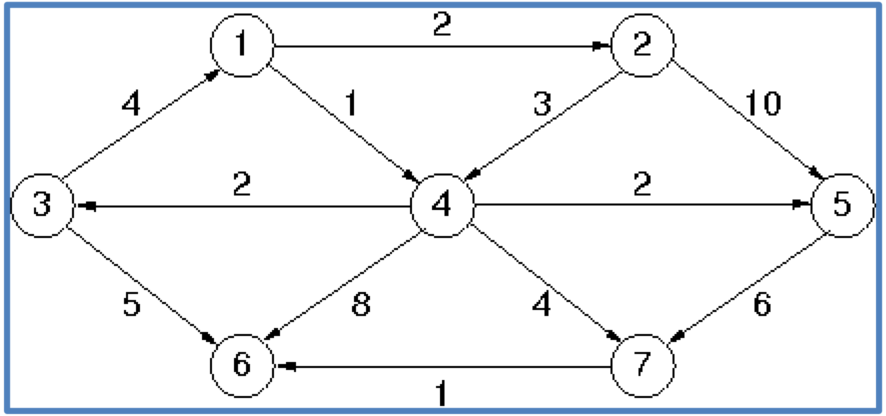
SHORTEST PATH IN A GRAPH



Partha P Chakrabarti
Indian Institute of Technology Kharagpur

Shortest Path in a Graph

Problem: Given an directed Graph G = (V, E), and two nodes, s and g in V, find a shortest (cost) path from s to g in V.

In unweighted graphs edge cost is 1. Thus shortest path is the path length.

In Fig 1, if s =A, g = G, Shortest Path = {A, B, D,G} and Cost = Length is 3.

The cost of a path is measured in terms of the sum of the edge costs of the path from s to g.

In Fig 2, if s=1, g=6, Shortest Cost Path = {1,4,7,6} where Length is 3 and cost is 6. There is a shorter length path {1,4,6} but of higher cost (9).

For undirected graphs, we replace an undirected edge e = (m,n) by two directed edges e1 = (m,n) and e2 = (n,m) of the same weight as e to get a directed graph.

The graph may have cycles or may be a Directed Acyclic Graph (DAG)

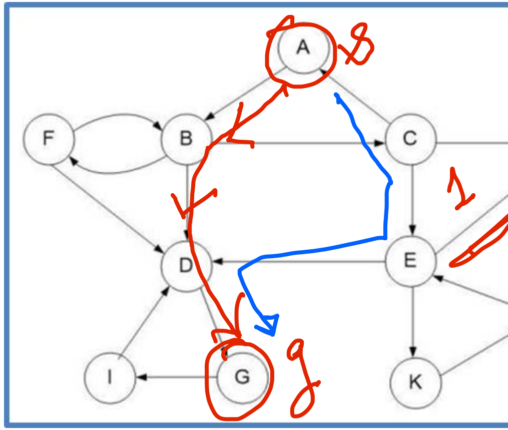


Fig 1

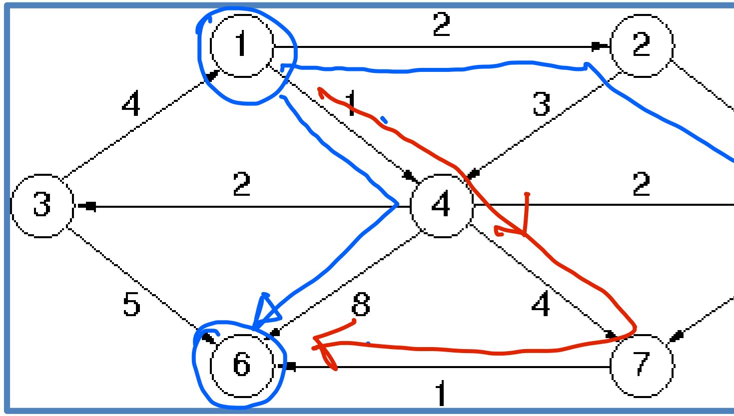


Fig 2

Depth-First Search

Global Data: G = (V,E)

