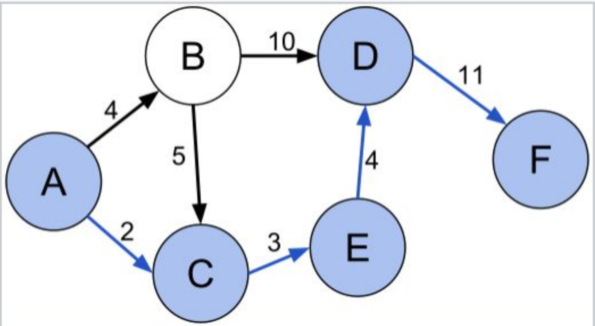
CIS 263 Introduction to Data Structures and Algorithms

Shortest Path (Graph Algorithms)

Shortest Path (Graph Algorithms)

- Given a start and a destination node, we have to find the shortest path from start to the destination node.



Shortest path (A, C, E, D, F) between vertices A and F in the weighted directed graph Src: Wikipedia

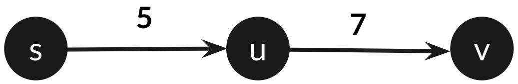
Shortest Path Algorithms

- Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge weight.
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
- A* search algorithm solves for single-pair shortest path using heuristics to try to speed up the search.
- Floyd–Warshall algorithm solves all pairs shortest paths.
- Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs.
- Viterbi algorithm solves the shortest stochastic path problem with an additional probabilistic weight on each node.

src: Wikipedia

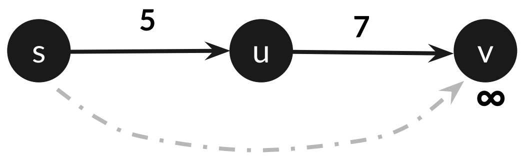
Dijkstra's algorithm

- We are given a graph as in the right
- We want to traverse from "s" to node "v"
- Cost to node "u" is 5; however
- There is no direct path from "s" to "v"



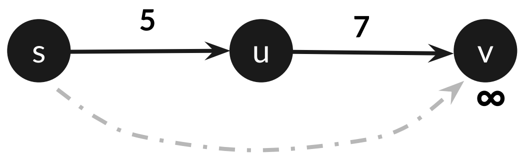
Dijkstra's algorithm

- We are given a graph as in the right
- We want to traverse from "s" to node "v"
- Cost to node "u" is 5; however
- There is no direct path from "s" to "v"
- We can assign an "∞" to node v (from u as there is no direct path.



Dijkstra's algorithm

- We are given a graph as in the right
- We want to traverse from "s" to node "v"
- Cost to node "u" is 5; however
- There is no direct path from "s" to "v"
- We can assign an "∞" to node v (from u as there is no direct path.
- **Dijkstra's algorithm proposes a relaxation rule:** *we can take an intermediate node if the summed cost is less than the indirect cost*

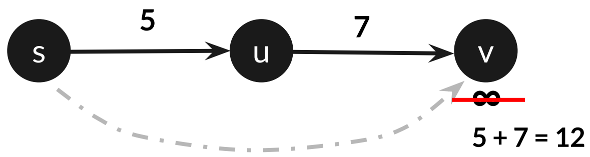


Relaxation rule:

if d[u] + c(u, v) < d[v]:
    d[v] = d[u] + c(u,v)
where,
  d[u] : cost to node u; d[v] : cost to node; and v, c(u, v): cost at edge (u, v)

Dijkstra's algorithm

- We are given a graph as in the right
- We want to traverse from "s" to node "v"
- Cost to node "u" is 5; however
- There is no direct path from "s" to "v"
- We can assign an "**∞**" to node v (from u as there is no direct path.
- **Dijkstra's algorithm proposes a relaxation rule:** *we can take an intermediate node if the summed cost is less than the indirect cost*
- As per the relaxation rule cost to v (via u) becomes (5+7) = 12

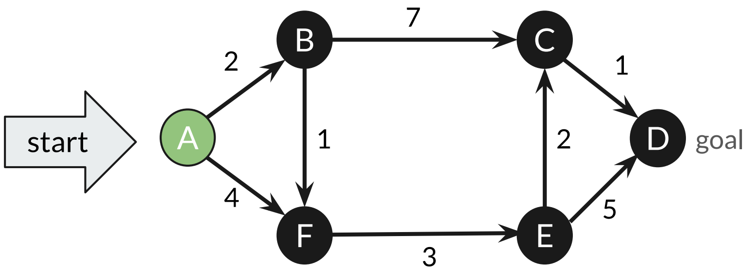


Relaxation rule:

if d[u] + c(u, v) < d[v]:
    d[v] = d[u] + c(u,v)
where,
  d[u] : cost to node u; d[v] : cost to node; and v, c(u, v): cost at edge (u, v)

