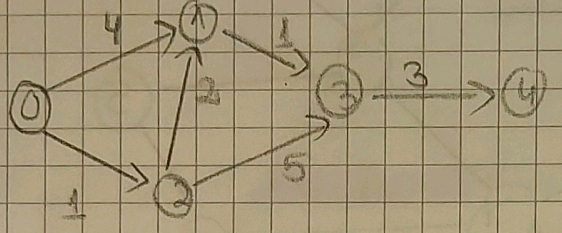
DIJKSTRA'S ALGORITHM

Complexity: $O(E * \log(V))$

2> Single Source Shortest Path (SSSP) algorithm for graphs with non-negative edge weights

3> once a node has been visited its optimal distance cannot be improved (Greedy approach)

Lazy Dijkstra's



dist - optimal distance from the start node

0	1	2	3	4
∞	∞	∞	∞	∞

^ we assume every node is unreachable

(index, dist) key value pairs

1) (0,0)
2) (1,4)
3) (2,1) *
4) (1,3)
5) (3,6) ⇒ finished with 2 ⇒ we switch to 1
6) (3,4) ⇒ finished with 1
7) (4,7)

	0	1
1)	0	∞
2)	0	4
3)	0	4
	0	3
	0	3
	0	3
	0	3

* After we finish visiting 0, we visit 2 as it's the most promising node (its dist value is the minimum one in the queue)

Best distance from 2 to 1 is dist[2] + edge cost = 3 < 4

After finishing with (1,3), we ~~remove~~ ignore (1,4) bc the dist is greater and the vertex has already been used

PQ stores (node index, best distance) pairs sorted by minimum distance

Code sequence

g - adjacency list
n - no. of nodes
s - index of start node

```
function dijkstra(g, n, s):
    vis = [false, false, ..., false] # size n
    dist = [∞, ∞, ..., ∞] # size n
    pq = empty priority queue
    pq.insert((s, 0))
```

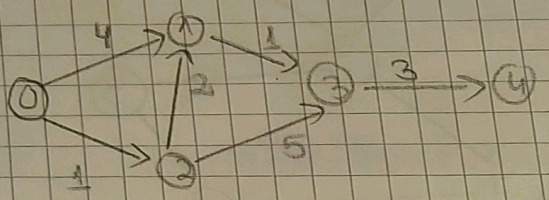
DIJKSTRA'S ALGORITHM

Complexity: $O(E * \log(V))$

2) Single Source Shortest Path (SSSP) algorithm for graphs with non-negative edge weights

3) once a node has been visited its optimal distance cannot be improved (Greedy approach)

Lazy Dijkstra's



dist - optimal distance from the start node

0	1	2	3	4
∞	∞	∞	∞	∞

↑ we assume every node is unreachable

(index, dist) key value pairs

1) (0,0)
2) (1,4)
3) (2,1) *
4) (1,3)
5) (3,6) ⇒ Pruned with 2
   ⇒ we switch to 1
6) (3,4) ⇒ Pruned with 4
7) (4,7)

	dist				
	0	1	2	3	4
1)	0	∞	∞	∞	∞
2)	0	4	∞	∞	∞
3)	0	4	1	∞	∞
	0	3	1	∞	∞
	0	3	1	6	∞
	0	3	1	4	∞
	0	3	1	4	7

Code sequence

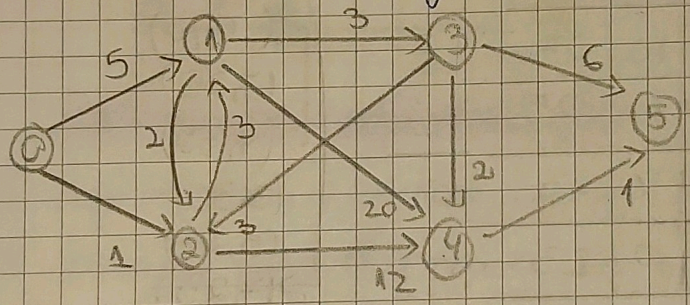
g - adjacency list
n - no. of nodes
s - index of start node

```
function dijkstra(g, n, s):
    vis = [false, false, ..., false] # size n
    dist = [∞, ∞, ..., ∞] # size n
    pq = empty priority queue
    pq.insert((s, 0))
```

```
while pq.size() != 0:
    index, minValue = pq.poll() // remove the next pair
    vis[index] = true
    if (dist[index] < minValue): continue
    for (edge : g[index]):
        if vis[edge.to]: continue # skip visited neighbours
        newDist = dist[index] + edge.cost
        if newDist < dist[edge.to]:
            dist[edge.to] = newDist
            pq.insert((edge.to, newDist))
return dist
```

Lazy Dijkstra's using an Indexed Priority Queue

2> avoid inserting duplicate key-value pairs



(index, dist)
~~(0,0)~~
~~(1,5)~~ (1,4)
~~(2,1)~~
~~(4,13)~~ (4,9)
~~(3,7)~~
(5,13) (5,10)