visited [i] indicates if node i is visited. / initially 0 /

Parent[i] = parent of a node in the Search / initially NULL /

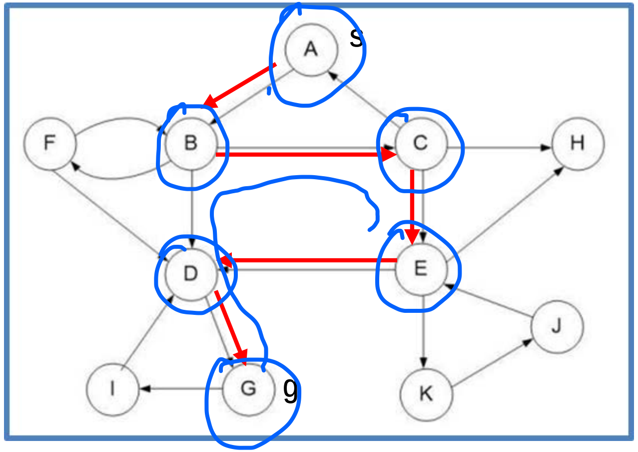
succ(i) = {set of nodes to which node i is connected}

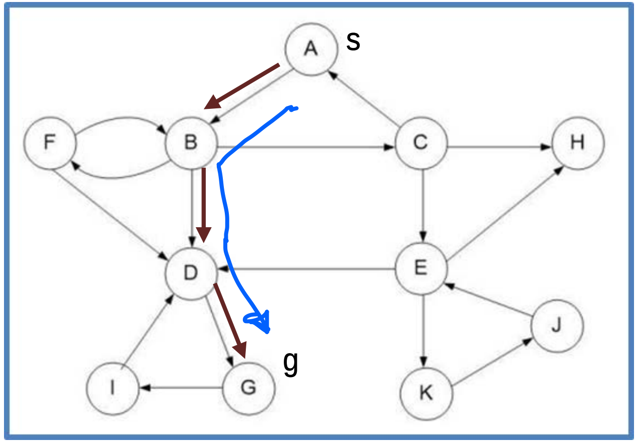
```
PathDfs(s,g) {
   visited[s] = 1;
   if (s == g) return with path through parent links;
   for each n in succ(s) do
      if (visited [n] == 0)
                    { Parent[n] = s;
                      PathDfs(n,g) }

}
```

Time Complexity O(|V| + |E|)

// The first solution in DFS may not be the shortest path.
If you wish to find the shortest path using DFS then you may need to backtrack and handle loops in the graph (Home Exercise) //





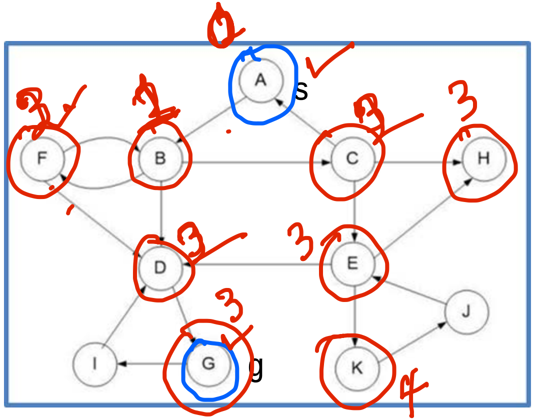
Breadth-First Search

```
visited[i] all initialized to 0;
Length[i] length from s to i, all initialized to 0;
Parent[i] = parent of i  / initially Null/
Queue Q initially {}
BFS(s, g) {
 visited [s] = 0;  Q = {s};
 While Q != {} {
   If Q is empty then return with failure ("No Path");
   n = DeQueue (Q);
   if (n== g) return with path through parent links;
   visited [n] = 1;
   For each k in succ (n)
         if (visited[k]==0) && (k is not already in Q) {
               parent[k] = n;
               Length[k] = Length[n]+1;
               EnQueue(Q,k); }
 }
Time Complexity O(|V| + |E|)
```

new node

If Q = {} failing

Step	Queue Q	Node DeQueued [Length]
1	{A}	A [0]
2	{B}	B [1]
3	{C,D,F}	C [2]
4	{D,F,E,H}	D [2]
5	{F,E,H,G}	F [2]
6	{E,H,G}	E [3]
7	{H,G,K}	H [3]
8	{G,K}	G [3]



Length[k] gives the length of the shortest path from s to k.

When a goal node is removed from the Queue Q, the shortest length path to it is found.

