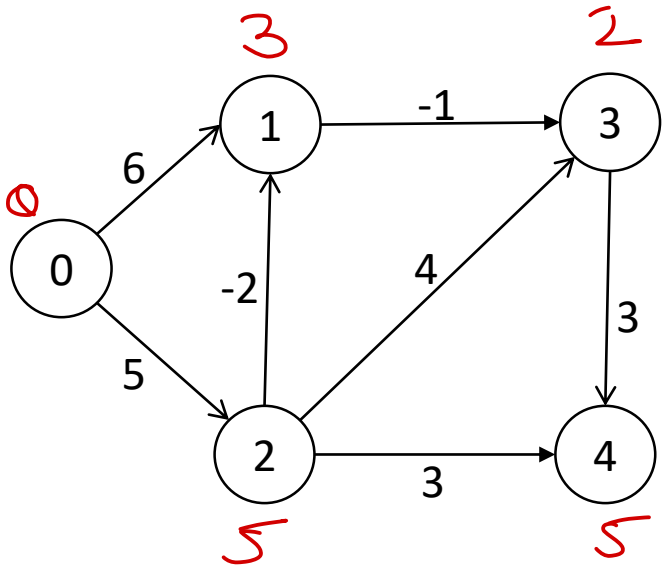
Data Structure & Algorithms

Nilesh Ghule

Bellman Ford Algorithm

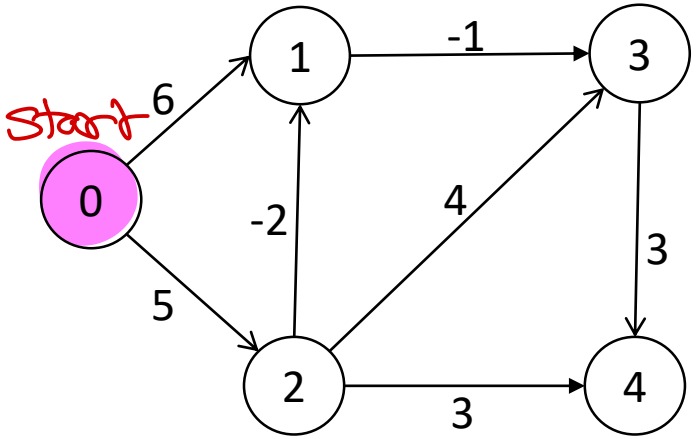
- Initializes distances from the source to all vertices as infinite and distance to the source itself as 0.
- Calculates shortest distance V-1 times: For each edge u-v, if dist[v] > dist[u] + weight of edge u-v, then update dist[v], so that dist[v] = dist[u] + weight of edge u-v.
- Check if negative edge in the graph: For each edge u-v, if dist[v] > dist[u] + weight of edge uv, then graph has –ve weight cycle.

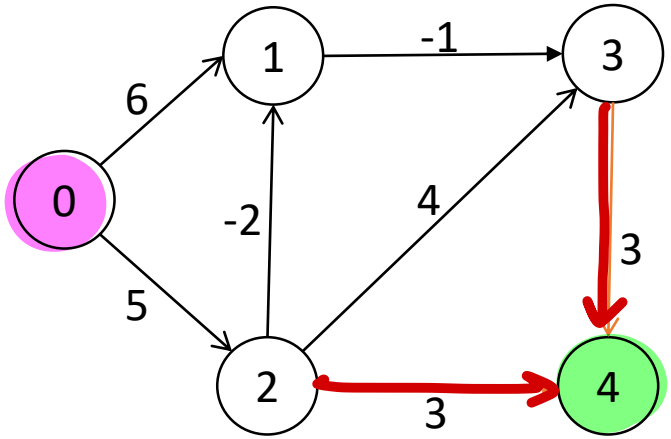


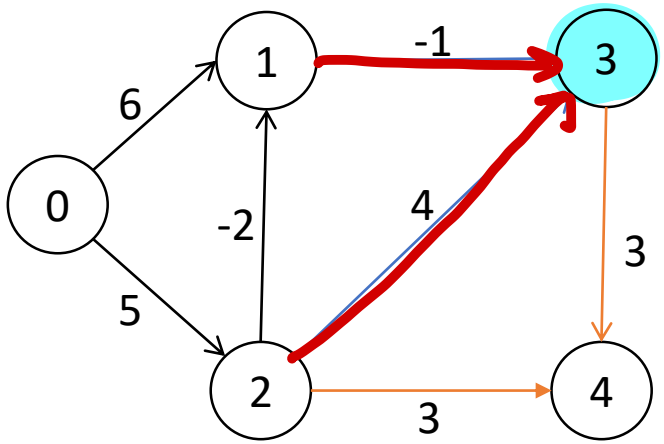
Src	Des	Wt
3	4	3
2	4	3
2	3	4
2	1	-2
1	3	-1
0	2	5
0	1	6

