COMP2521 24T1

Graphs (VI)
Dijkstra's Algorithm

Kevin Luxa
cs2521@cse.unsw.edu.au

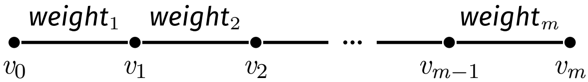
shortest path
dijkstra's algorithm

Shortest Path

In a weighted graph...

A path is a sequence of edges
connected end-to-end

(v_0, v_1, w_1), (v_1, v_2, w_2), ... , (v_{m-1}, v_m, w_m)



The cost of a path is
the sum of edge weights along the path

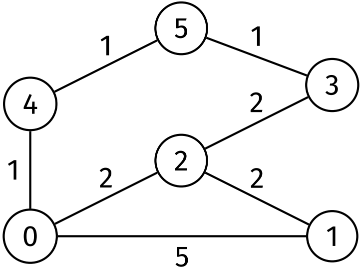
The shortest path between two vertices s and t is
the path from s to t with minimum cost

Variations on shortest path problem:

- Source-target shortest path
  - Shortest path from source vertex s to target vertex t
- Single-source shortest path
  - Shortest path from source vertex s to all other vertices
- All-pairs shortest path
  - Shortest path between all pairs of source and target vertices

Shortest Path

In a weighted graph,
a path with more edges may be "shorter" than
a path with fewer edges



Invented by Dutch computer scientist
Edsger W. Dijkstra in 1956

Dijkstra's Algorithm

Dijkstra's algorithm
is used to find the shortest path
in a weighted graph with non-negative weights

Data Structures

Data structures used in Dijkstra's algorithm:

- Distance array (`dist`)
  - To keep track of shortest currently known distance to each vertex
- Predecessor array (`pred`)
  - Same purpose as in BFS/DFS
  - To keep track of the predecessor of each vertex on the shortest currently known path to that vertex
  - Used to construct the shortest path
- Set of vertices
  - Stores unexplored vertices

Algorithm

1. Create and initialise data structures
   - Create distance array, initialised to infinity
     - In C, can use `INT_MAX` (from `<limits.h>`)
   - Create predecessor array, initialised to -1
   - Initialise set of vertices to contain all vertices
2. Set distance of source vertex (s) to 0
3. While set of vertices is not empty:
   1. Remove vertex from vertex set with smallest distance in distance array
      - Let this vertex be v
   2. **Explore** v - that is, for each edge v – w:
      - Check if using this edge gives a shorter path to w
      - If so, update w's distance and predecessor - this is called **edge relaxation**

Edge Relaxation

During Dijkstra's algorithm, the `dist` and `pred` arrays:

- contain data about the shortest path discovered *so far*
- need to be updated if a shorter path to some vertex is found
  - this is done via **edge relaxation**

Edge Relaxation

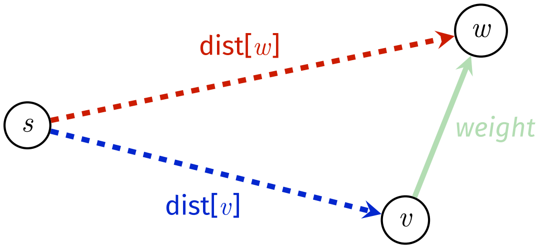
Suppose we are considering edge $(v, w, weight)$.

Edge Relaxation

Suppose we are considering edge $(v, w, weight)$.

We have the following data:

- `dist[`v`]` - length of shortest known path from s to v
- `dist[`w`]` - length of shortest known path from s to w (which may be ∞)

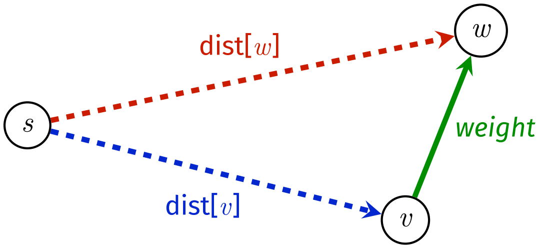


Edge Relaxation

Suppose we are considering edge $(v, w, weight)$.

We have the following data:

- `dist`$[v]$ - length of shortest known path from s to v
- `dist`$[w]$ - length of shortest known path from s to w (which may be ∞)

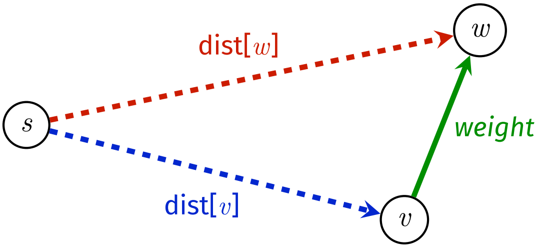


In edge relaxation, we take the shortest known path from s to v and *extend* it using edge $(v, w, weight)$ to create a *new* path from s to w.

