DS:
Shortest Path

Liwei

What is Single Source Shortest Path?

- Single source shortest path is about finding a path between a given vertex (called Source) to all other vertices in a graph such that, the total distance between them (source & destination) is minimum.
- Examples
  - Let's say we have office in 5 different citites and we need to travel from Head office to all other offices.
  - Flight charges between cities are known, what is the cheepest way to reach each office from HQ?

Algorithms: single source shortest path

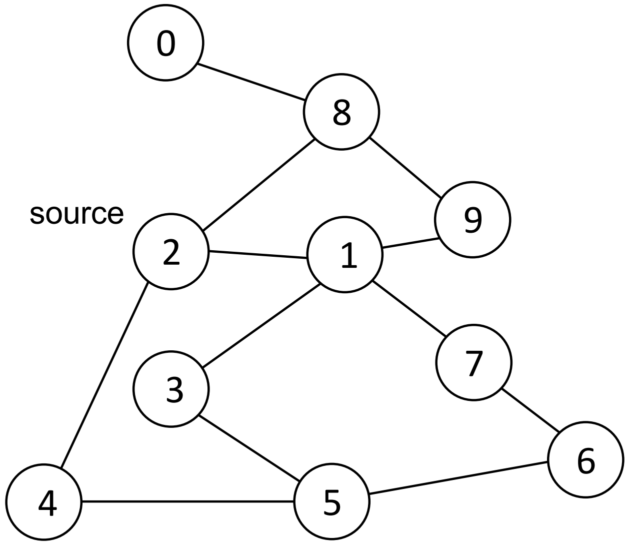
- Breadth first search
- Depth first search
- Dijkstra
- Bellman Ford

What is BFS

- BFS is an algorithm for traversing Graph data structures. It starts at some arbitrary node of a graph and explores the neightboring nodes (which are at current level) first, before moving to the next level neighbors.
- BFS is can be twisted to find Single Source Shortest Path. We need a extra variable called Parent to keep track of path.

Algorithm – Single Source Shortest Path using BFS

```
BFS-for-SSSP (any starting vertex)
        intialize a Queue
        Create a Parent reference in each node
        Enqueue (Source Vertex)
        do until queue is not empty
                currentVertex = Dequeue(Vertex)
                for each adjacent vertices
                        if adjacent vertex is visited
                                don’t do anything
                        else
                                enqueue adjacent vertex
                                update their parent as currentVertex
                                mark currentVertex as visited
```



QUEUE

Algorithm – Single Source Shortest Path using BFS

```
BFS-for-SSSP (any starting vertex)
        intialize a Queue                                   O(1)
        Create a Parent reference in each node              O(1)
        Enqueue (Source Vertex)                             O(1)
        do until queue is not empty  ---------------------------------- O(V)
                currentVertex = Dequeue(Vertex)             O(1)
                for each adjacent vertices  ----------------O(Adj vertex)   O(Adj vertex)   O(E)
                        if adjacent vertex is visited       O(1)
                                don't do anything           O(1)
                        else                                O(1)
                                enqueue adjacent vertex     O(1)
                                update their parent as currentVertex
                                mark currentVertex as visited
```

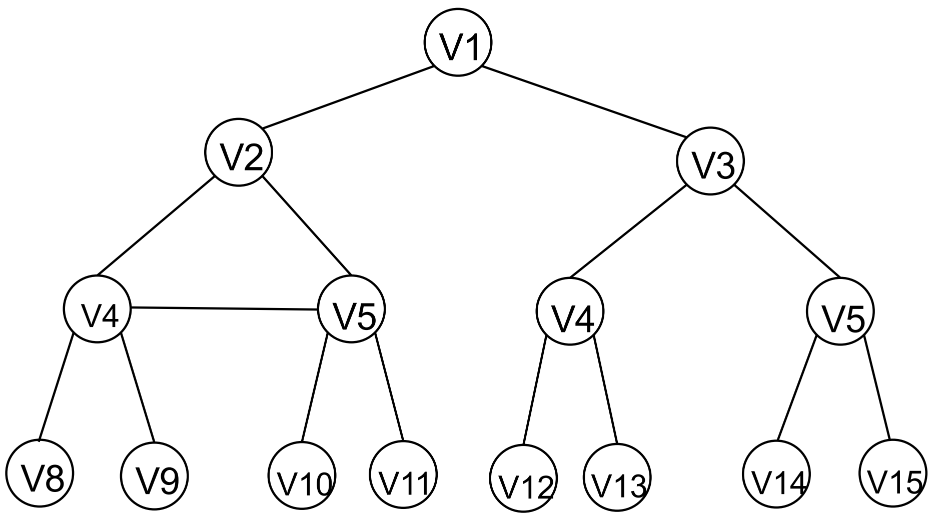
Time complexity: O(E)

Single Source Shortest Path Algorithms

Type of Graphs	BFS
unWeighted-unDirected	V
unWeighted-Directed	V
Positive-Weighted-unDirected	X
Positive-Weighted-Directed	X
Negative-Weighted-unDirected	X
Negative-Weighted-Directed	X

Why Weighted Graph does not work with BFS

- BFS explores a given graph only in breath-way. But there can always be a better route, which is not breadth-way.