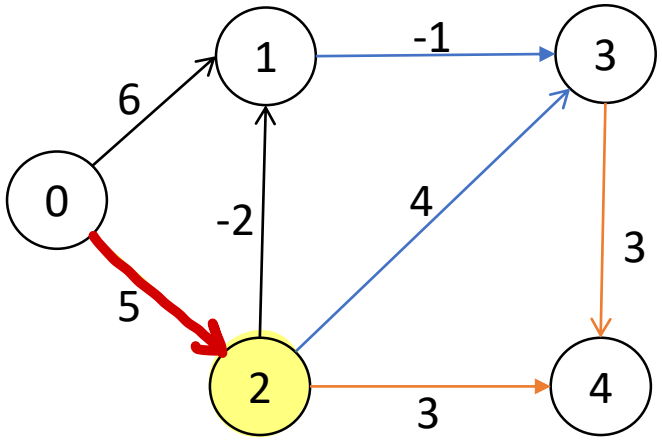
	0	1	2	3	4
Pass 1:	0	6	5	∞	∞
Pass 2:	0	3	5	2	8
Pass 3:	0	3	5	2	5
Pass 4:	0	3	5	2	5

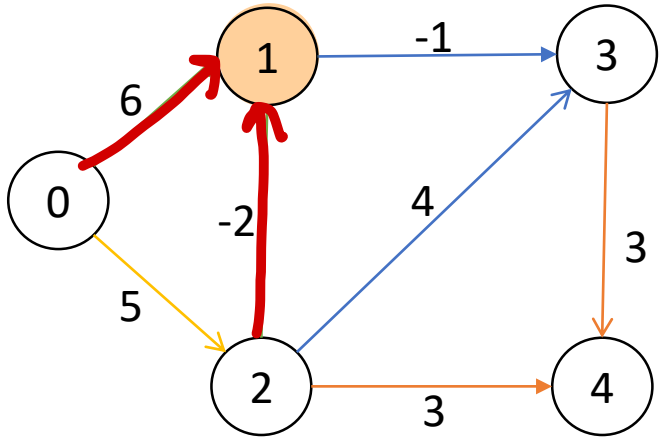
Bellman Ford Algorithm



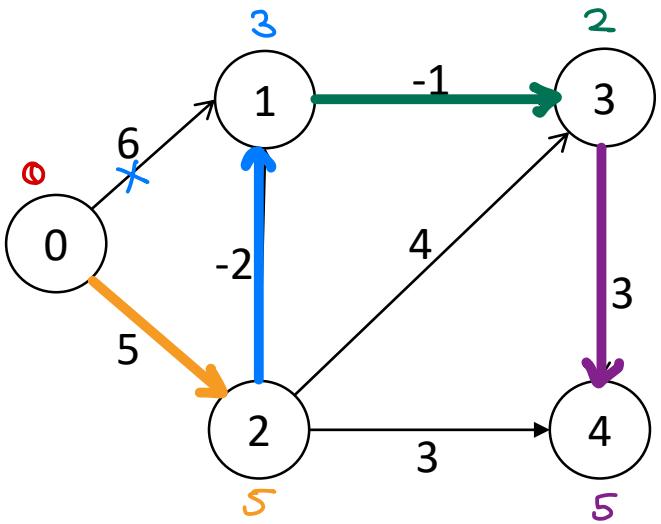


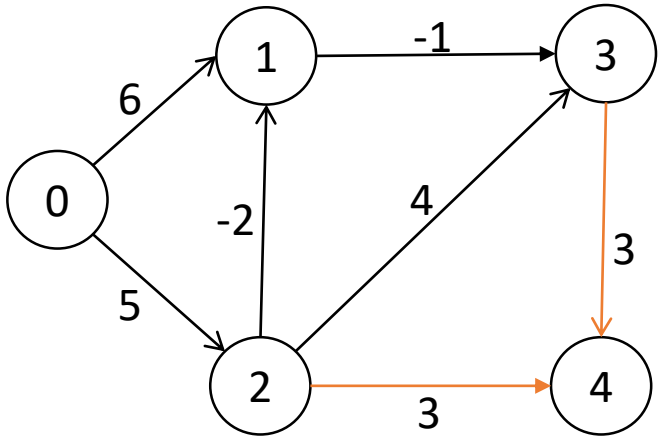


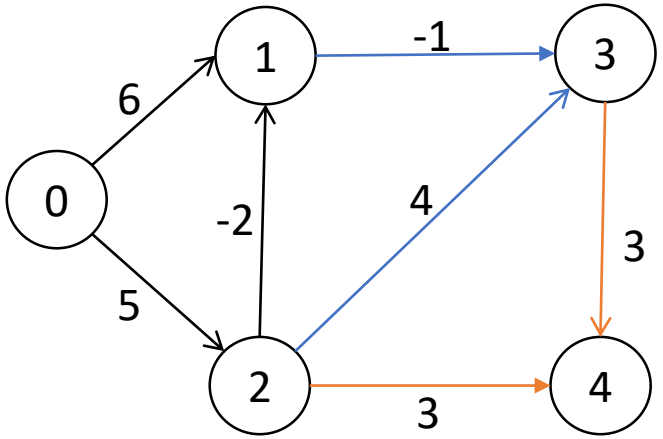