Edge Relaxation

Now we have two paths from s to w:

- Shortest known path
- New path via v



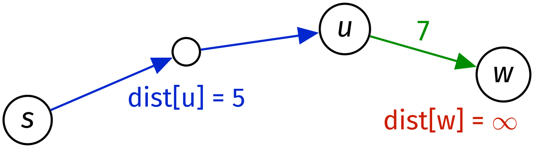
If the new path is shorter, then we update `dist`[w] and `pred`[w].

```
if dist[v] + weight < dist[w]:
    dist[w] = dist[v] + weight
    pred[w] = v
```

Edge Relaxation

Example 1

Before relaxation along $(u, w, 7)$

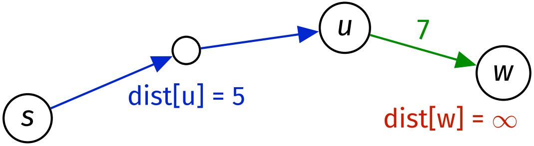


	…	$[u]$	…	$[w]$
dist	…	5	…	∞
pred	…	…	…	-1

Edge Relaxation

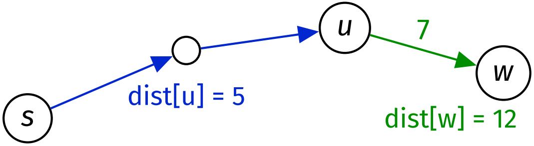
Example 1

Before relaxation along $(u, w, 7)$



	...	$[u]$	...	$[w]$
dist	...	5	...	∞
pred	...	...	...	-1

After relaxation along $(u, w, 7)$

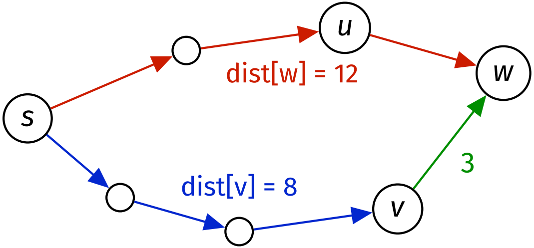


	...	$[u]$	...	$[w]$
dist	...	5	...	12
pred	...	...	...	u

Edge Relaxation

Example 2

Before relaxation along $(v, w, 3)$

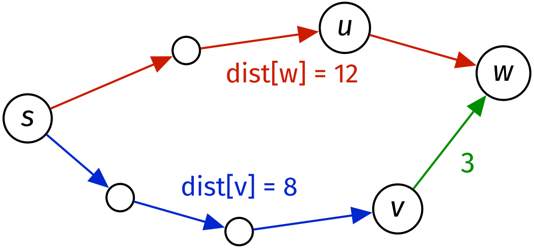


	...	[u]	[v]	[w]
dist	...	5	8	12
pred	...	...	...	u

Edge Relaxation

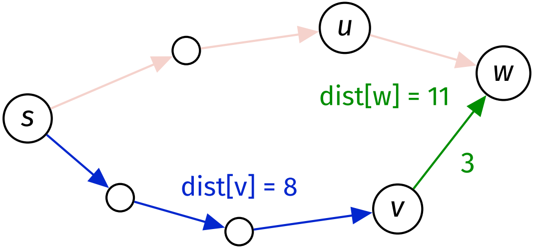
Example 2

Before relaxation along $(v, w, 3)$



	...	[u]	[v]	[w]
dist	...	5	8	12
pred	...	...	...	u

After relaxation along $(v, w, 3)$



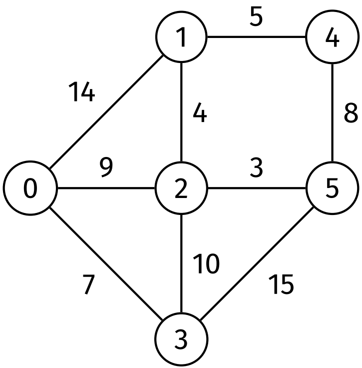
	...	[u]	[v]	[w]
dist	...	5	8	11
pred	...	...	...	v

Pseudocode

```
dijkstraSSSP(G, src):
    Input: graph G, source vertex src

    create dist array, initialised to ∞
    create pred array, initialised to -1
    create vSet containing all vertices of G

    dist[src] = 0
    while vSet is not empty:
        find vertex v in vSet such that dist[v] is minimal
        remove v from vSet
        for each edge (v, w, weight) in G:
            relax along (v, w weight)
```

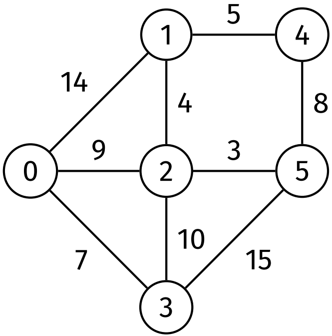
Example

Dijkstra's algorithm starting at 0



Example

Initialisation

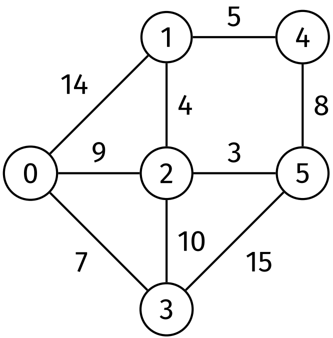



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

Example

After first iteration (v = 0)

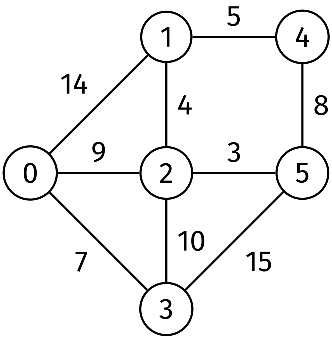



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Example

After second iteration (v = 3)

