CSC 226 SUMMER 2023
ALGORITHMS AND DATA STRUCTURES II
ASSIGNMENT 3
UNIVERSITY OF VICTORIA

1. Suppose all edge weights in a graph are integers in the range from 1 to $|V|$ where V is the set of vertices. How fast can you make Kruskal's algorithm run? Explain.

2. Given an MST for an edge-weighted graph G, suppose that an edge in G that does not disconnect G is deleted. Design an algorithm using which we could find the minimum spanning tree for the modified graph easily without constructing the new tree from scratch again. What is its running time?

3. Show how to solve the single source shortest path problem of the graph in the lecture slides on Dijkstra's algorithm, slide 11 using Dijkstra's algorithm, when the source node is g. Give the initial values of $D(v)$ for each node v. Each time a node is pulled into the cloud, give the D values which have changed as a result.

4. Show how to solve the single source shortest path problem of the graph in the lecture slides on Bellman-Ford's algorithm, slide 8 using the Belllman Ford algorithm, when the source node is b. Give the sequence of D values for all the nodes initially and then just give changes to D values which occurs, by giving the node affected and its new D value, in order of which they occur. In each round, consider the edges in lexicographic order.

5. Design an algorithm for the single source shortest path problem (SSSP) on directed acyclic graphs (DAGs) in $O(m + n)$ time.

1. Suppose all edge weights in a graph are integers in the range from 1 to $|V|$ where V is the set of vertices. How fast can you make Kruskal's algorithm run? Explain.

- Kruskal's algorithm is a graph that sorts the edges in non-decreasing order based on weights. Since the edge weights are integers in the range of 1 to $|V|$, we can use Counting sort which correctly sorts n integers in the range of 0 to k in $O(n+k)$ time.

So the runtime to sort the edges by Counting Sort is $O(|V|+|E|)$ time.
After sorting the edges, Kruskal's algorithm proceeds to process the sorted edges, greedily to construct a MST.

The total time complexity of Kruskal's algorithm would be $O(|V|+|E|+|V|\log|V|)$, where the first term represents the sorting time and the second term represents processing the sorted edges. Since $|E|$ is usually larger than $|V|$, we can rewrite this as $O(|E|+|V|\log|V|)$

2. Given an MST for an edge-weighted graph G, suppose that an edge in G that does not disconnect G is deleted. Design an algorithm using which we could find the minimum spanning tree for the modified graph easily without constructing the new tree from scratch again. What is its running time?

If the deleted edge isn't part of the MST, then there is no need to make any change to the MST.

However, if the deleted edge was originally in the MST, removing it creates 2 separate connected components (let's call this sub1, and sub2) To find the new MST, I need to find the edge with the minimum weight that connects sub1 and sub2.

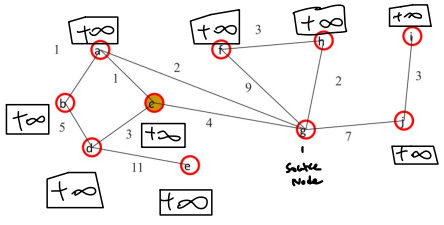
So, I'll use the cut property of MST which guarantees that the edge with the minimum weight linking sub1 and sub2 must be included in the new MST of the modified graph. This is because by removing the original edge, the graph remains connected, so there must exist an edge that connects two components.

To find this edge, I can simply iterate through all the edges in the graph and check if an edge connects sub1 and sub2. The worst-case runtime of this is $O(m)$. m is the number of edges in the graph.
The look up efficiently of whether the source and target vertices of an edge belong to a given set of vertices can be achieved in constant time.

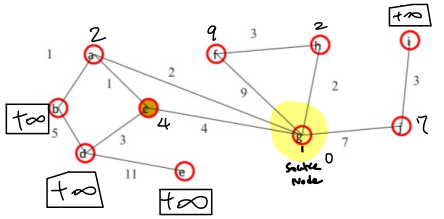
3. Show how to solve the single source shortest path problem of the graph in the lecture slides on Dijkstra's algorithm, slide 11 using Dijkstra's algorithm, when the source node is g. Give the initial values of $D(v)$ for each node v. Each time a node is pulled into the cloud, give the D values which have changed as a result.

1. Initialize the algorithm
   - Set the source node g as the current node.
   - Set the initial distance value $D(v)$ for each node v to infinity, except for the source node g which is 0.
2. Update the distance
   - For each neighbor of the current node g, calculate t tentative distance from the source node g to that neighbor he by adding the distance from g to the current node with weight of the edge between the current node and source node.
   - If the calculated distance is smaller than the current distance, update $D(v)$ with the new smaller distance.
3. Pull into the cloud
   - Find the shortest distance from source node and pull the node with the shortest distance into the cloud.
4. Repeat step 2 and 3.