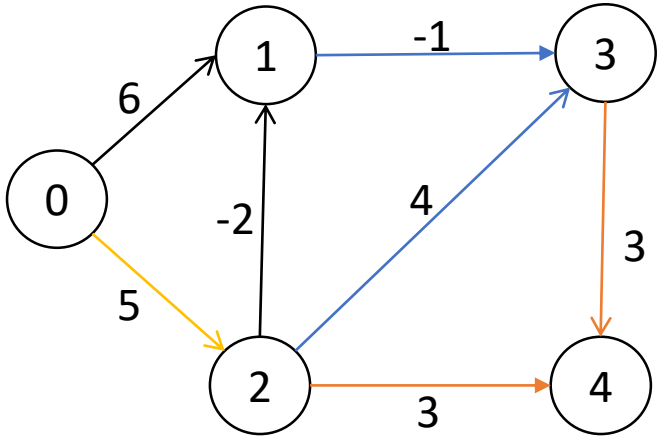


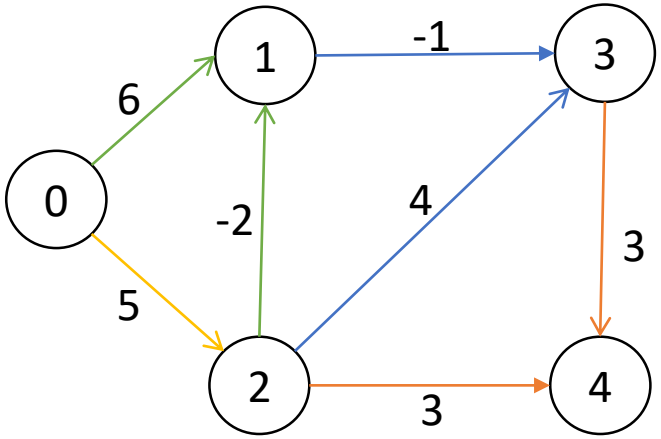
Bellman Ford Algorithm





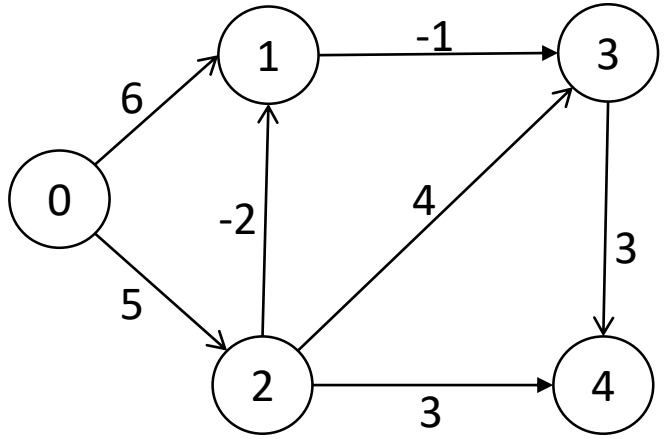






Bellman Ford Algorithm

	0	1	2	3	4
Pass 0	0	∞	∞	∞	∞
Pass 1	0	6	5	∞	∞
Pass 2	0	3	5	2	8
Pass 3	0	3	5	2	5
Pass 4	0	3	5	2	5