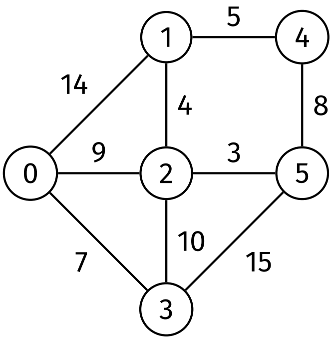



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

Example

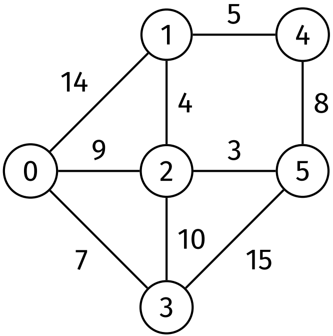
After third iteration (v = 2)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

After fourth iteration (v = 5)

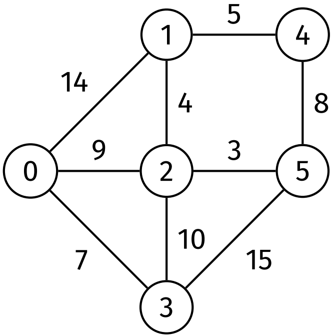



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

Example

After fifth iteration (v = 1)

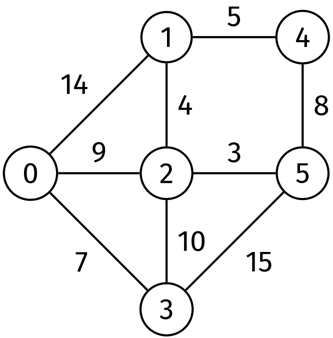



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

Example

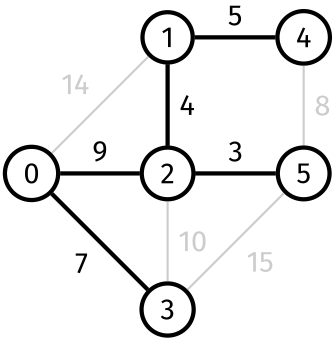
After sixth iteration (v = 4)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