Dijkstra's algorithm: a greedy algorithm



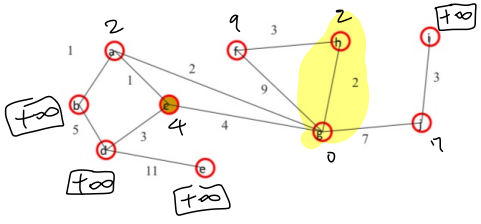
Dijkstra's algorithm: a greedy algorithm



$D(a) = 2$	$D(f) = 9$
$D(b) = +\infty$	$D(g) = 0$
$D(c) = 4$	$D(h) = 2$
$D(d) = +\infty$	$D(i) = +\infty$
$D(e) = +\infty$	$D(j) = 7$

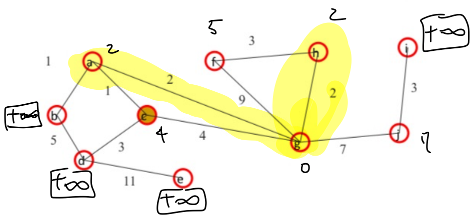
— in the cloud
— add to cloud

Dijkstra's algorithm: a greedy algorithm



$D(a) = 2$	$D(f) = 9$
$D(b) = +\infty$	$D(g) = 0$
$D(c) = 4$	$D(h) = 2$
$D(d) = +\infty$	$D(i) = +\infty$
$D(e) = +\infty$	$D(j) = 7$

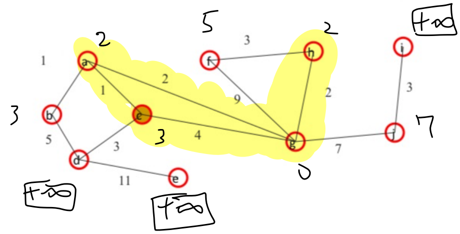
Dijkstra's algorithm: a greedy algorithm



D(a) = 2	D(f) = 5
D(b) = +∞	D(g) = 0
D(c) = 4	D(h) = 2
D(d) = +∞	D(i) = +∞
D(e) = +∞	D(j) = 7

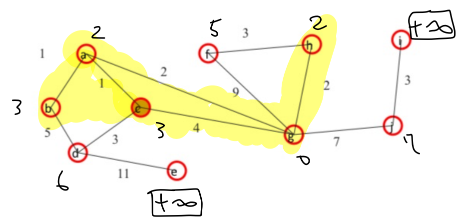
...update number

Dijkstra's algorithm: a greedy algorithm



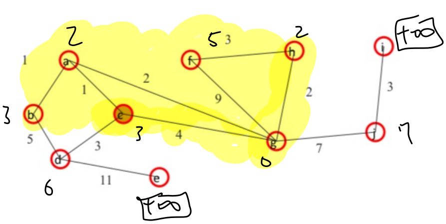
D(a) = 2	D(f) = 5
D(b) = 3	D(g) = 0
D(c) = 3	D(h) = 2
D(d) = +∞	D(i) = +∞
D(e) = +∞	D(j) = 7

Dijkstra's algorithm: a greedy algorithm



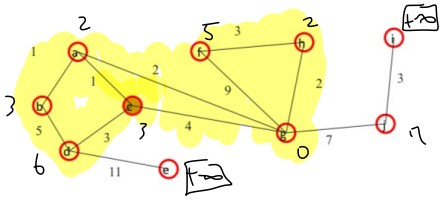
D(a) = 2	D(f) = 5
D(b) = 3	D(g) = 0
D(c) = 3	D(h) = 2
D(d) = +∞	D(i) = +∞
D(e) = +∞	D(j) = 7

Dijkstra's algorithm: a greedy algorithm



D(a) = 2	D(f) = 5
D(b) = 3	D(g) = 0
D(c) = 3	D(h) = 2
D(d) = 6	D(i) = +∞
D(e) = +∞	D(j) = 7

Dijkstra's algorithm: a greedy algorithm



$D(a) = 2$ $D(f) = 5$

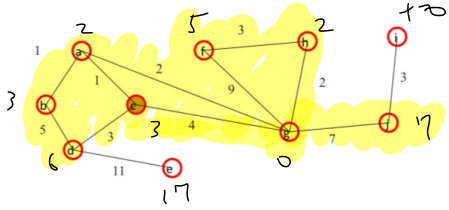
$D(b) = 3$ $D(g) = 0$

$D(c) = 3$ $D(h) = 2$

$D(d) = 6$ $D(i) = +\infty$

$D(e) = +\infty$ $D(j) = 7$

Dijkstra's algorithm: a greedy algorithm



$D(a) = 2$ $D(f) = 5$

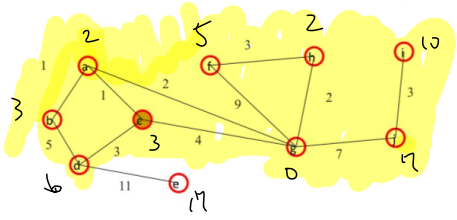
$D(b) = 3$ $D(g) = 0$

$D(c) = 3$ $D(h) = 2$

$D(d) = 6$ $D(i) = +\infty$

$D(e) = 17$ $D(j) = 7$

Dijkstra's algorithm: a greedy algorithm



$D(a) = 2$ $D(f) = 5$

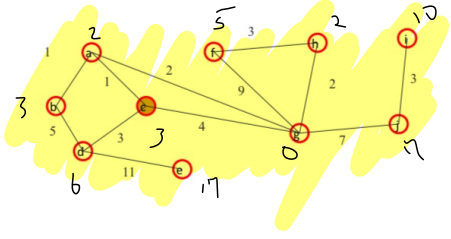
$D(b) = 3$ $D(g) = 0$

$D(c) = 3$ $D(h) = 2$

$D(d) = 6$ $D(i) = 10$

$D(e) = 17$ $D(j) = 7$

Dijkstra's algorithm: a greedy algorithm



$D(a) = 2$ $D(f) = 5$

$D(b) = 3$ $D(g) = 0$

$D(c) = 3$ $D(h) = 2$

$D(d) = 6$ $D(i) = 10$

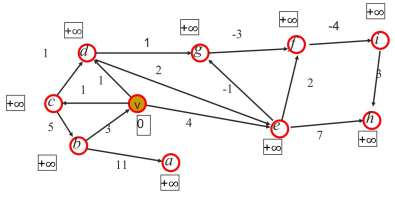
$D(e) = 17$ $D(j) = 7$

4. Show how to solve the single source shortest path problem of the graph in the lecture slides on Bellman-Ford's algorithm, slide 8 using the Belllman Ford algorithm, when the source node is b. Give the sequence of D values for all the nodes initially and then just give changes to D values which occurs, by giving the node affected and its new D value, in order of which they occur. In each round, consider the edges in lexicographic order.

