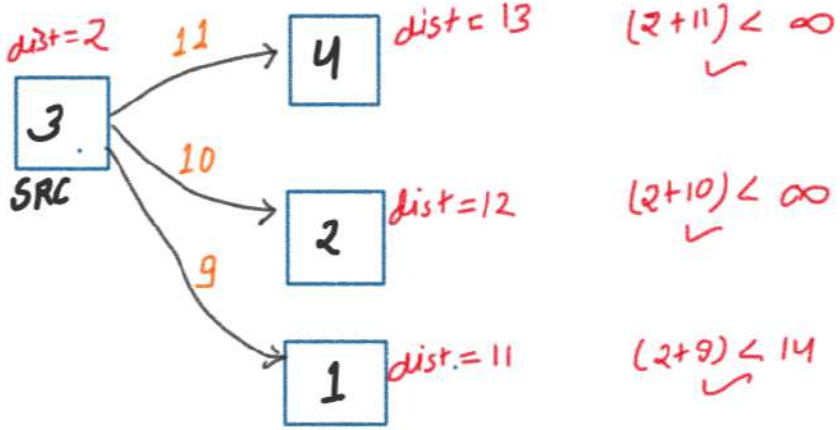
(2, 3)

(B) Delete selected node from set

(C) Update New Distance for new Node and Create New entry for New Distance in set.

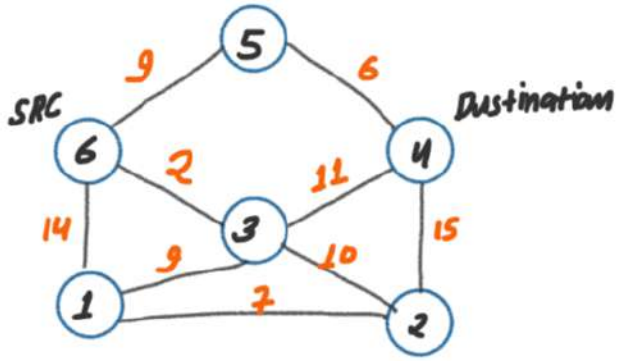


SET



Dist

∞	11	12	2	13	9	0
0	1	2	3	4	5	6



Iteration 3

(A) Select shortest distance Node from the set.

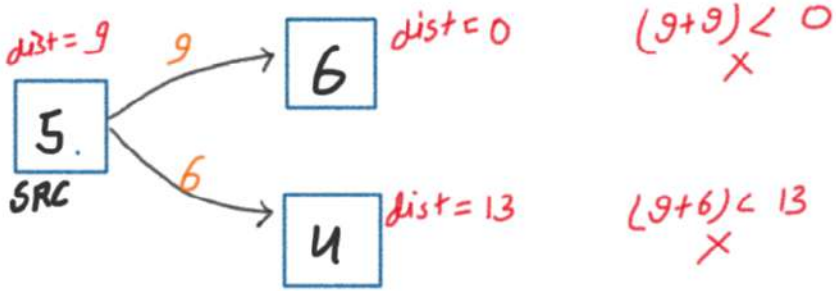
(9, 5)

(B) Delete the selected node from set

(C) Update new distance for new Node and create new entry for new Distance in set.