Src	Dest	Wt
3	4	3
2	4	3
2	3	4
2	1	-2
1	3	-1
0	2	5
0	1	6

Warshall Floyd Algorithm

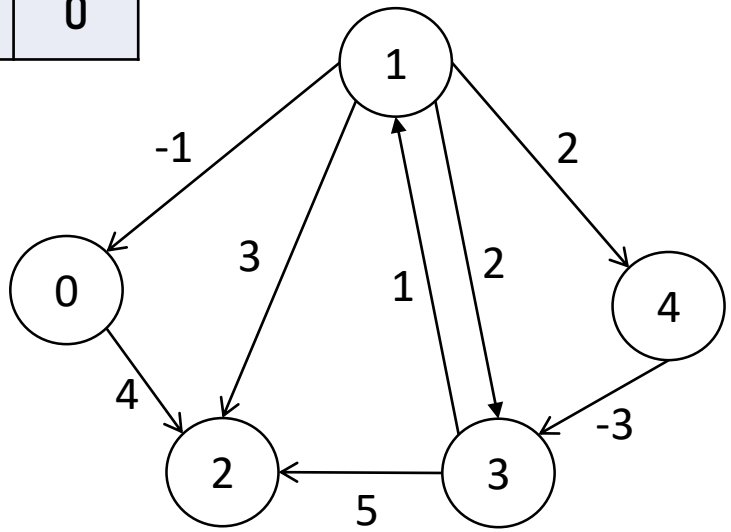
- Algorithm

1. Create distance matrix to keep distance of every vertex from each vertex. Initially assign it with weights of all edges among vertices (i.e. adjacency matrix).

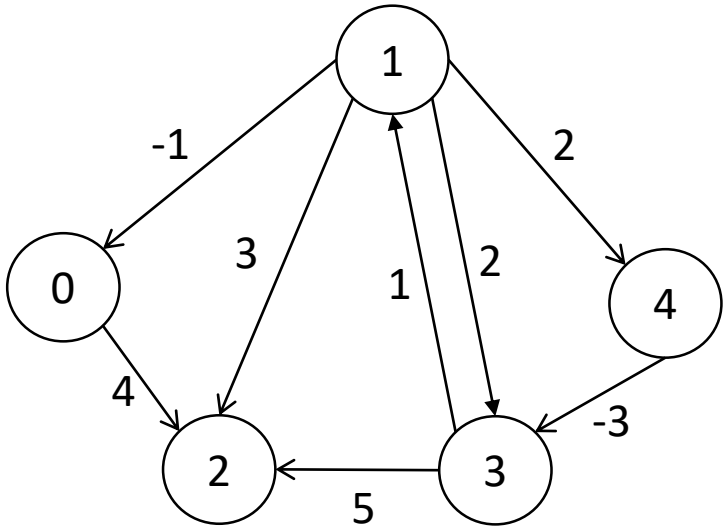
2. Consider each vertex (i) in between pair of any two vertices (s, d) and find the optimal distance between s & d considering intermediate vertex i.e. dist(s,d) = dist(s,i) + dist(i,d), if dist(s,i) + dist(i,d) < dist(s,d).

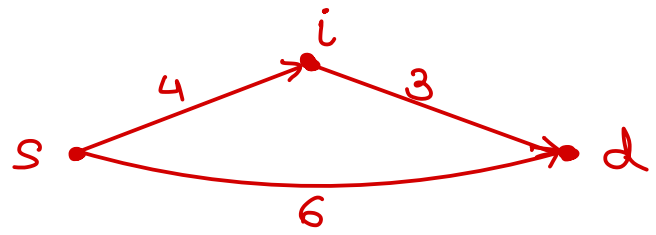
	0	1	2	3	4
0	0	∞	4	∞	∞
1	-1	0	3	2	2
2	∞	∞	0	∞	∞
3	∞	1	5	0	∞
4	∞	∞	∞	-3	0

← initial state

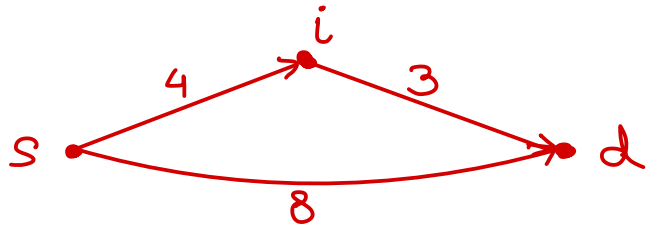


Warshall Floyd Algorithm





dist[s][d] = 6



dist[s][d] = 7

= dist[s][i] + dist[i][d]