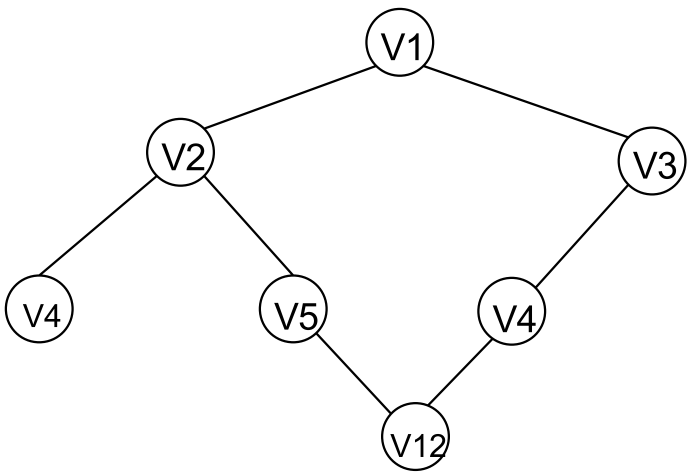


Algorithms: single source shortest path

- Breadth first search
- Depth first search
- Dijkstra
- Bellman Ford

Why DFS does not work for SSSP

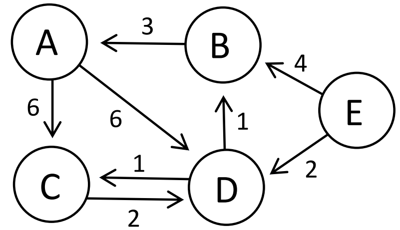
- DFS has the tendency to go as far as possible from source. Hence it can never find Shortest Path



Algorithms: single source shortest path

- Breadth first search
- Depth first search
- Dijkstra
- Bellman Ford

Algorhms - Dijkstra



Dijkstra()

set the distance of all the vertices as infinite and source vertex as 0

Save all the vertices in min-Heap

do until min-Heap is not empty

currentVertex = extract from min-Heap

for each neighbor of currentVertex

if currentVertex’s distance + currentEdge < neighbor’s distance

update neighbor’s distance and parent

Algorhms - Dijkstra

Dijkstra()

set the distance of all the vertices as infinite and source vertex as 0 — $O(V)$

save all the vertices in min-Heap — $O(V)$

do until min-Heap is not empty $O(V)$

    currentVertex = extract from min-Heap $O(\log V)$

    for each neighbor of currentVertex $O(V)$ — $O(V^2)$

        if currentVertex's distance + currentEdge < neighbor's distance $O(1)$

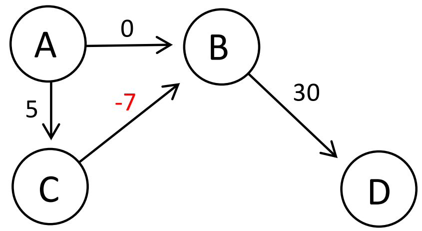
            update neighbor's distance and parent $O(1)$

Time complexity: $O(V^2)$

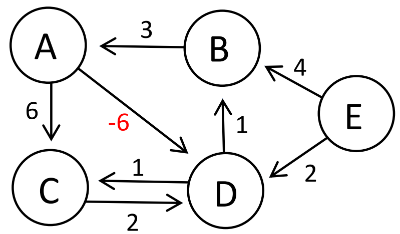
Space complexity: $O(V)$

Single Source Shortest Path Algorithms

Type of Graphs	BFS	Dijkstra
unWeighted-unDirected	V	V
unWeighted-Directed	V	V
Positive-Weighted-unDirected	X	V
Positive-Weighted-Directed	X	V
Negative-Weighted-unDirected	X	X
Negative-Weighted-Directed	X	X
Negative Cycles	X	X