The Bellman-Ford algorithm allows us to find the shortest paths from single source vertex to all other vertices, even in the presence of negative edge weights.

1. Initialization
   - Set the distance of the source vertex b to 0. (dist[b] = 0)
   - Set the distance of all other vertices to ∞ (dist[a]=∞ dist[c]=∞....)
2. Relaxation
   iterate through all edges (u,v) in the graph and update the distances if a shorter path is found.
   If dist[u] + weight(u,v) < dist[v], update dist[v] = dist[u] + weight(u,v).
3. Negative cycle detection
   Run an additional iteration to check for negative cycles,

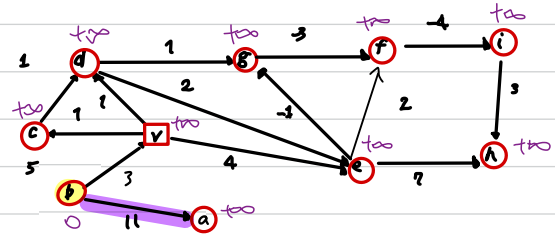
i=1



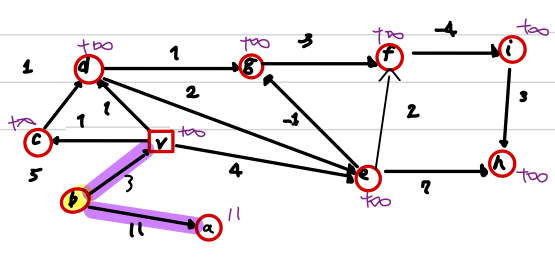
The lexicographic order is,
(a,b),(b,r),(c,d),(d,e),(d,g),(e,f)(e,g),(e,h),(f,i)(g,f),(i,h),(v,c),(r,d),(v,e)

---updated value

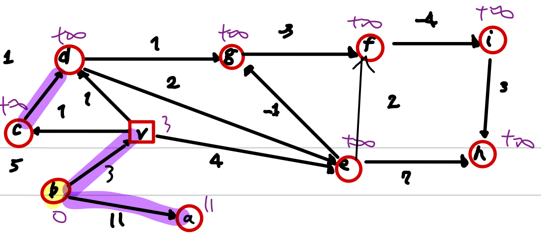
dist[a] = ∞	dist[c]= ∞	dist[y] = ∞
dist[b] = 0	dist[d]= ∞	dist[h] = ∞
	dist[e]= ∞	dist[i] = ∞
	dist[f]= ∞	dist[v]= ∞



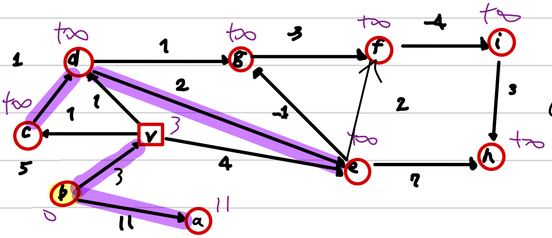
dist[a] = ∞	dist[c]= ∞	dist[y] = ∞
dist[b] = 0	dist[d]= ∞	dist[h] = ∞
	dist[e]= ∞	dist[i] = ∞
	dist[f]= ∞	dist[v]= ∞



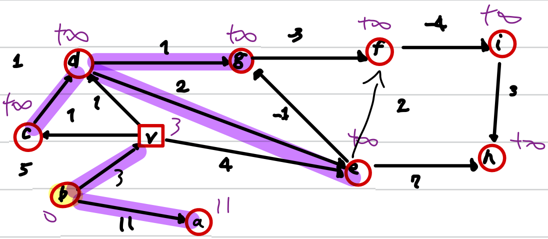
dist[a] = 11	dist[c]= ∞	dist[y] = ∞
dist[b] = 0	dist[d]= ∞	dist[h] = ∞
	dist[e]= ∞	dist[i] = ∞
	dist[f]= ∞	dist[v]= ∞



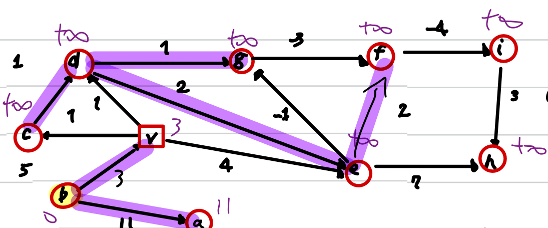
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



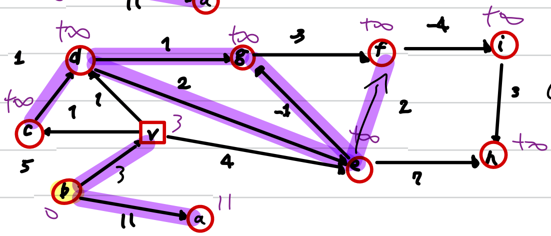
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