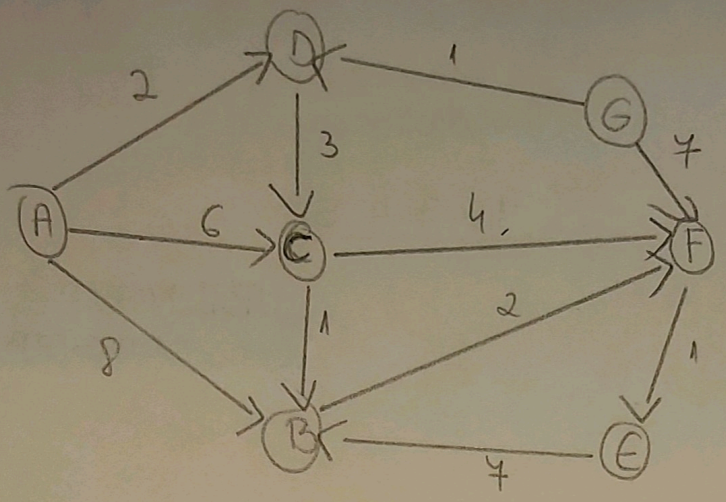
dist

0	1	2	3	4	5
0	∞	∞	∞	∞	∞
0	5	∞	∞	∞	∞
0	5	1	∞	∞	∞
0	5	1	∞	13	∞
0	4	1	∞	13	∞
0	4	1	7	13	∞
0	4	1	7	9	∞
0	4	1	7	9	13
0	4	1	7	9	10

for 2
dist[1] + cost = 2 < 1

for 4
dist[4] + cost = 24 < 13

DIJKSTRA



Find the minimum cost path from vertex A to all other vertices using the Dijkstra algorithm.

a) For each iteration, the current ~~loop~~ vertex, the content of the queue and the distance and prev dictionaries.

	x	y	dist dictionary	q: priority queue	prev: dictionary
initialization			A B C D E F G 0	←(A,0)	A B C D E F G 0
iteration 1	x=A	y=B y=C y=D	A B C D E F G 0 8 0 8 6 0 8 6 2	←(B,8) ←(C,6),(B,8)← ←(D,2),(C,6),(B,8)	A B C D E F G 0 A 0 A A 0 A A A
iteration 2	x=D	y=C	A B C D E F G 0 8 5 2	←(C,6),(B,8)← ←(C,5),(B,8)←	A B C D E F G 0 A D A
iteration 3	x=C	y=B y=F	A B C D E F G 0 6 5 2 0 6 5 2 9	←(B,8)← ←(B,6)← ←(B,6),(F,9)←	A B C D E F G 0 C D A 0 C D A C
iteration 4	x=B	y=F	A B C D E F G 0 6 5 2 8	←(F,9)← ←(F,8)←	A B C D E F G 0 C D A B
iteration 5	x=F	y=E	A B C D E F G 0 6 5 2 9 8	← ← ←(E,9)	A B C D E F G 0 C D A F B
iteration 6	x=E	y=B	A B C D E F G 0 6 5 2 9 8	← ←	A B C D E F G 0 C D A F B

A→B : A→D→C→B : cost: 6

A→C : A→D→C : cost 5

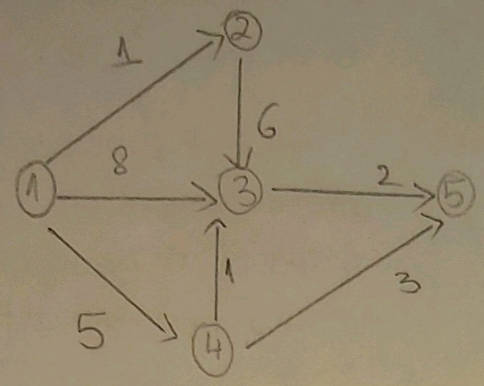
A→D : A→D : cost 2

A→E : A→D→C→B→F→E . cost 9

A→F : A→D→C→B→F : cost 8

A→G : there is no such path

- Find the minimum cost path from all vertices to vertex 5 using Dijkstra in reverse



Prev dictionary

1 2 3 4 5

BACKWARDS DIJKSTRA

	x	y	dist dictionary	Q: priority queue	prev dict
init			~~0~~ 1 2 3 4 5 0	←(5,0)	1 2 3 4 5 0
it. 1	5	3 4	1 2 3 4 5 2 0 1 2 3 4 5 2 3 0	←(3,2) ←(3,2),(4,3)	1 2 3 4 5 5 0 5 5 0
it. 2	3	1 2 4	1 2 3 4 5 10 2 3 0 10 8 2 3 0 10 8 2 3 0	←(4,3),(1,10) ←(4,3),(2,8),(1,10) -unchanged	1 2 3 4 5 3 5 5 0 3 3 5 5 0 -unchanged
it 3	4	1	1 2 3 4 5 8 8 2 3 0	←(2,8),(1,8)←	1 2 3 4 5 4 3 5 5 0
it 4	2	1	1 2 3 4 5 8 8 2 3 0	←(1,8)←	1 2 3 4 5 4 3 5 5 0
it. 5	1	-	1 2 3 4 5 8 8 2 3 0	← ←	1 2 3 4 5 4 3 5 5 0

The minimum cost path from

1 to 5: 1→4→5 cost 8

2 to 5: 2→3→5 cost 8

3 to 5: 3→5 cost 2

4 to 5: 4→5 cost 3