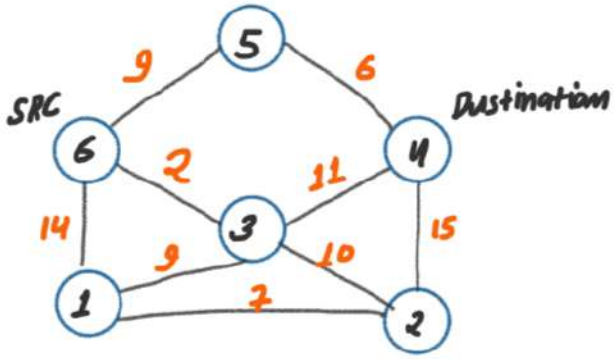
SET

(13, 4)
(12, 2)
(11, 1)
~~(9, 5)~~



Dist

0	1	2	3	4	5	6
∞	11	12	2	13	9	0



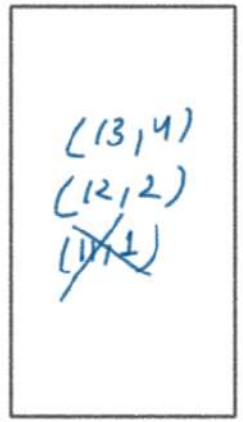
Situation 4

(A) Select shortest distance Node from the Set.

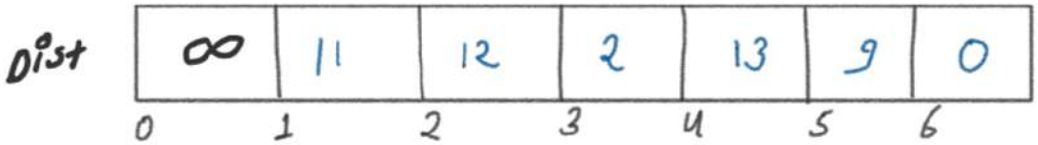
(11, 1)

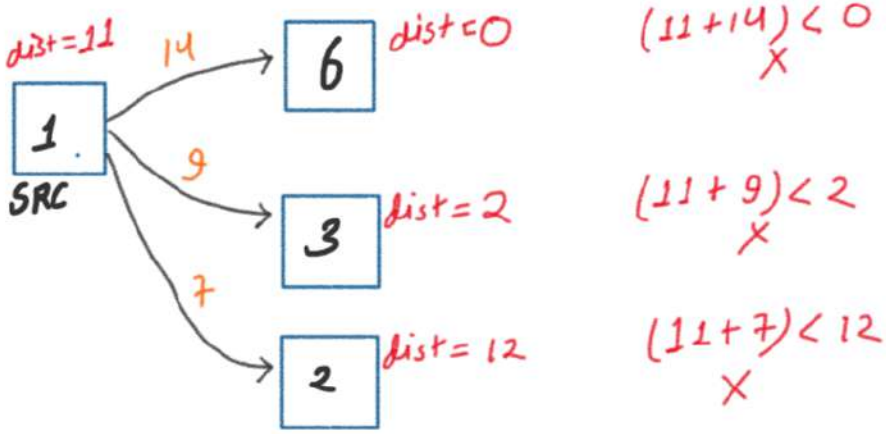
(B) Delete selected node from Set

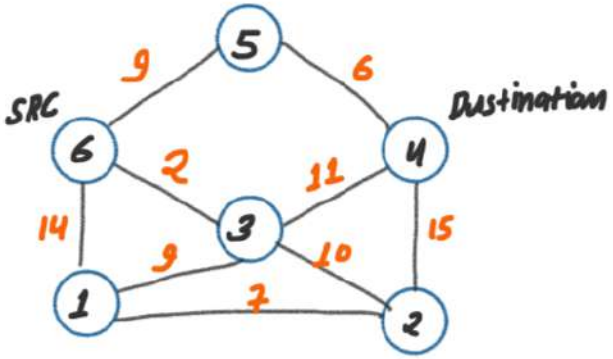
(C) Update New Distance for Node and Create New Entry for New Distance in Set.



SET







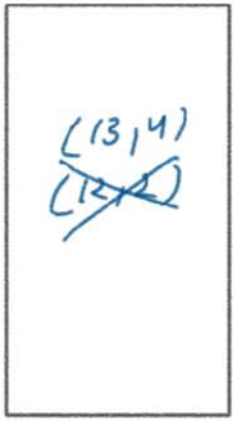
Situations

(A) Select shortest distance Node from the Set.