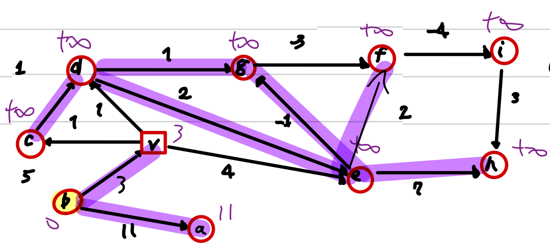
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



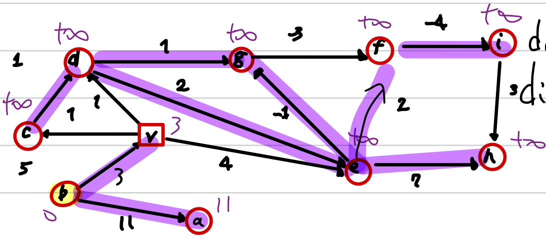
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



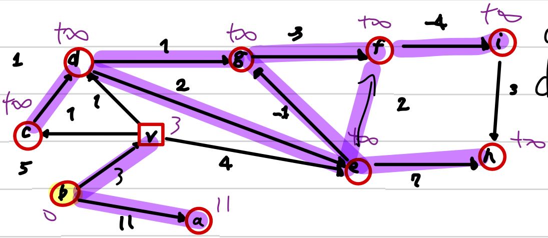
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



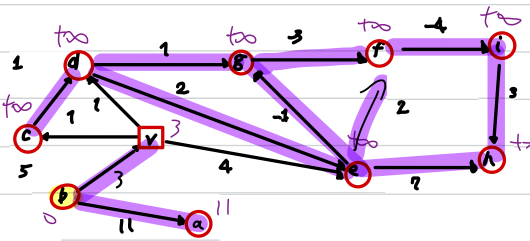
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



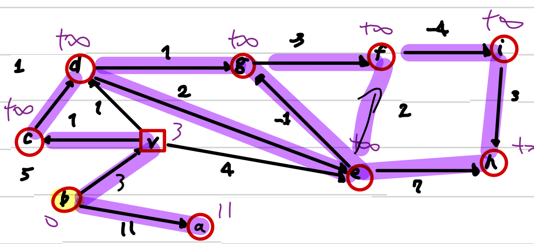
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



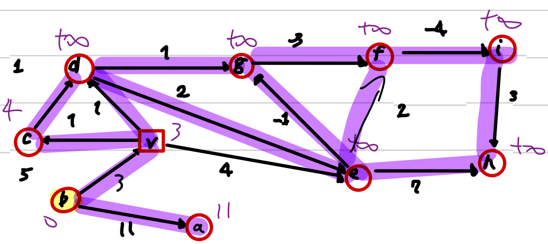
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



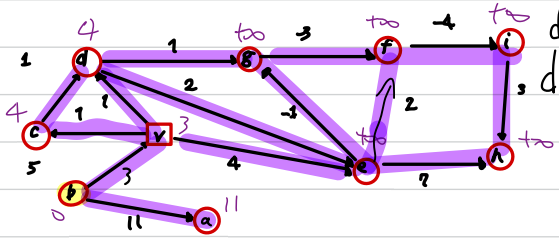
dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



dist[a] = 11
dist[b] = 0
dist[c] = ∞
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



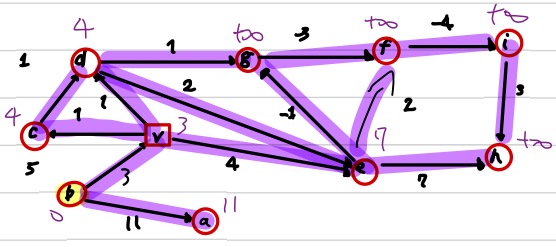
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = ∞
dist[e] = ∞
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



dist[a] = 11
dist[b] = 0

dist[c] = 4
dist[d] = 4
dist[e] = ∞
dist[f] = ∞

dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3

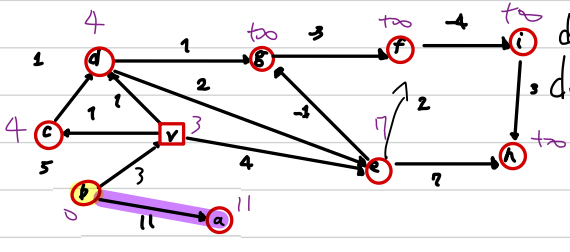


dist[a] = 11
dist[b] = 0

dist[c] = 4
dist[d] = 4
dist[e] = 7
dist[f] = ∞

dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3

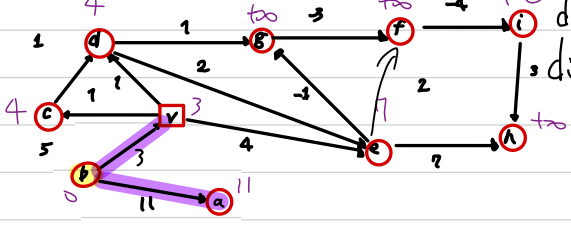
i = 2



dist[a] = 11
dist[b] = 0

dist[c] = 4
dist[d] = 4
dist[e] = 7
dist[f] = ∞

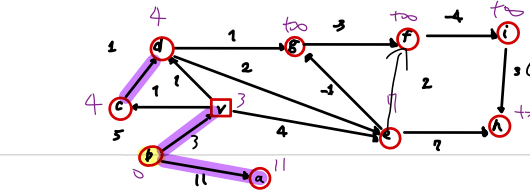
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