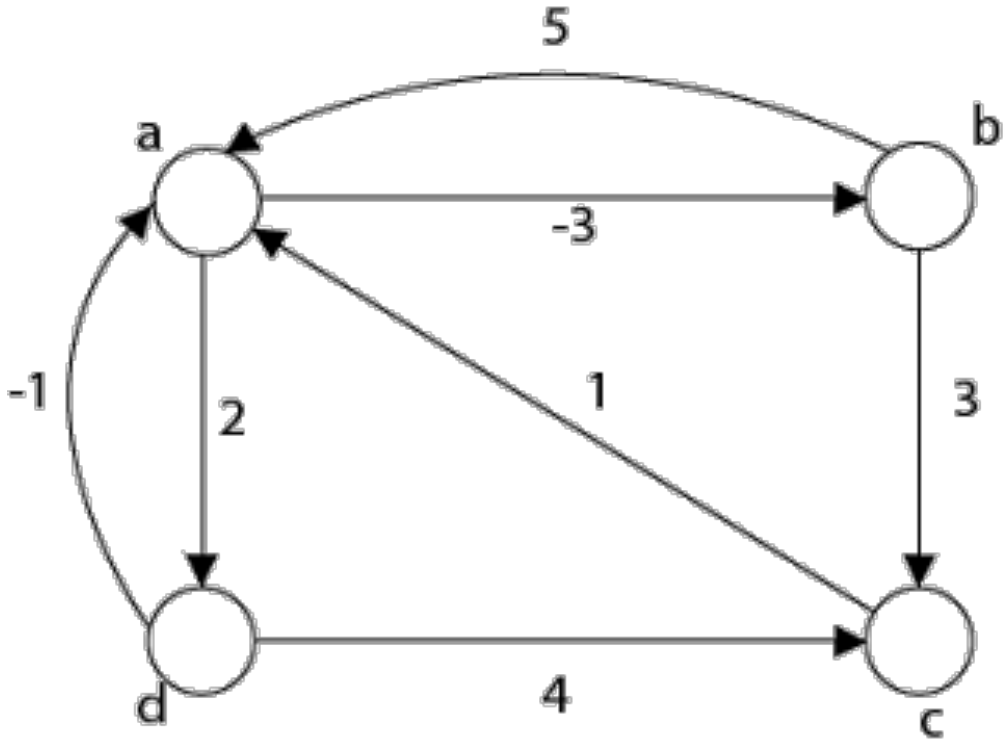
```
for(i=0;i<V;i++){
    for(s=0;s<V;s++){
        for(d=0;d<V;d++){
            if(dist[s][i]+dist[i][d] < dist[s][d])
                dist[s][d]=dist[s][i]+dist[i][d];
        }
    }
}
```

$O(V^3)$

Johnson's Algorithm

- Time complexity of Warshall Floyd is $O(V^3)$.
- Applying Dijkstra's algorithm on V vertices will cause time complexity O(V * V log V). This is faster than Warshall Floyd.
- However Dijkstra's algorithm can't work with –ve weight edges.
- Johnson use Bellman ford to reweight all edges in graph to remove –ve edges. Then apply Dijkstra to all vertices to calculate shortest distance. Finally reweight distance to consider original edge weights.
- Time complexity of the algorithm:

$$O(VE + V^2 \log V).$$

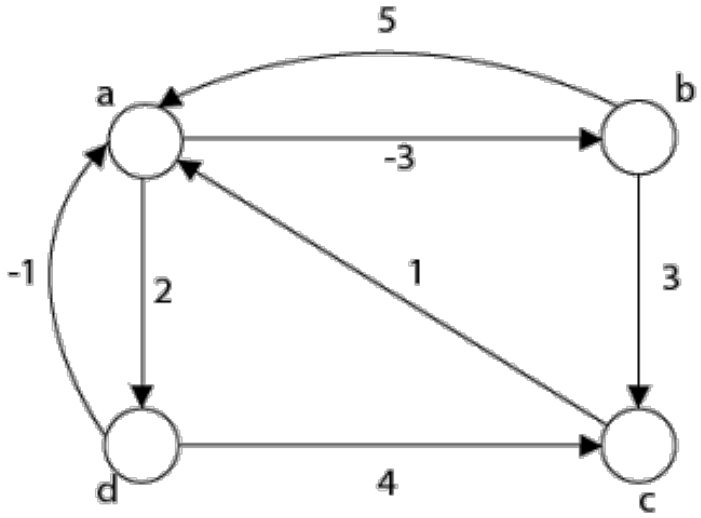


Johnson's Algorithm

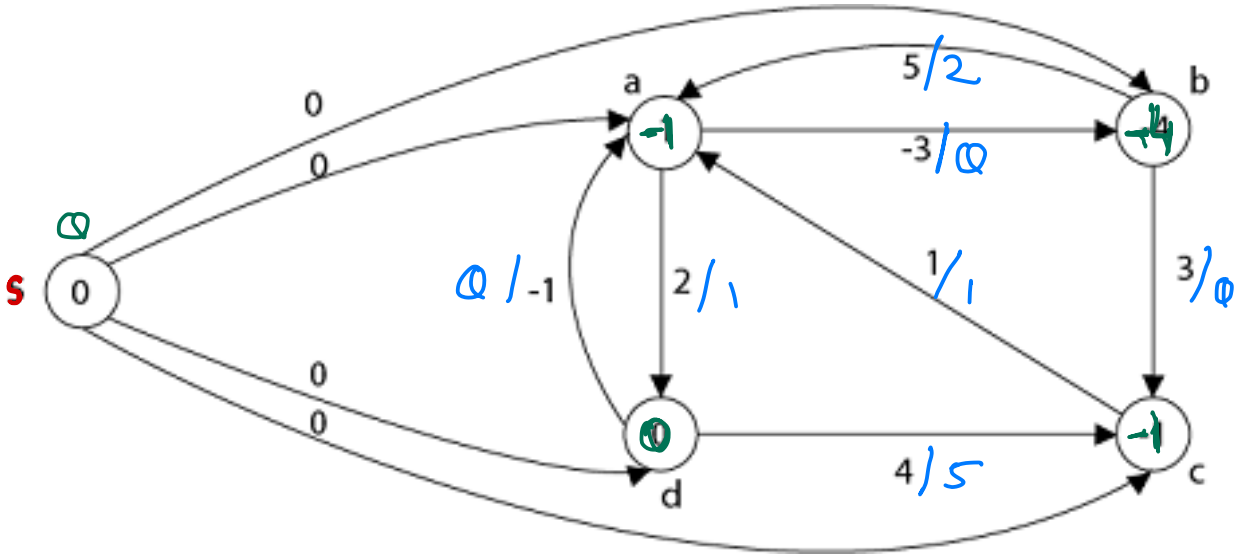
1. Add a vertex (s) into a graph and add edges from it all other vertices, with weight 0.
2. Find shortest distance of all vertices from (s) using Bellman Ford algorithm.

   a = -1, b = -4, c = -1, d = 0 and s = 0
3. Reweight all edges (u, v) in the graph, so that, they become non negative.

   weight(u, v) = weight(u, v) + d(u) – d(v)



- w(a, b) = 0
- w(b, a) = 2
- w(b, c) = 0
- w(c, a) = 1
- w(d, c) = 5
- w(d, a) = 0
- w(a, d) = 1