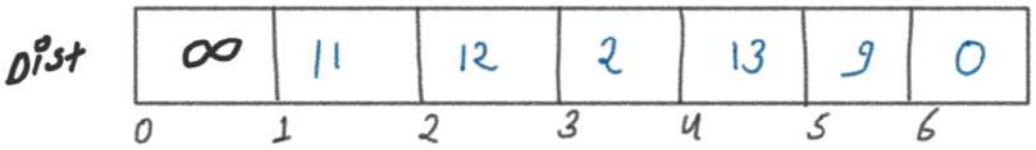
(12, 2)

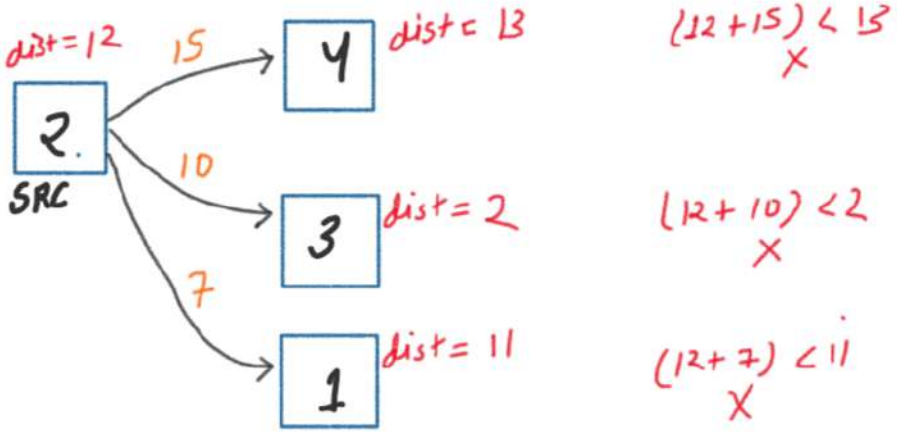
(B) Delete selected node from Set

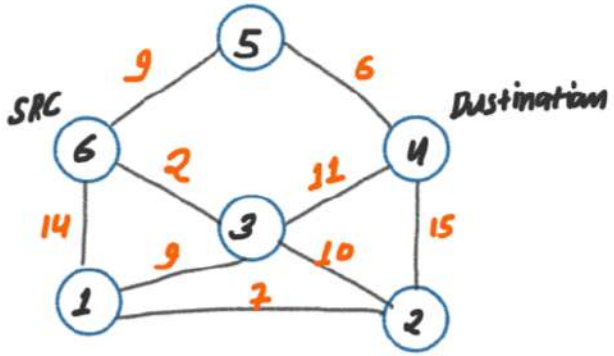
(C) Update New Distance for Node and Create New entry for New Distance in Set.



SET







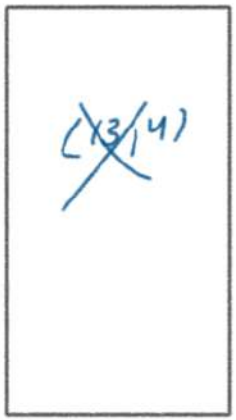
Situation 6

(A) Select shortest distance Node from the Set.

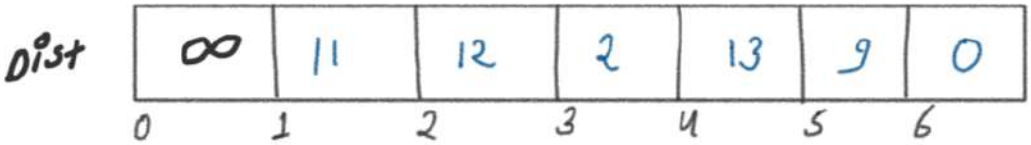
(13, 4)