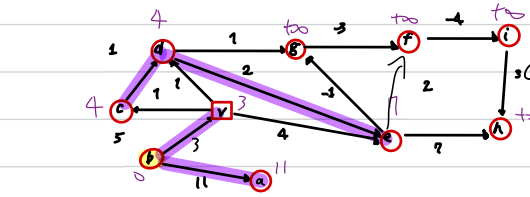
dist[a] = 11
dist[b] = 0

dist[c] = 4
dist[d] = 4
dist[e] = 7
dist[f] = ∞

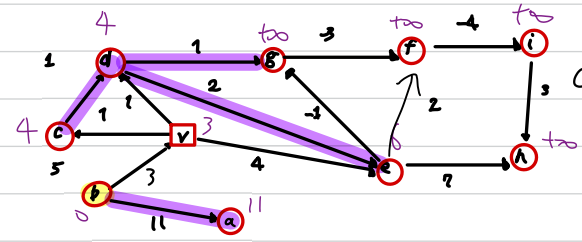
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



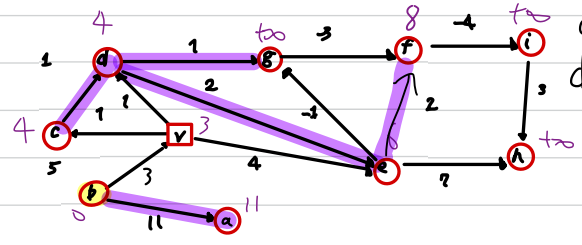
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 7
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



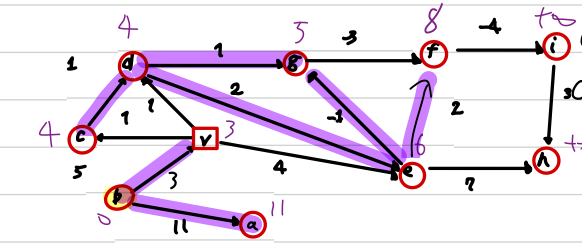
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 7
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



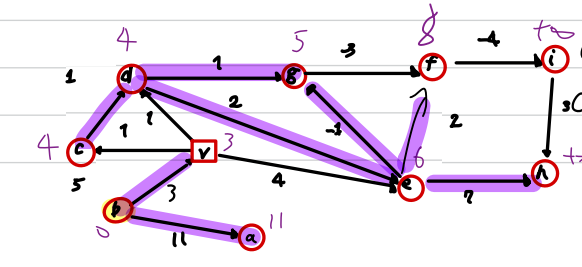
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = ∞
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



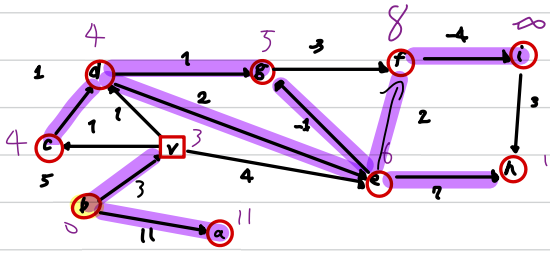
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 8
dist[g] = ∞
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



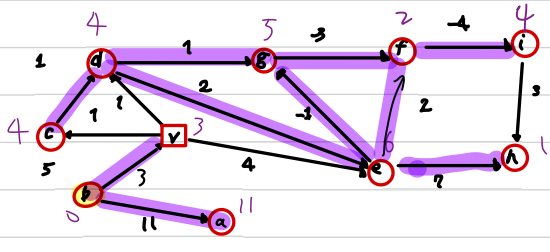
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 8
dist[g] = 5
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



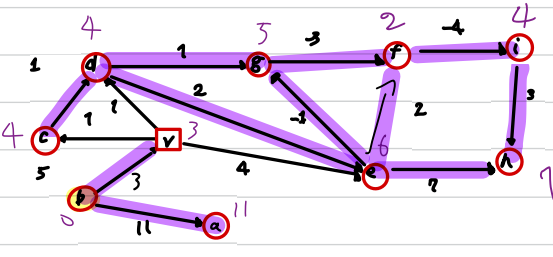
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 8
dist[g] = 5
dist[h] = ∞
dist[i] = ∞
dist[v] = 3



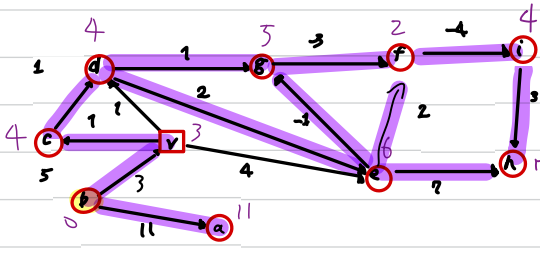
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 8
dist[g] = 5
dist[h] = 13
dist[i] =
dist[v] = 3



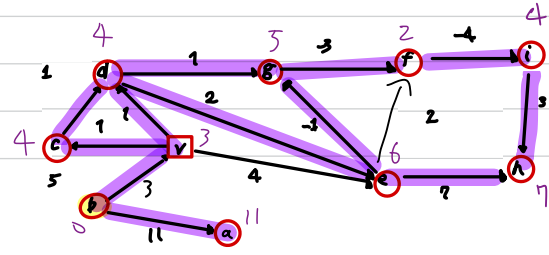
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = ∞
dist[g] = 5
dist[h] = 13
dist[i] = 4
dist[v] = 3