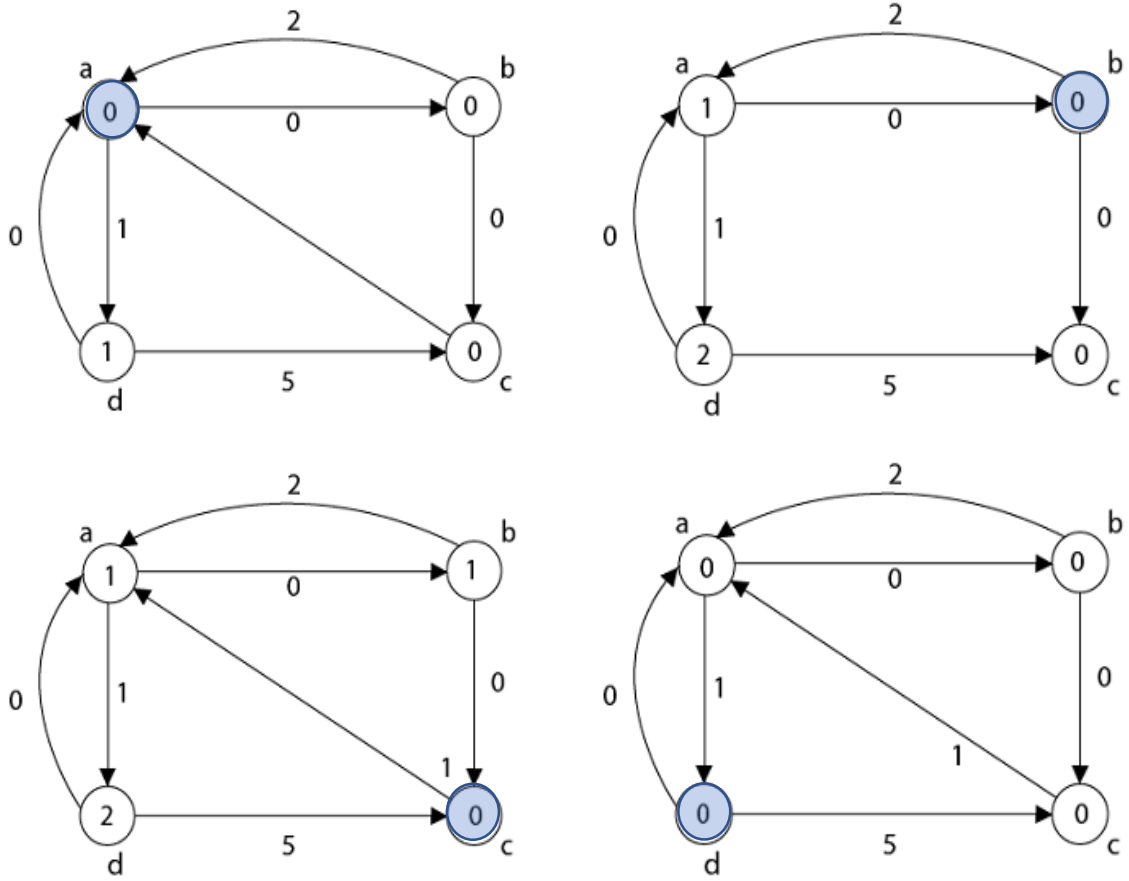


Johnson's Algorithm

4. Apply Dijkstra on each vertex to calculate shortest distance to all other vertices.
5. Reweight all distances to consider original weights.

$$dist(u, v) = dist(u, v) + d(v) - d(u)$$

	a	b	c	d
a	0 -> 0	0 -> -3	0 -> 0	1 -> 2
b	1 -> 4	0 -> 0	0 -> 3	2 -> 6
c	1 -> 1	1 -> -2	0 -> 0	2 -> 3
d	0 -> -1	0 -> -4	0 -> -1	0 -> 0