Negative-Weighted



Negative Cycles

Algorithms: single source shortest path

- Breadth first search
- Depth first search
- Dijkstra
- Bellman Ford

Bellman Ford's Algorithm

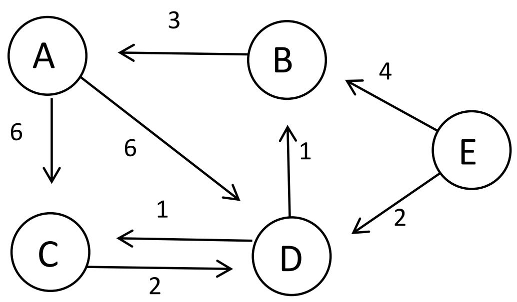
- The Bellman-Ford algorithm is used to find SSSP.
- If a graph contains a "negative cycle" that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle.
- In such a case, the Bellman-Ford algorithm can detect "negative cycles" and report their existence.

Given

Edge	Weight
A>C	6
A>D	6
B>A	3
C>D	2
D>C	1
D>B	1
E>B	4
E>D	2

if (distance of source vertex + current weight between source and destination vertex) < (current distance of destination vertex)
update distance of destination vertex with (the distance of source vertex + current weight between source and destination vertex)

Distance Matrix	
Vertex	Give Distance
A	∞
B	∞
C	∞
D	∞
E	0



Final answer

vertex	Distance from E	Path from E
A	6	B>D>E
B	3	D>E
C	3	D>E
D	2	E
E	0	0

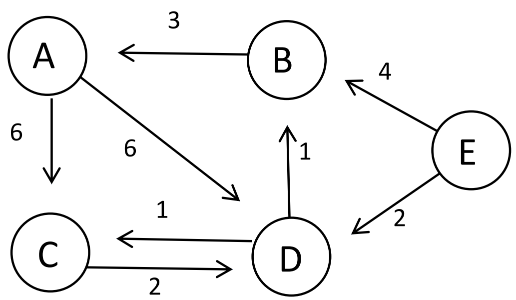
Given

Edge	Weight
A>C	6
A>D	6
B>A	3
C>D	2
D>C	1
D>B	1
E>B	4
E>D	2

if (distance of source vertex + current weight between source and destination vertex) < (current distance of destination vertex)
update distance of destination vertex with the distance of source vertex + current weight between source and destination vertex

Distance Matrix

Vertex	Give Distance	Iteration #1	
		Distanc	Parent
A	∞		
B	∞		
C	∞		
D	∞		
E	0		



Final answer

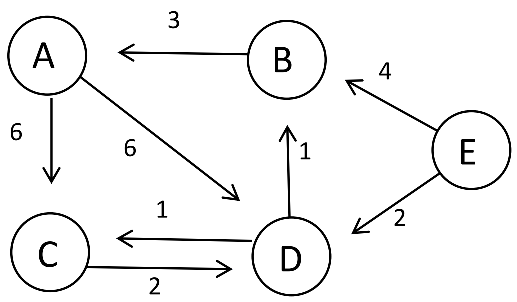
vertex	Distance from E	Path from E
A	6	B>D>E
B	3	D>E
C	3	D>E
D	2	E
E	0	0

Given

Edge	Weight
A>C	6
A>D	6
B>A	3
C>D	2
D>C	1
D>B	1
E>B	4
E>D	2

if (distance of source vertex + current weight between source and destination vertex) < (current distance of destination vertex)
update distance of destination vertex with the distance of source vertex + current weight between source and destination vertex

Distance Matrix					
Vertex	Give Distance	Iteration #1		Iteration #2	
		Distance	Parent	Distance	Parent
A	∞	∞	-		
B	∞	4	E		
C	∞	∞	-		
D	∞	2	E		
E	0	0	-		



Final answer

vertex	Distance from E	Path from E
A	6	B>D>E
B	3	D>E
C	3	D>E
D	2	E
E	0	0

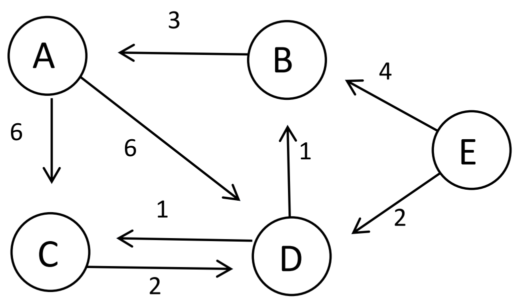
Given

Edge	Weight
A>C	6
A>D	6
B>A	3
C>D	2
D>C	1
D>B	1
E>B	4
E>D	2

if (distance of source vertex + current weight between source and destination vertex) < (current distance of destination vertex)
  update distance of destination vertex with the distance of source vertex + current weight between source and destination vertex