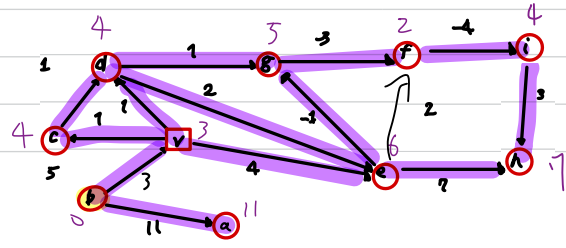


dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 7
dist[i] = 4
dist[v] = 3

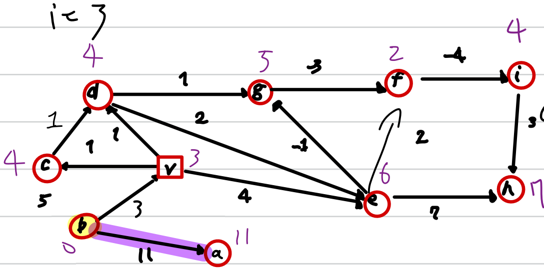


dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 7
dist[i] = 4
dist[v] = 3

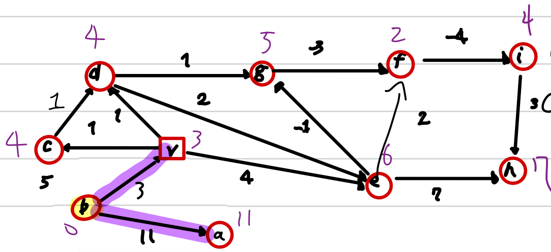




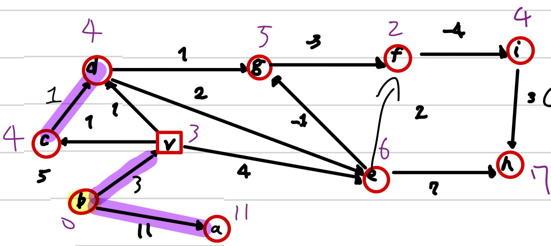
i = 3



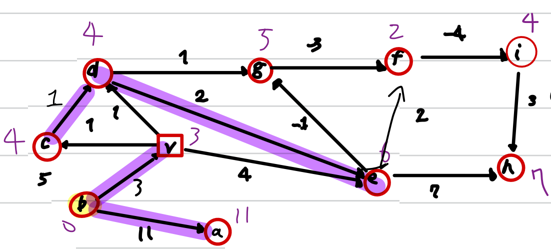
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



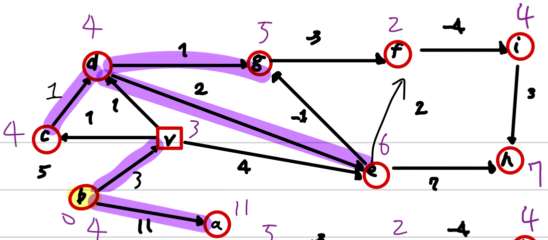
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



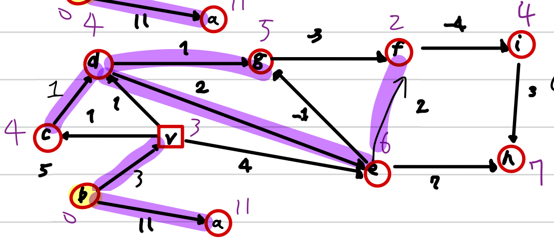
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



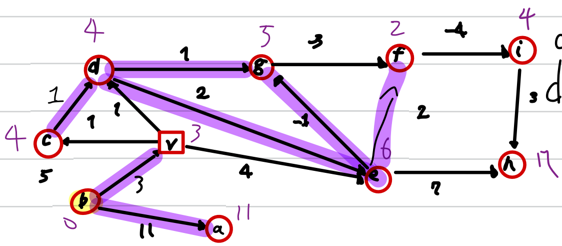
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



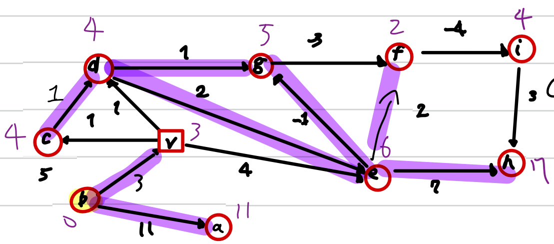
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



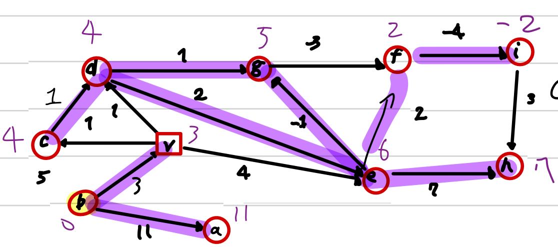
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



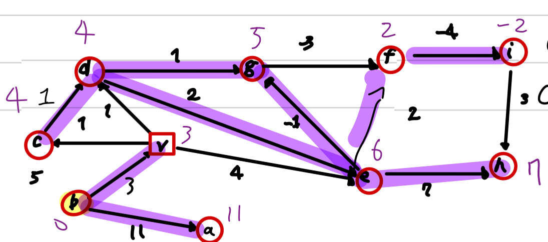
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



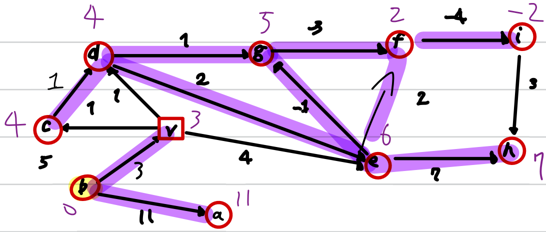
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