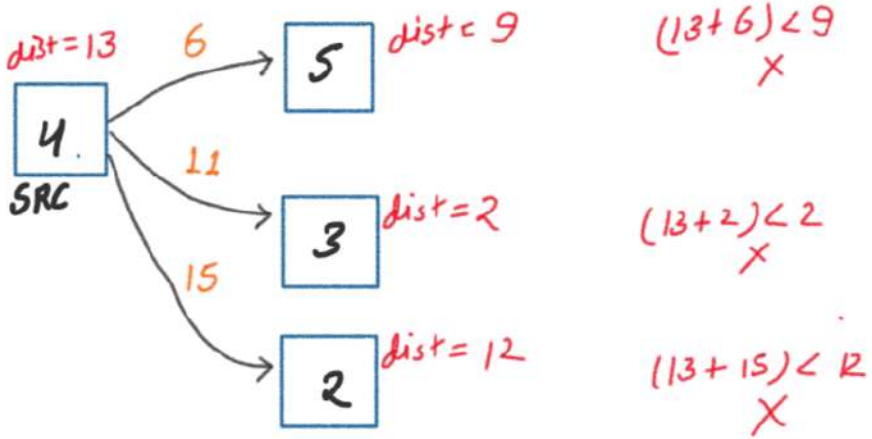
(B) Delete selected node from Set

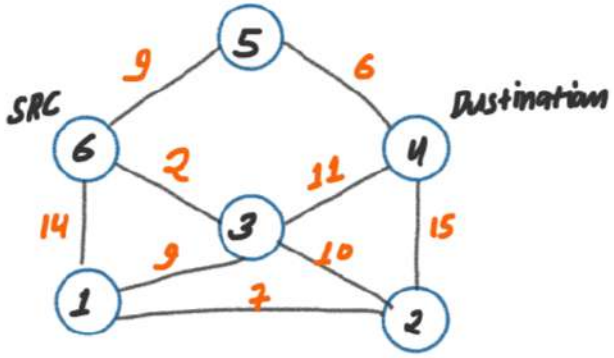
(C) Update New Distance for Node Node and Create New Entry for New Distance in Set.



SET







Iteration 7

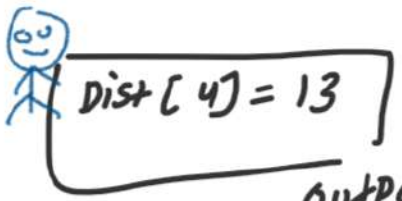
Now Set is Empty

ISKA matlb HUMNE All shortest Distance find kar liya Hai from SRC to All Nodes Including SRC and distination node