The Algorithm, will work in case there are multiple nodes which satisfy the goal condition and we are to find a path to any one of them

Shortest Paths Weighted Directed Graphs

+ve cost G edges

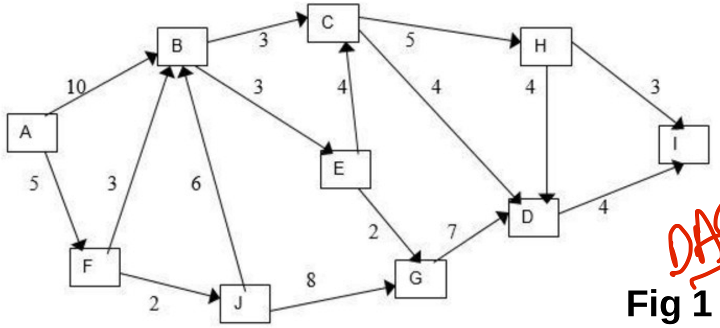


Fig 1

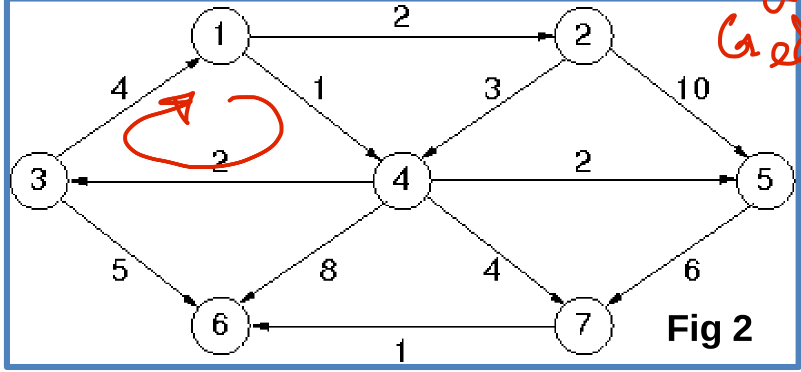


Fig 2

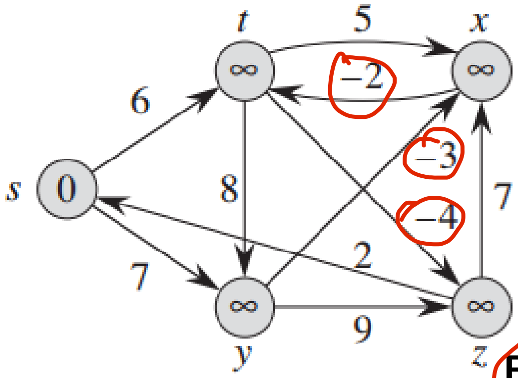


Fig 3

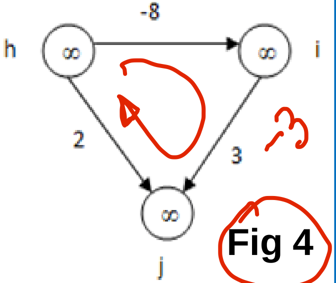
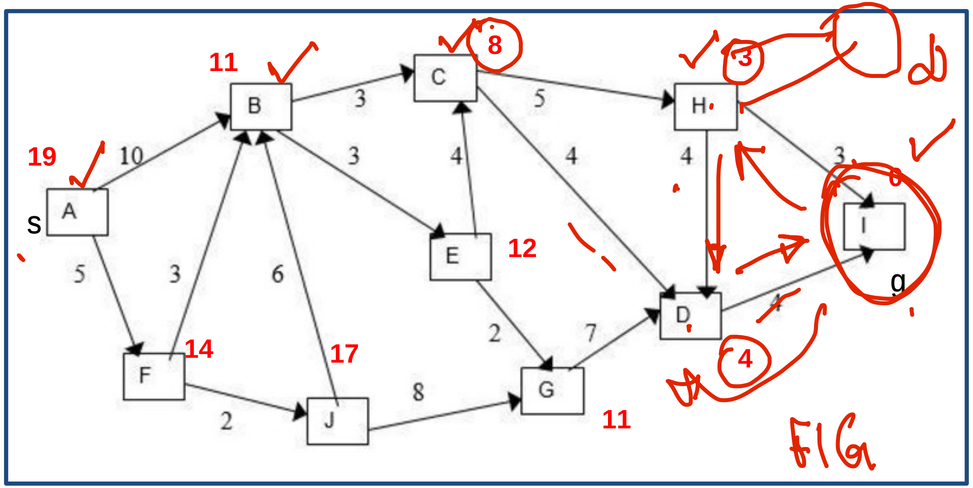