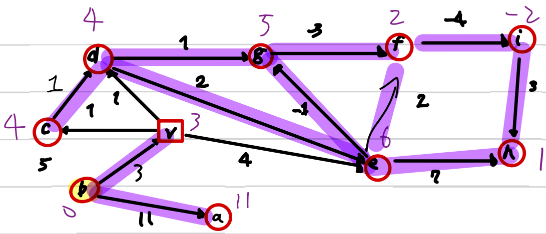
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = ∞
dist[v] = 3



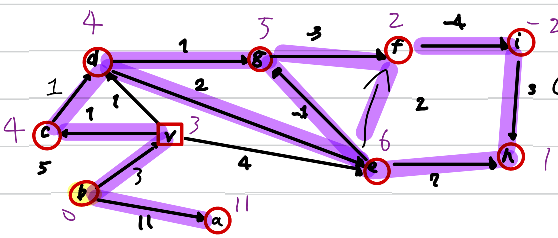
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = -2
dist[v] = 3



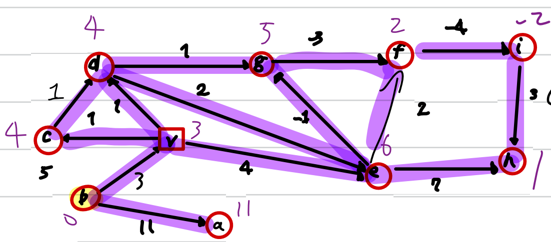
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = -2
dist[v] = 3



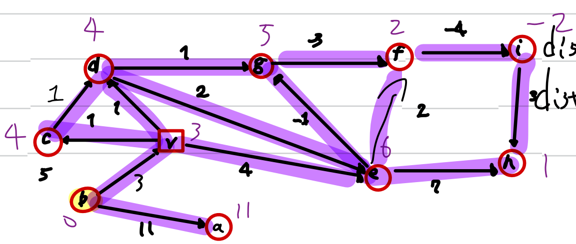
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 13
dist[i] = -2
dist[v] = 3



dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3

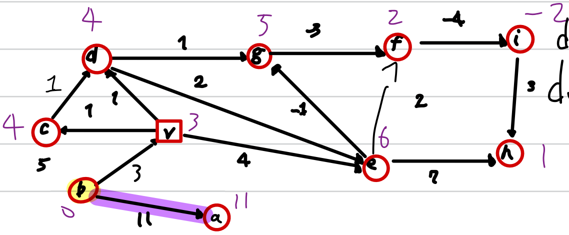


dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3

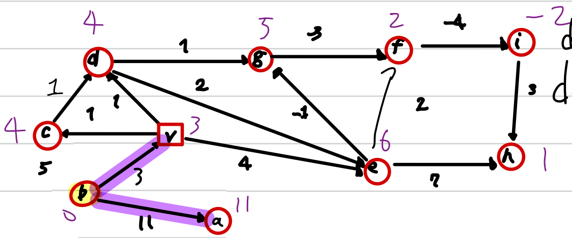


dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3

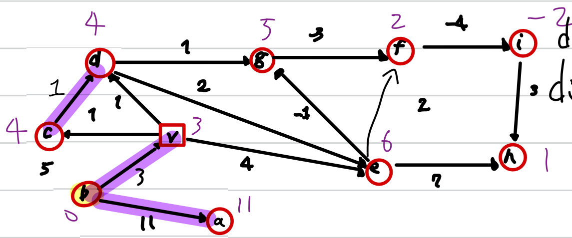
i=4



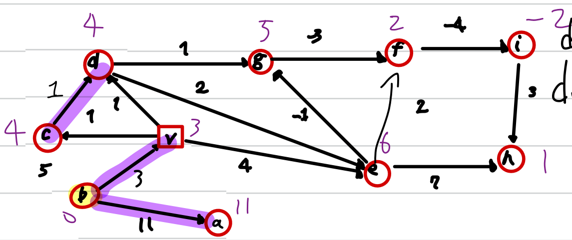
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



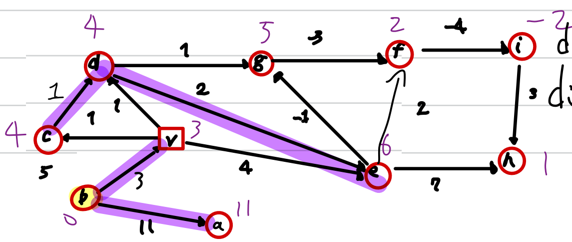
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



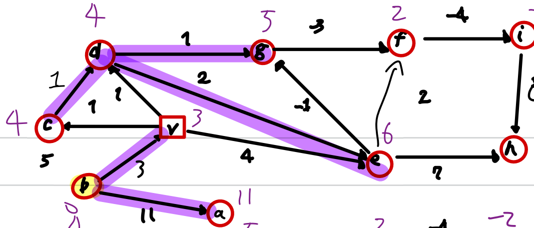
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



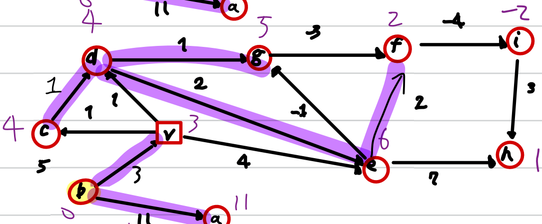
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



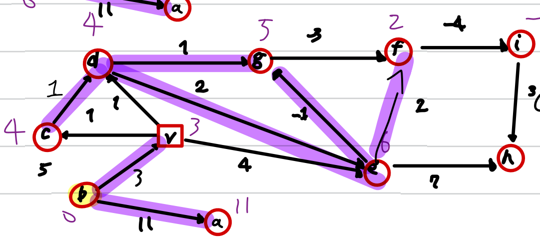
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