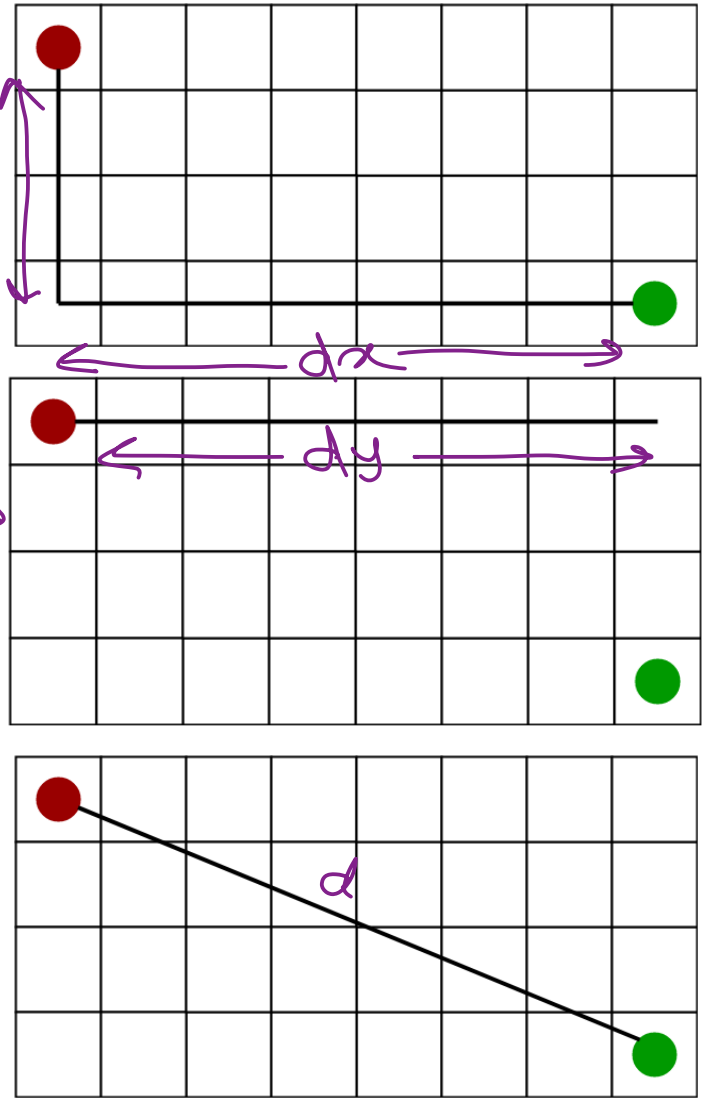
A* Search Algorithm

- Point to point approximate shortest path finder algorithm.
- This algorithm is used in artificial intelligence.
- Commonly used in games or maps to find shortest distance in faster way.
- It is modification of BFS.
- Put selected adjacent vertices on queue, based on some heuristic.
- A math function is calculated for vertices
  - f(v) = g(v) + h(v) → vertex with min f(v) is picked.
  - g(v) → cost of source to vertex v
  - *h(v)* → estimated cost of vertex v to destination.

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	0	1	1	1
1	1	1	1	0	1	1	1	0	1	1
2	1	1	1	0	1	1	0	1	0	1
3	0	0	1	0	1	0	0	0	0	1
4	1	1	1	0	1	1	1	0	1	0
5	1	0	1	1	1	1	0	1	0	0
6	1	0	0	0	0	1	0	0	0	1
7	1	0	1	1	1	1	0	1	1	1
8	1	1	1	0	0	0	1	0	0	1
9	1	0	1	0	0	0	0	0	0	1

A* Search Algorithm

- h(v) represent heuristic and depends on problem domain. Three common techniques to calculate heuristic:
- Manhattan distance
  - When moves are limited in four directions only.
  - h = dx + dy
- Diagonal distance
  - When moves are allowed in all eight directions (one step).
  - h = MAX(dx, dy)
- Euclidean distance
  - When moves are allowed in any direction.
  - h = $\sqrt{dx^2 + dy^2}$
- Note that heuristic may result in longer paths in typical cases.



A* search algorithm

- Start point g(v) = 0, h(v) = 0 & f(v) = 0.
- Push start point vertex on a priority queue (by f(v)).
- Until queue is empty
  - Pop a point (v) from queue.
  - Add v into the path.
  - For each adjacent point (u)
    - If u is destination, build the path.
    - If u is invalid or already on path or blocked, skip it.
    - Calculate newg = g(v) + 1, newh = *heuristic* and newf = newg + newh.
    - If newf is less than f(u), f(u) = newf and also parent(u) = v. Rearrange elements in priority queue.

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	0	1	1	1
1	1	1	1	0	1	1	1	0	1	1
2	1	1	1	0	1	1	0	1	0	1
3	0	0	1	0	1	0	0	0	0	1
4	1	1	1	0	1	1	1	0	1	0
5	1	0	1	1	1	1	0	1	0	0
6	1	0	0	0	0	1	0	0	0	1
7	1	0	1	1	1	1	0	1	1	1
8	1	1	1	0	0	0	1	0	0	1
9	1	0	1	0	0	0	0	0	0	1

A* Search Algorithm

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	0	1	1	1
1	1	1	1	0	1	1	1	0	1	1
2	1	1	1	0	1	1	0	1	0	1
3	0	0	1	0	1	0	0	0	0	1
4	1	1	1	0	1	1	1	0	1	0
5	1	0	1	1	1	1	0	1	0	0
6	1	0	0	0	0	1	0	0	0	1
7	1	0	1	1	1	1	0	1	1	1
8	1	1	1	0	0	0	1	0	0	1
9	1	0	1	0	0	0	0	0	0	1

	0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	1	1	0	1	1	1
1	1	1	1	0	1	1	1	0	1	1
2	1	1	1	0	1	1	0	1	0	1
3	0	0	1	0	1	0	0	0	0	1
4	1	1	1	0	1	1	1	0	1	0
5	1	0	1	1	1	1	0	1	0	0
6	1	0	0	0	0	1	0	0	0	1
7	1	0	1	1	1	1	0	1	1	1
8	1	1	1	0	0	0	1	0	0	1
9	1	0	1	0	0	0	0	0	0	1

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>