SET

Dist

0	1	2	3	4	5	6
∞	11	12	2	13	9	0



Output

This is the shortest path from 6 to 4

```cpp
// 2. Shortest path in a weighted undirected graph using Dijkstra Algorithm

#include<iostream>
#include<stack>
#include<vector>
#include<unordered_map>
#include<limits.h>
#include<list>
#include<set>

using namespace std;

class Graph
{
    public:
        unordered_map<int, list<pair<int, int>>> adjList;

        void addEdges(int u, int v, int wt, int direction){
            if(direction == 1){
                // Directed Graph
                adjList[u].push_back({v,wt});
            }
            else{
                // Undirected Graph
                adjList[u].push_back({v,wt});
                adjList[v].push_back({u,wt});
            }
        }

        void dijkstraShortestDistance(int n, int src, int dest) {.....}
};

int main(){
    Graph g;
    g.addEdges(1,6,14,0);
    g.addEdges(1,3,9,0);
    g.addEdges(1,2,7,0);
    g.addEdges(2,3,10,0);
    g.addEdges(2,4,15,0);
    g.addEdges(3,4,11,0);
    g.addEdges(6,3,2,0);
    g.addEdges(6,5,9,0);
    g.addEdges(5,4,6,0);

    g.printAdjList();

    int n = 6;
    int src = 6;
    int dest = 4;
    g.dijkstraShortestDistance(n, src, dest);

    return 0;
}
```

```cpp
void dijkstraShortestDistance(int n, int src, int dest) {
    vector<int> dist(n+1, INT_MAX);
    set<pair<int,int>> st;

    // Initial state
    st.insert({0,src});
    dist[src] = 0;

    // Distance updation logic
    while (!st.empty()){
        // A. select shortest distance node from the set
        auto topElement = st.begin();
        pair<int, int> topPair = *topElement;
        int topDist = topPair.first;
        int topNode = topPair.second;
        // B. delete the selected node from the set
        st.erase(st.begin());

        // C. update new distance for nbr node and
        // create the new entry for the new distance in set
        for(auto nbrPair: adjList[topNode]) {
            int nbrNode = nbrPair.first;
            int nbrDist = nbrPair.second;

            // Now check the distance: ki mujhe new distance update krna hai ya nhi
            if(topDist + nbrDist < dist[nbrNode]) {
                // Found the new distance

                // 📝 Update the new entry to the set
                // Find previousEntry and delete it
                auto previousEntry = st.find({dist[nbrNode], nbrNode});
                if(previousEntry != st.end()) {
                    // previousEntry present in set then delete it
                    st.erase(previousEntry);
                }

                // Update dist array for new distance
                dist[nbrNode] = topDist + nbrDist;
                // Create a new entry in the set for new distance
                st.insert({dist[nbrNode], nbrNode});
            }
        }
    }
    cout << "Shortest Distance from " << src << " Node to " << dest << " Node: " <<
dist[dest] << endl;
}
```

OUTPUT FOR OUR GIVEN EXAMPLE

```
Shortest Distance from 6 Node to 4 Node: 13
```

DIJKSTRA Limitation

REMBER these points

1. Not used for negative weight
2. Not used for unreachable node
3. Not used for negative cycle of graph

SET DATA STRUCTURE

- *What is set data structure?*
- *How set data structure work?*
- *What is the time complexity of set data structure operation?*
  - *insert: O(1)*
  - *find: O(1)*
  - *erase: o(1)*

*A set is a container that stores **unique** elements in **sorted** order.*
It is implemented using a self-balancing binary search tree.
*The set's elements can be **added** or **deleted**, but once they are added, they **cannot be changed**.*

```cpp
// Here is an example of how to use a set in C++:

#include<iostream>
#include<set>
using namespace std;

int main() {
  // Create a set of integers.
  set<int> mySet;

  // Add some elements to the set.
  mySet.insert(1);
  mySet.insert(2);
  mySet.insert(3);

  // Check if an element is in the set.
  if (mySet.find(2) != mySet.end()) {
    cout << "The element 2 is in the set." << std::endl;
  } else {
    cout << "The element 2 is not in the set." << std::endl;
  }

  // Delete an element from the set.
  mySet.erase(2);

  // Iterate over the elements of the set.
  for (set<int>::iterator it = mySet.begin(); it != mySet.end(); ++it) {
    cout << *it << endl;
  }

  return 0;
}

/*
Expected output:
The element 2 is in the set.
1
3
*/
```

INSERTION

1. *To add an element to a set, you use the **insert() function**.*
   The insert() function takes the element as a parameter and returns an iterator to the element.
   If the element is already in the set, the insert() function does nothing and returns an iterator to the existing element.

DELETION

2. *To delete an element from a set, you use the **erase() function**.*
   The erase() function takes the element as a parameter and returns an iterator to the next element in the set.
   If the element is not in the set, the erase() function does nothing and returns an iterator to the end of the set.

SEARCHING

3. *To check if an element is in a set, you use the **find() function**.*
   The find() function takes the element as a parameter and returns an iterator to the element if it is in the set.
   If the element is not in the set, the find() function returns an iterator to the end of the set.

ITERATION

4. *To iterate over the elements of a set, you use a for loop.*
   *The for loop **starts** at the beginning of the set and ends at the end of the set.*
   *The for loop **iterates** over the elements of the set in sorted order.*