Example



Done

```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

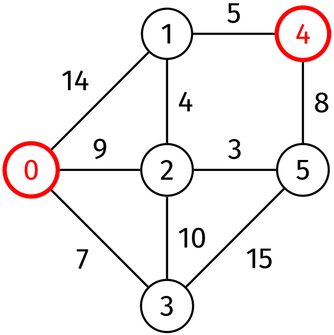
	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

Path Finding

The shortest path from the source vertex to any other vertex can be constructed by tracing backwards through the predecessor array (like for BFS)

Example: Shortest path from 0 to 4

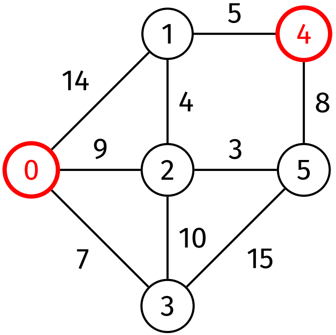
0 4



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Example: Shortest path from 0 to 4

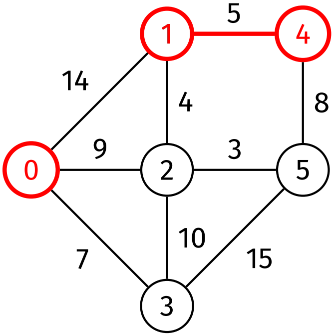
0 4



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Example: Shortest path from 0 to 4

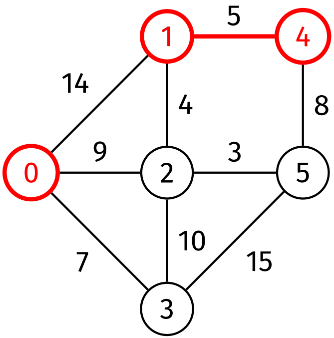
$$0 \qquad 1 \longrightarrow 4$$



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Example: Shortest path from 0 to 4

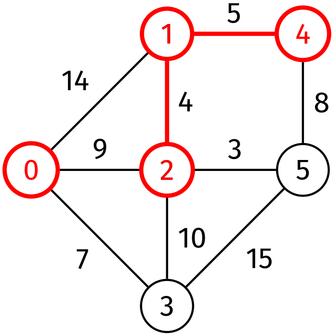
0 1 $\longrightarrow$ 4



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Example: Shortest path from 0 to 4

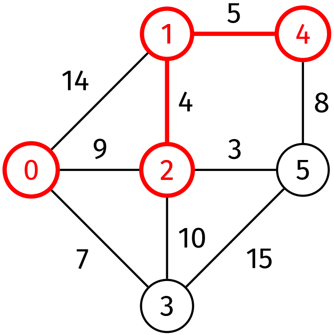
$0 \quad\quad 2 \longrightarrow 1 \longrightarrow 4$



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Example: Shortest path from 0 to 4

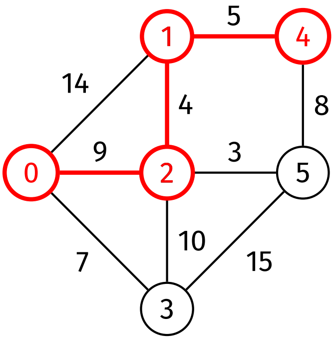
$0 \qquad 2 \longrightarrow 1 \longrightarrow 4$



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Example: Shortest path from 0 to 4

$$0 \longrightarrow 2 \longrightarrow 1 \longrightarrow 4$$



	[0]	[1]	[2]	[3]	[4]	[5]
pred	-1	2	0	0	1	2

Path Finding

How to find shortest path between two other vertices (neither of which are the source vertex)?

Generally, you will need to rerun Dijkstra's algorithm from one of these vertices.

Vertex Set

The vSet can be implemented in different ways:

1. Visited array
2. Explicit array/list of vertices
3. Priority queue

Vertex Set

Visited array implementation

Visited array implementation:

- Similar to visited array in BFS/DFS
- Array of V booleans, initialised to false
- After exploring vertex v, set visited[v] to true
- At the start of each iteration, find vertex v such that visited[v] is false and dist[v] is minimal $\Rightarrow O(V)$

Vertex Set

Array/list of vertices implementation

Array/list of vertices implementation:

- Store all vertices in an array/linked list
- After exploring vertex v, remove v from array/linked list
- At the start of each iteration, find vertex in array/list such that dist[v] is minimal $\Rightarrow O(V)$

Vertex Set

Priority queue implementation

Priority queue implementation:

- A priority queue is an ADT...
  - where each item has a priority
  - with two main operations:
    - **Insert**: insert item with priority
    - **Delete**: remove item with highest priority
- Use priority queue to store vertices, use *distance* to vertex as priority (smaller distance = higher priority)
- A good priority queue implementation has $O(\log n)$ insert and delete

Priority queues will be discussed in Week 9.

Proof by induction.

Aim is to prove that before and after each iteration:

1. For all explored nodes s, dist[s] is shortest distance from source to s
2. For all unexplored nodes t, dist[t] is shortest distance from source to t *via explored nodes only*

Ultimately, all nodes are explored, so by 1:

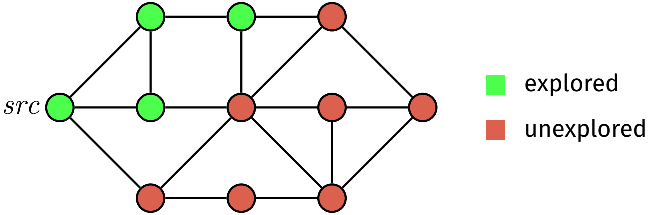
- For all nodes v, dist[v] is the shortest distance from source to v

Base case:

- Start of first iteration
  - 1 holds, as there are no explored nodes
  - 2 holds, because
    - dist[source] = 0
    - For all other nodes t, dist[t] = ∞

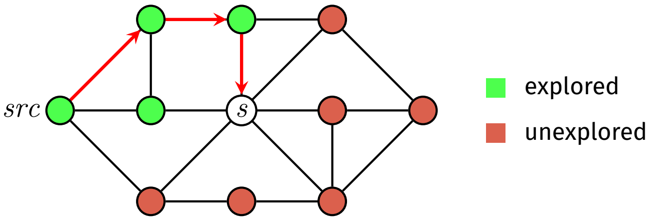
Induction step:

- Assume that 1 and 2 hold at the start of an iteration



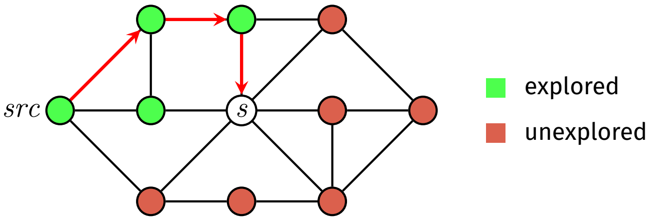
Induction step:

- Assume that 1 and 2 hold at the start of an iteration
- Let s be an unexplored node with minimum distance



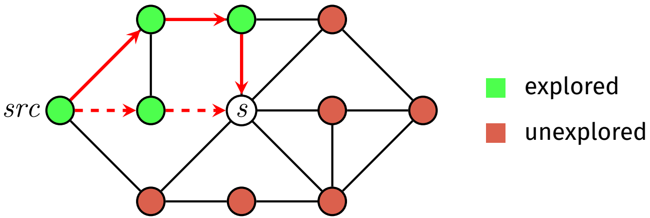
Induction step:

- Assume that 1 and 2 hold at the start of an iteration
- Let s be an unexplored node with minimum distance
- We claim that dist[s] is the shortest distance from source to s



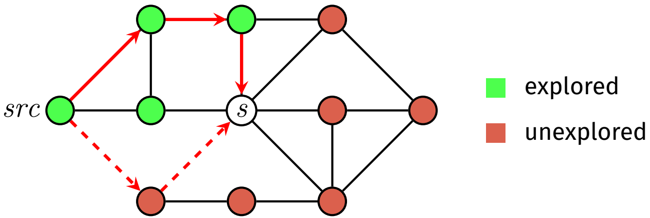
Induction step:

- Assume that 1 and 2 hold at the start of an iteration
- Let s be an unexplored node with minimum distance
- We claim that dist[s] is the shortest distance from source to s
  - If there is a shorter path to s via explored nodes only, then dist[s] would have been updated when exploring the predecessor of s on that path



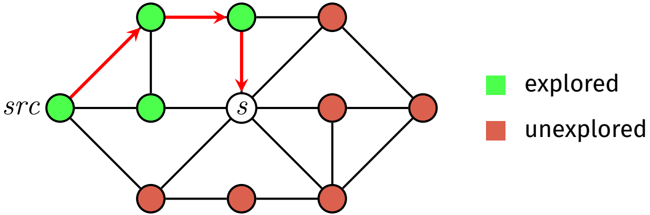
Induction step:

- Assume that 1 and 2 hold at the start of an iteration
- Let s be an unexplored node with minimum distance
- We claim that dist[s] is the shortest distance from source to s
  - If there is a shorter path to s via explored nodes only, then dist[s] would have been updated when exploring the predecessor of s on that path
  - If there is a shorter path to s via an unexplored node u, then dist[u] < dist[s], which is a contradiction, since s has minimum distance out of all unexplored nodes



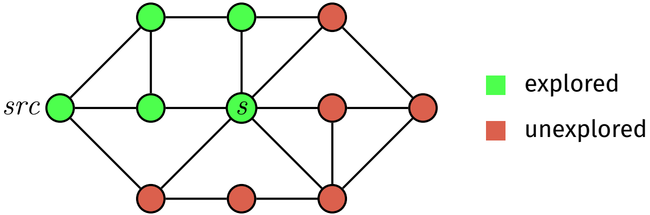