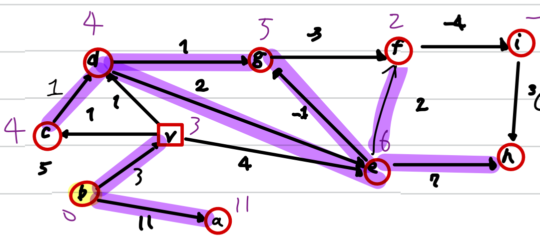
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



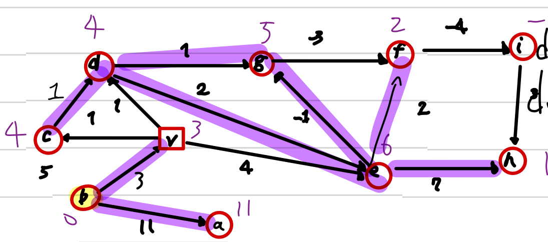
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



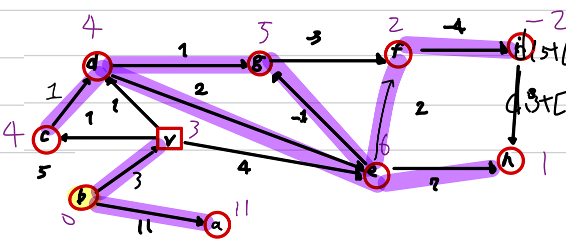
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



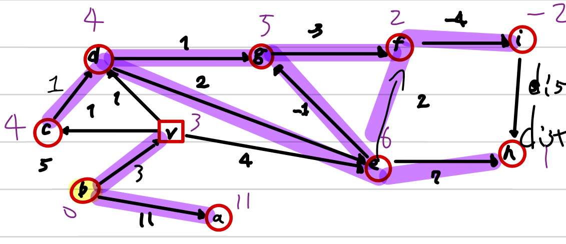
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



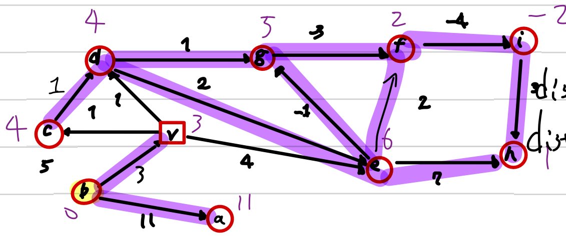
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



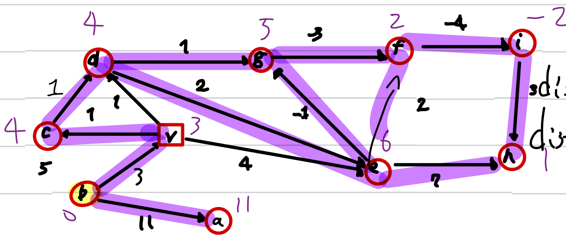
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



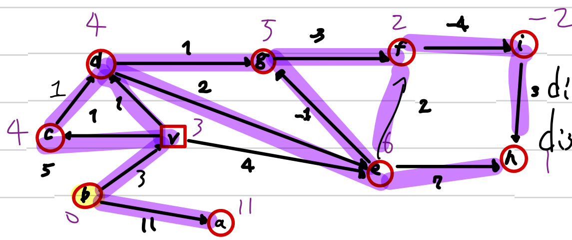
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



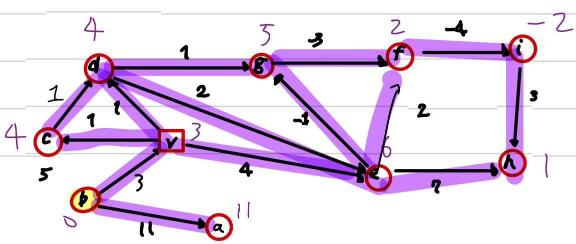
dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



dist[a] = 11
dist[b] = 0
dist[c] = 4
dist[d] = 4
dist[e] = 6
dist[f] = 2
dist[g] = 5
dist[h] = 1
dist[i] = -2
dist[v] = 3



When i=4, there is no change of the distance value, so I'll stop operating now.

5. Design an algorithm for the single source shortest path problem (SSSP) on directed acyclic graphs (DAGs) in $O(m+n)$ time.

I will use Topological sorting to design this algorithm.
First of all, I initialize distances to all vertices as infinity and 0 as distance to source vertex.
And then I'll use topological sorting.

The algorithm is :

1. Initialize distance D[] = {INF, INF, ...} and D[s] = 0 (s is the source vertex)
2. Create a topological order of all vertices.
3. Do all things below for all vertex u in topological order.

```
if( D[v] > D[u] + weight(u, v))
    D[v] = D[u] + weight(u, v)
```

v is the current vertex in the adjacency list of "u".
u is current vertex in the topological ordering.

· Basically, I check whether I can find the path to vertex "v" from source vertex through vertex "u" in a shorter length path than the current path. If I find the path is shorter, I update the values of D[v]

The time complexity is :

· Topological sorting ... This takes O(V + E) using DFS which V is number of vertices and E is the number of edges.

- Finding shortest path function ... this also takes O(V+E) since we use the while loop for traversing through stack and for loop for traversing through adjacency list of vertices.

So the total time complexity is O((V+E) + (V+E)) = O(V+E) ■