Distance Matrix

Vertex	Give Distance	Iteration #1		Iteration #2		Iteration #3	
		Distance	Parent	Distance	Parent	Distance	Parent
A	∞	∞	-	4+3=7	B		
B	∞	4	E	2+1=3	D		
C	∞	∞	-	2+1=3	D		
D	∞	2	E	2	E		
E	0	0	-	0	-		



Final answer

vertex	Distance from E	Path from E
A	6	B>D>E
B	3	D>E
C	3	D>E
D	2	E
E	0	0

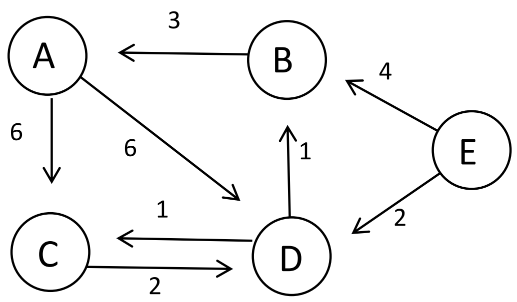
Given

Edge	Weight
A>C	6
A>D	6
B>A	3
C>D	2
D>C	1
D>B	1
E>B	4
E>D	2

if (distance of source vertex + current weight between source and destination vertex) < (current distance of destination vertex)
update distance of destination vertex with the distance of source vertex + current weight between source and destination vertex

Distance Matrix										
Vertex	Give Distance	Iteration #1		Iteration #2		Iteration #3		Iteration #4	
		Distance	Parent	Distance	Parent	Distance	Parent	Distance	Parent
A	∞	∞	-	4+3=7	B	3+3=6	B	6	B
B	∞	4	E	2+1=3	D	3	D	3	D
C	∞	∞	-	2+1=3	D	3	D	3	D
D	∞	2	E	2	E	2	E	2	E
E	0	0	-	0	-	0	-	0	-

go (v-1) runs



Final answer

vertex	Distance from E	Path from E
A	6	B>D>E
B	3	D>E
C	3	D>E
D	2	E
E	0	0

Bellman Ford Algorithm

```
BellmanFord(G)
    set all the vertex distance to infinite and source as 0
    for 1 to V-1
        for each edge(u,v)
            if d(v) > d(u) + w(u,v)        // if current weight of v is greater
                                           // than current weight of U + current edge
                d(v) = d(u) + w(u,v)
                update parent for ‘v’

    for each edge (u,v)
        if d(v) > d(u) + w(u,v)
            the report existence of negative-weight cycle

    print all vertex with distance and parent
```

偵測負環的邏輯是，步數走 V - 1 步以後如果還可以變更短就是有負環：
因此在V-1步後，若仍存在某個edge之weight 能降低與該edge連結之vertex的distance，則可判斷Graph中存在negative cycle

Bellman Ford Algorithm

```
BellmanFord(G)
    set all the vertex distance to infinite and source as 0      O(V)
    for 1 to V-1                                                 O(E)
        for each edge(u,v)                                       O(1)     O(EV)
                if d(v) > d(u) + w(u,v)

                d(v) = d(u) + w(u,v)                             O(1)
                update parent for 'v'                            O(1)              O(EV)

    for each edge (u,v)                                          O(E)
        if d(v) != d(u) + w(u,v)                                 O(1)     O(E)
                the report existence of negative-weight cycle    O(1)

    print all vertex with distance and parent                    O(V)
```

Time complexity: O(VE)

Single Source Shortest Path Algorithms

Type of Graphs	BFS	Dijkstra	Bellman Ford
unWeighted-unDirected	V	V	V
unWeighted-Directed	V	V	V
Positive-Weighted-unDirected	X	V	V
Positive-Weighted-Directed	X	V	V
Negative-Weighted-unDirected	X	X	V
Negative-Weighted-Directed	X	X	V
Negative Cycles	X	X	V (can detect)

When to use which algorithms for SSSP

	BFS	Dijkstra	Bellman Ford
Time Complexity	O(V^2)	O(V^2)	O(V*E)
Implementation	Easy	Moderate	Moderate
Limitation	Does not work for weighted graphs	Does not work for nagetive-weighted graph / nagetive circles	N/A
unWeighted Graph	V	V	V
	Use this as Time Complexity is good and easy to implement	Don't use as implemenation is not as easy as BFS	Don't use as Time Complexity gets bad
Weighted Graph	X	V	V
	Not supported	Use this as Time Complexity is better than Bellman	Don't use as Time Complexity is not good
Negative Circles	X	X	V
	Not supported	Not supported	Use this as BFS and Dijkstra does not supports negative circles