Induction step (continued):

- dist[s] is the shortest distance from source to s



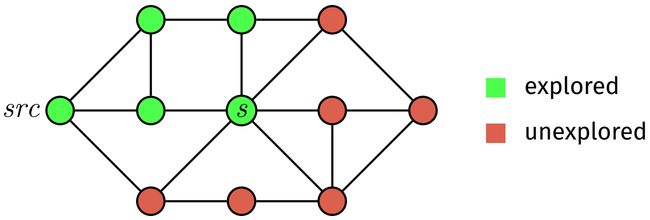
Induction step (continued):

- dist[s] is the shortest distance from source to s
- After exploring s:



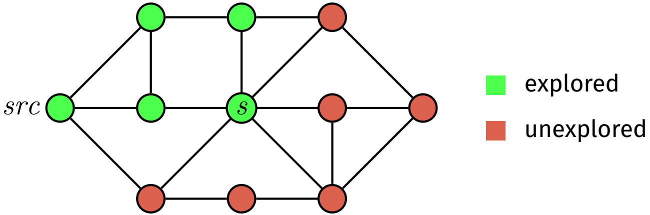
Induction step (continued):

- dist[s] is the shortest distance from source to s
- After exploring s:
  - 1 still holds for s, since dist[s] is not updated while exploring s
    - Same for all other explored nodes



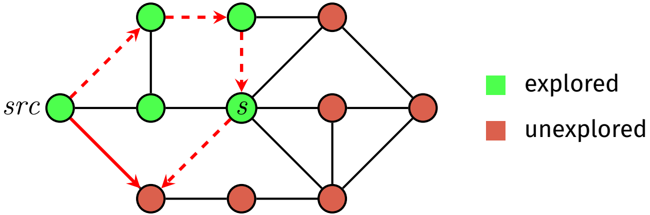
Induction step (continued):

- dist[s] is the shortest distance from source to s
- After exploring s:
  - 1 still holds for s, since dist[s] is not updated while exploring s
    - Same for all other explored nodes
  - 2 still holds for all unexplored nodes t, since:



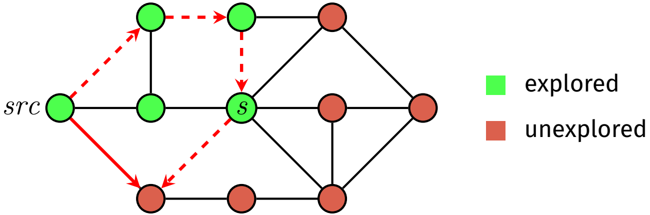
Induction step (continued):

- dist[s] is the shortest distance from source to s
- After exploring s:
  - 1 still holds for s, since dist[s] is not updated while exploring s
    - Same for all other explored nodes
  - 2 still holds for all unexplored nodes t, since:
    - If there is a shorter path to t via s then we would have updated dist[t] while exploring s



Induction step (continued):

- dist[s] is the shortest distance from source to s
- After exploring s:
  - 1 still holds for s, since dist[s] is not updated while exploring s
    - Same for all other explored nodes
  - 2 still holds for all unexplored nodes t, since:
    - If there is a shorter path to t via s then we would have updated dist[t] while exploring s
    - Otherwise, we would not have updated dist[t] and it would remain as it is



Analysis:

- Each edge is considered once $\Rightarrow O(E)$
  - Undirected edges are considered once in each direction
- Outer loop has V iterations
- Every iteration, algorithm must find vertex v in vSet with minimum distance - time complexity depends on vSet implementation
  - Boolean array $\Rightarrow O(V)$ per iteration
    $\Rightarrow$ overall cost $= O(E + V^2) = O(V^2)$
  - Array/list of vertices $\Rightarrow O(V)$ per iteration
    $\Rightarrow$ overall cost $= O(E + V^2) = O(V^2)$
  - Priority queue $\Rightarrow O(\log V)$ per iteration
    $\Rightarrow$ overall cost $= O(E + V \log V)$

Other Shortest Path Algorithms

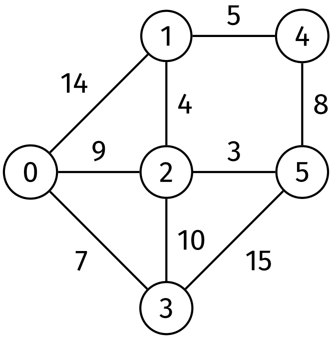
- Floyd-Warshall Algorithm
  - All-pairs shortest path
  - Works for graphs with negative weights
- Bellman-Ford Algorithm
  - Single-source shortest path