Dijkstra’s algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞

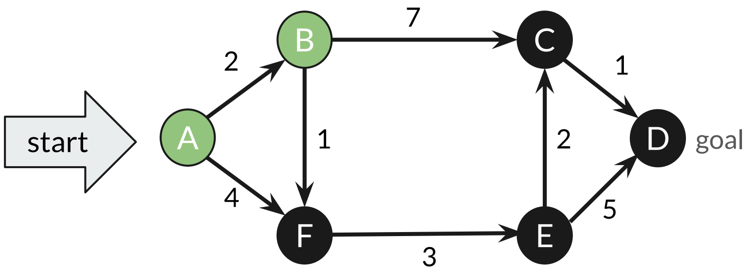


v	d[v]
B	2
C	∞
D	∞
E	∞
F	4

Shortest path: A

Dijkstra’s algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let’s **relax** those two nodes

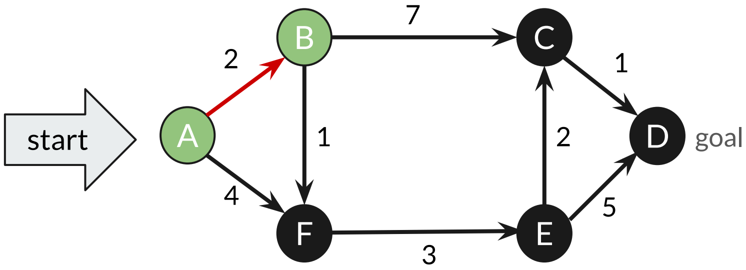


v	d[v]
B	2
C	∞
D	∞
E	∞
F	4

Shortest path: | A | B |

Dijkstra's algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let's **relax** those two nodes

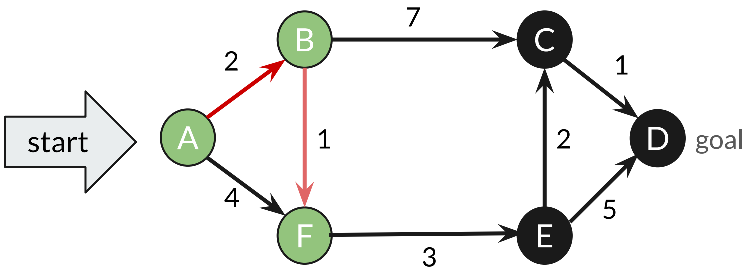


v	d[v]	
B	2	
C	9	~~∞~~
D	∞	
E	∞	
F	3	~~4~~

Shortest path: A | B

Dijkstra's algorithm

- Lets try another graph
- We want to traverse from "A" to node "D"
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let's **relax** those two nodes
- Move to the closest node is F; F has direct connection to E; lets relax E



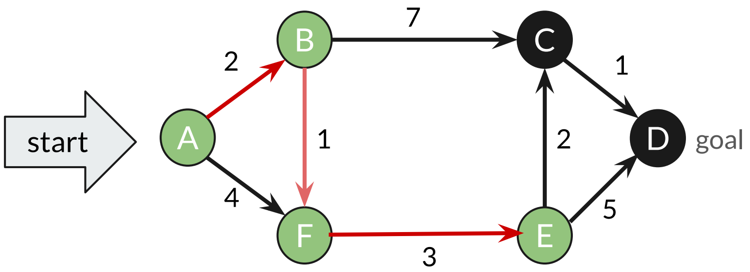
v	d[v]	
B	2	
C	9	
D	∞	
E	6	~~∞~~
F	3	

Shortest path: | A | B | F |

Dijkstra’s algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let’s **relax** those two nodes
- Move to the closest node is F; F has direct connection to E; lets relax E
- Move to E