Fig 4

<u>Varieties of Shortest Path Problems</u>:
Source-Goal Problem ✓ S,g
Single Source – All Nodes Problem S
All Pairs Shortest Paths ✓ (i,j)

<u>Types of Graphs</u>:
Directed Acyclic Graphs (DAGs) [Fig 1] ✓ 1
General Graphs with positive edge costs [Fig 2] ✓ 2
General Graphs with no negative cost cycles [Fig 3] ✓
Negative cost cycle [Fig 4] ✓ 3 4

Shortest Cost Path in DAGs



visited [i] indicates if node i is visited. / initially 0 /
cost[i] = cost of path from i to g, initially infinity
succ(i) = {set of nodes to which node i is connected}

```
DFSP(node,g) {
    local variable value = ∞;
    visited[node] = 1;
    if (node == g) { cost[node] = 0; return 0};
    for each n in succ(node) do {
        if (visited [n] == 0) DFSP(n);
        value = min (value, (cost[n] + C[node,n]))
    }
    cost[node] = value;
    return cost[node];
}
```

Time Complexity O(|V| + |E|)
Will not work for Graphs which have cycles.
Works for negative edge cost DAGs.
Can be adapted to all pairs shortest paths for DAGs (Exercise).

RECURSIVE DEFINITION:

SP(n,g) = 0 if n == g;
= ∞ (Infinity), if (n !=g) and succ(n) is NULL;
= min { SP(m,g) + C[n,m] },
for all m in succ(n),
if (n != g) and succ (n) is non-empty

Best First Search in Weighted Directed Graphs

G = (V,E) / Assume positive edge costs/
visited[i] all initialized to 0
cost[j] cost from s to j, all initialized to ∞
Ordered Queue OrQ initially {}

ordered by cost[i]

```
BFSW(s,g) {
 cost [s] = 0; OrQ = {s};
 While OrQ != {} {
  If OrQ is empty terminate ("No Solution");
  j = Remove_Min (OrQ); visited[j] = 1;
  if (j == g) terminate with solution cost[j];
  For each k in succ (j) {
  If (visited[k] == 0) {
       if (cost[k] > (cost[j] + C[j,k])) {
          cost[k] = cost[j] + C[j,k];
          Insert_Reorder(OrQ,k);}
     }
 }
} / This method is called Dijkstra's Algorithm /
```

cost update

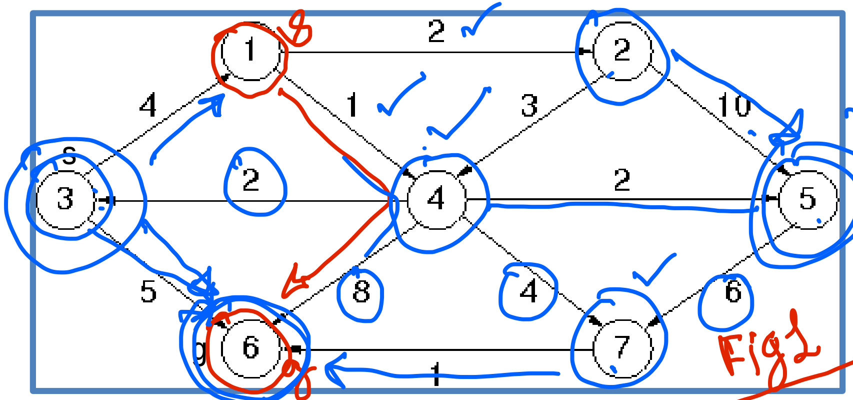


Fig1

	Queue OrQ with node costs	Node Removed
1	{1[0]}	1 [0]
2	{4[1], 2[2]}	4 [1]
3	{2[2], 3[3], 5[3], 7[5], 6[9]}	2 [2]
4	{3[3], 5[3], 7[5], 6[9]}	3 [3]
5	{5[3], 7[5], 6[8]}	5 [3]
6	{7[5],6[8])	7[5]
7	{6[6]}	6 [6]

goal

Whenever a node is removed from OrQ, the best cost path to that node has been obtained. (Detailed proof is left as exercise)

new

Complexity is O(|E| log |E|), that is, O(|E| log |V|) using MinHeap or Balanced Tree. May also be implemented by an array in O(|V|*|V| + |E|)

Fibonacci

Thank you

Already visited node
may need to
be revisited

<u>Homework Exercise</u>:

Algorithm for Shortest Path in a weighted Graph with negative edge costs but no negative cycles