  - Works for graphs with negative weights
  - Can detect negative cycles

Feedback

https://forms.office.com/r/5c0fb4tvMb

Appendix

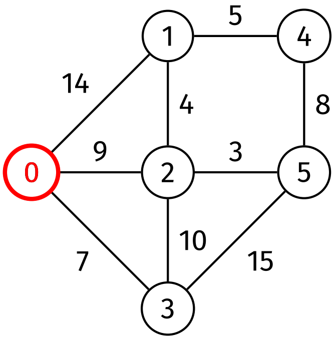
Initialisation




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

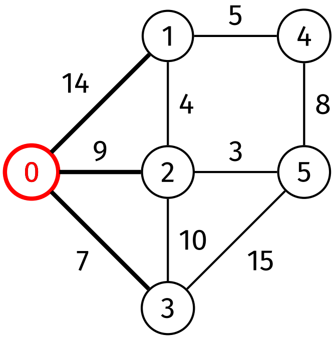
Remove 0 from vSet




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

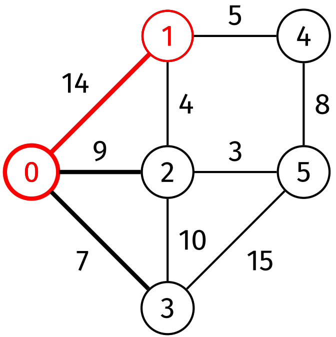
Explore 0




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

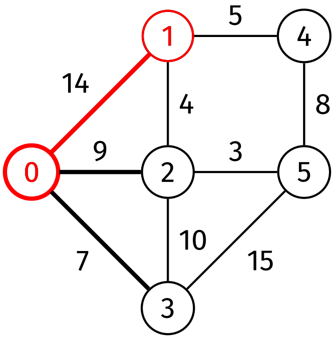
Relax along (0, 1, 14)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

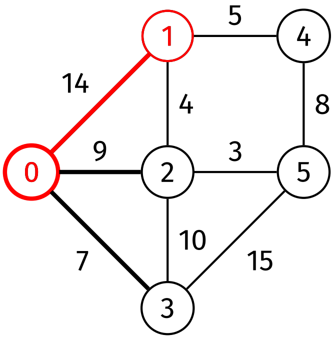
Relax along (0, 1, 14)
dist[0] + 14




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

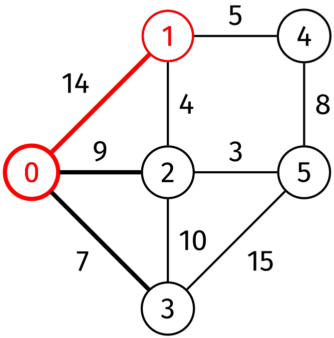
Relax along (0, 1, 14)
dist[0] + 14 = 14




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

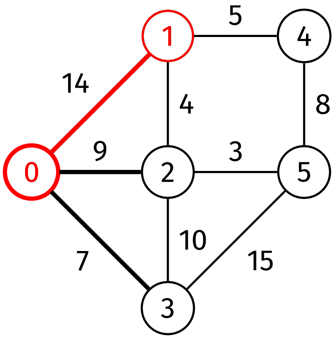
Relax along (0, 1, 14)
dist[0] + 14 = 14 < dist[1]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	∞	∞	∞	∞	∞
pred	-1	-1	-1	-1	-1	-1

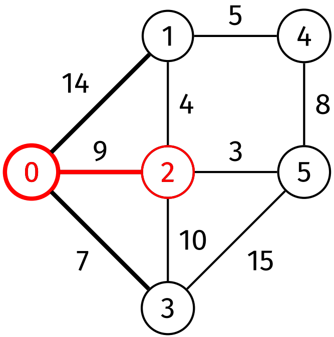
Relax along (0, 1, 14)
dist[0] + 14 = 14 < dist[1]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	∞	∞	∞	∞
pred	-1	0	-1	-1	-1	-1

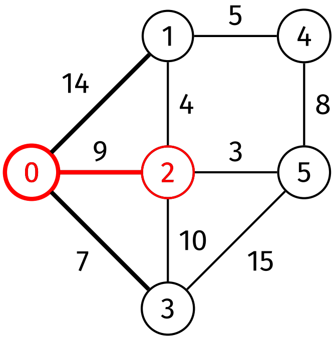
Relax along (0, 2, 9)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	∞	∞	∞	∞
pred	-1	0	-1	-1	-1	-1

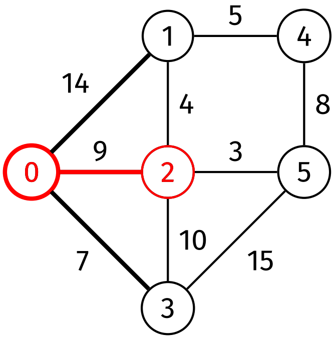
Relax along (0, 2, 9)
dist[0] + 9




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	∞	∞	∞	∞
pred	-1	0	-1	-1	-1	-1

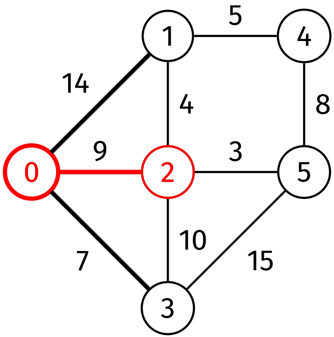
Relax along (0, 2, 9)
dist[0] + 9 = 9




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	∞	∞	∞	∞
pred	-1	0	-1	-1	-1	-1

Relax along (0, 2, 9)
dist[0] + 9 = 9 < dist[2]

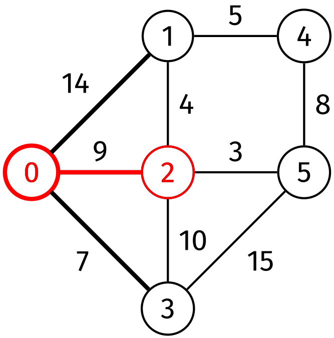



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	∞	∞	∞	∞
pred	-1	0	-1	-1	-1	-1

Dijkstra's Algorithm
Example

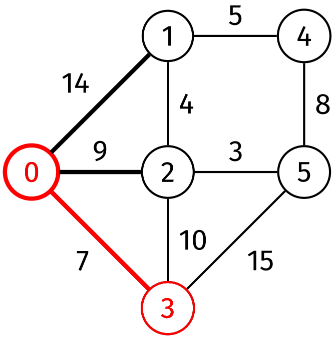
Relax along (0, 2, 9)
dist[0] + 9 = 9 < dist[2]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	∞	∞	∞
pred	-1	0	0	-1	-1	-1

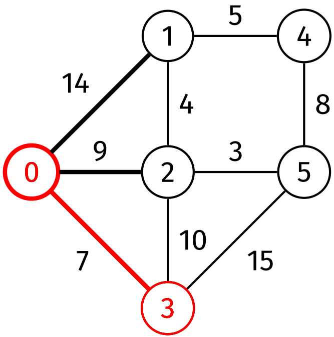
Relax along (0, 3, 7)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	∞	∞	∞
pred	-1	0	0	-1	-1	-1

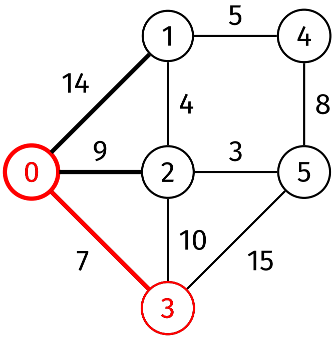
Relax along (0, 3, 7)
dist[0] + 7




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	∞	∞	∞
pred	-1	0	0	-1	-1	-1

Relax along (0, 3, 7)
dist[0] + 7 = 7



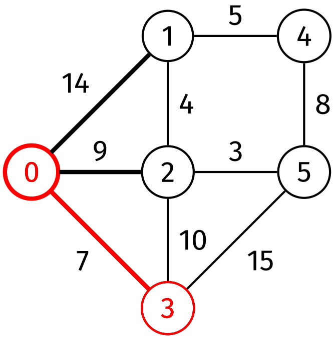

