Dijkstra

- Greedy Algorithm
- Only works for edges with positive weights

General Overview

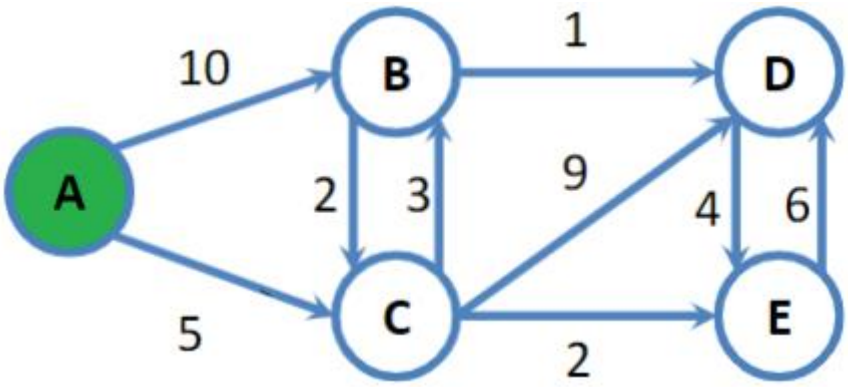
1) Initialize the distance of the source vertex to 0 and the distance of all other vertices to infinity.

2) Create a priority queue(Minimum Heap) to store the discovered vertices. Add the source vertex to the minimum heap created.

3) While the minimum heap is not empty, do the following:

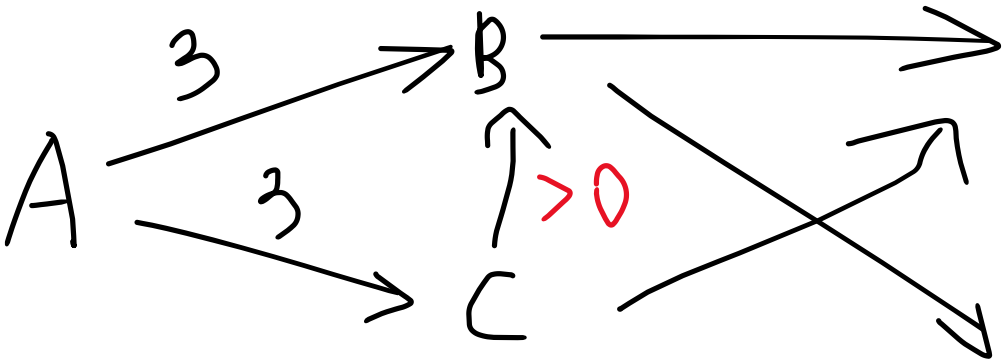
- Remove the vertex with the smallest distance from the source vertex and set it to be visited. We will call this $Vertex_i$
- Go through all the edges in $Vertex_i$. If the corresponding vertex is not discovered, add it to the minimum heap. If it is not visited, if the distance of the corresponding vertex is greater than the distance of $Vertex_i$ + the weight of the edge, the distance of the corresponding vertex is now the distance of $Vertex_i$ + the weight of the edge.

4) Repeat step 3 until the minimum heap is empty.



Vertices	Distance from source
A	0
B	8(A->C->B)
C	5(A->C)
D	9(A->C->B->D)
E	7(A->C->E)

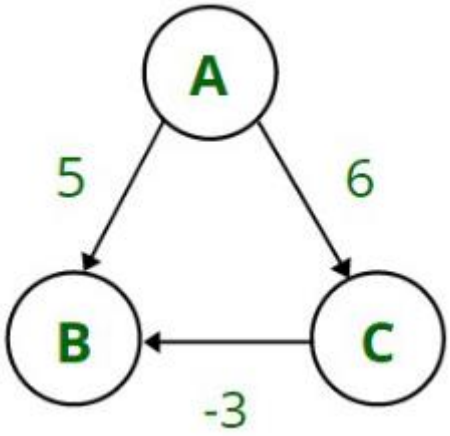
The greediness of Dijkstra and why it works



- A greedy algorithm is an algorithm that only looks at the local optimal, and by doing so will result in the global optimal
- Dijkstra only works with positive edges!
- When we serve, we automatically get the vertex with the shortest distance from the source that has not been visited yet.
- After serving, we finalize the vertex as visited and we know that no matter how many vertices we visit in the future, the distance of the served vertex will always be of this minimal distance

If we serve from the minimum heap and get vertex B, we know that the distance from vertex A to vertex C is >= 3. If someone say that there is a shorter path from A to B by going through vertex C, we can easily disprove that statement. This is because if the distance of A to C is >= 3, and we know that Dijkstra only works with positive edges, the path from C to B will definitely be more than 0. So, the fact that the distance from A to C is >= 3 and the distance of the path from C to B is more than 0, if we add them up it's definitely more than A to B itself. Hence, we can confidently conclude that every time we serve from the minimum heap, the distance of the source vertex to the served vertex will always be of this minimal distance no matter how many vertices and edges we visit in the future.

Why does it not handle negative weights?



- Take A as the source vertex
- Distance of B from A is 5. Distance of C from A is 6.
- Alternatively, by going through A->C->B, distance of B from A is 3.
- However, when we want to serve, we will serve B and finalize it by changing visited to True, so the distance will be set to 5, even though a shorter path exists.
- By having negative weights, the notion of local optimal leading to global optimal will be broken!

Time Complexity of Dijkstra

```
1   discover_queue = MinHeap()
2   discover_queue.append([source,0])
3
4   while discover_queue is not empty:
5       u = discover_queue.serve()
6       u.visited = True
7       for each <u,v,w> in u.edges:
8           if v.visited = True:
9               pass
10          else:
11              if v.discovered = False:
12                  discover_queue.append([v, u.distance+w])
13                  v.discovered = True
14              else:
15                  if v.distance > u.distance+w:
16                      discover_queue.update(v, u.distance+w)
17                      v.discovered = True
```

Line 4: O(V) because each vertex is placed in the minimum heap only once
Line 5: O(log V) because serving from minimum heap
Line 7: O(V) because the maximum number of edges from a vertex is V-1
Line 16: O(log V) because when updating the minimum heap we need to make sure the vertex rise to the correct position
Total time complexity is **O(E log V).** In dense graphs, $E \approx V^2$. Total space complexity is O(V).