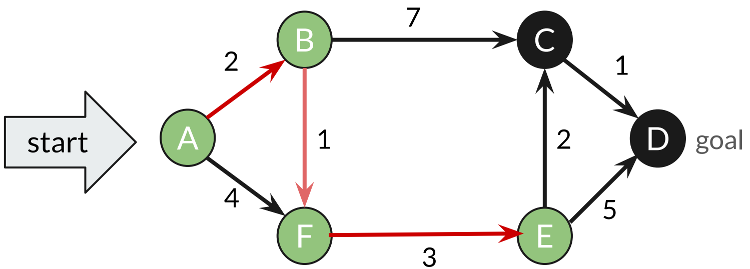
Shortest path: | A | B | F | E |



v	d[v]
B	2
C	9
D	∞
E	6
F	3

Dijkstra's algorithm

- Lets try another graph
- We want to traverse from "A" to node "D"
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let's **relax** those two nodes
- Move to the closest node is F; F has direct connection to E; lets relax E
- Move to E
- E will relax C and D



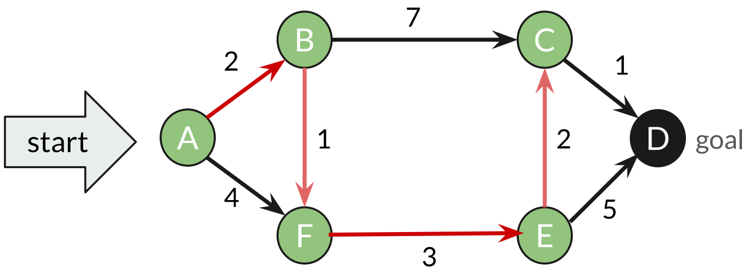
v	d[v]	
B	2	
C	8	~~9~~
D	11	~~∞~~
E	6	
F	3	

Shortest path: | A | B | F | E |

Dijkstra's algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let's **relax** those two nodes
- Move to the closest node is F; F has direct connection to E; lets relax E
- Move to E
- E will relax C and D
- Move to the next closest node C

Shortest path: | A | B | F | E | C |

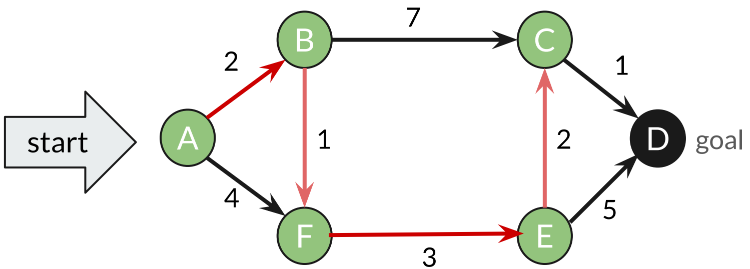


v	d[v]
B	2
C	8
D	11
E	6
F	3

Dijkstra’s algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let’s **relax** those two nodes
- Move to the closest node is F; F has direct connection to E; lets relax E
- Move to E
- E will relax C and D
- Move to the next closest node C
- C will relax D

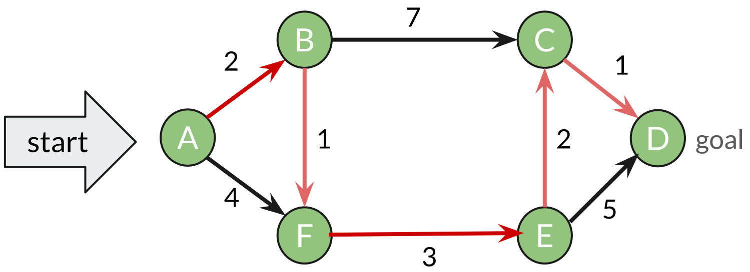
Shortest path: A | B | F | E | C



v	d[v]
B	2
C	8
D	9	~~11~~
E	6
F	3

Dijkstra’s algorithm

- Lets try another graph
- We want to traverse from “A” to node “D”
- Initialize a tracking table
- We have direct connection to B and F; for others we initialize cost to ∞
- Move to the closest node is B; B has direct connection to C, and F; let’s **relax** those two nodes
- Move to the closest node is F; F has direct connection to E; lets relax E
- Move to E
- E will relax C and D
- Move to the next closest node C
- C will relax D
- Move to the next closest node D (**destination/goal hit**)



v	d[v]
B	2
C	8
D	9
E	6
F	3

Total cost

Shortest path:

A	B	F	E	C	D

Dijkstra’s algorithm – limitations

- It’s a greedy algorithm; so optimal solution cannot be guaranteed.
- Doesn’t work for negative weights