```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	∞	∞	∞
pred	-1	0	0	-1	-1	-1

Dijkstra's Algorithm

Example

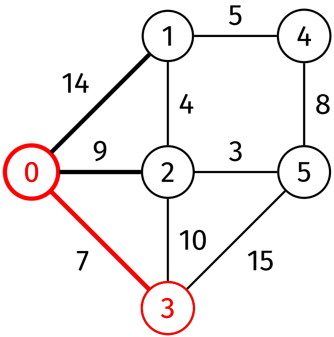
Relax along (0, 3, 7)
dist[0] + 7 = 7 < dist[3]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	∞	∞	∞
pred	-1	0	0	-1	-1	-1

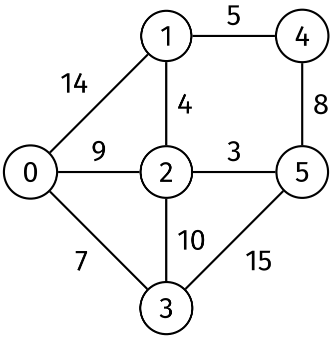
Relax along (0, 3, 7)
dist[0] + 7 = 7 < dist[3]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Done with exploring 0



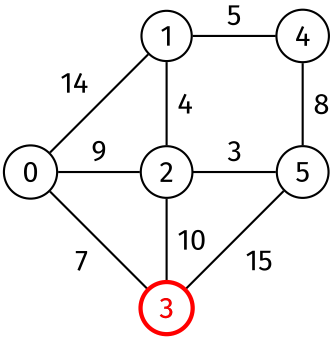

```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Dijkstra's Algorithm

Example

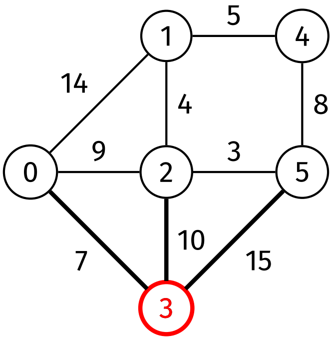
Remove 3 from vSet




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

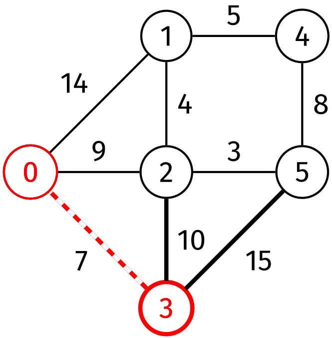
Explore 3




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

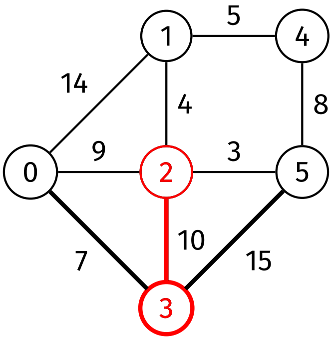
No need to consider (3, 0, 7)
(0 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

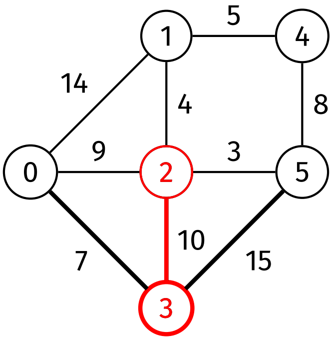
Relax along (3, 2, 10)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

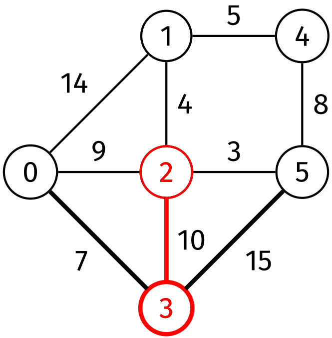
Relax along (3, 2, 10)
dist[3] + 10




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

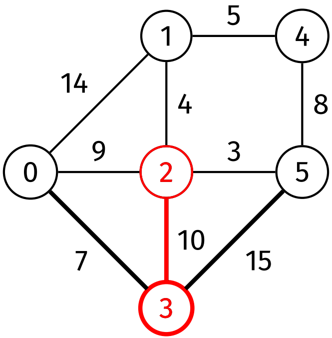
Relax along (3, 2, 10)
dist[3] + 10 = 17




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Relax along (3, 2, 10)
dist[3] + 10 = 17 $\not<$ dist[2]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

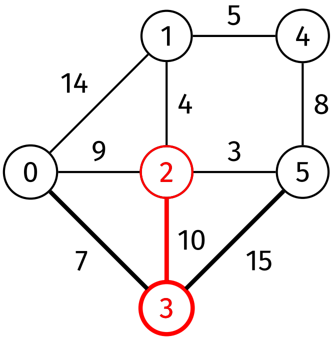
	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Dijkstra's Algorithm

Example

Relax along (3, 2, 10)

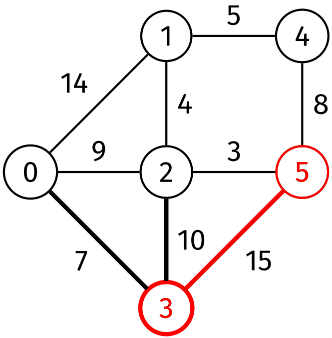
$\text{dist}[3] + 10 = 17 \not< \text{dist}[2]$




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Relax along (3, 5, 15)

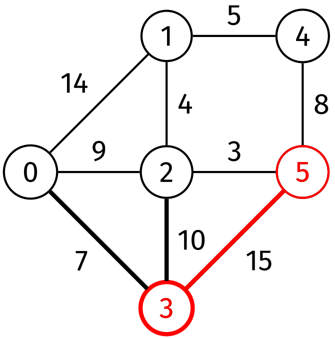



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

Dijkstra's Algorithm
Example

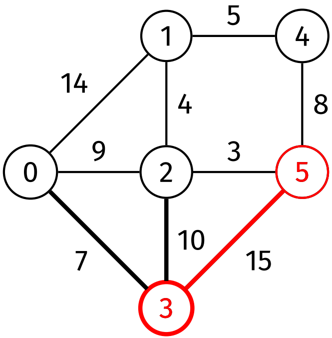
Relax along (3, 5, 15)
dist[3] + 15




```
while vSet is not empty:
    find vertex v in vSet such that
          dist[v] is minimal
          and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

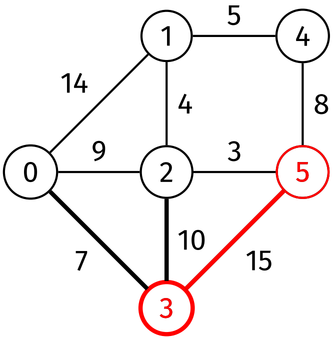
Relax along (3, 5, 15)
dist[3] + 15 = 22




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

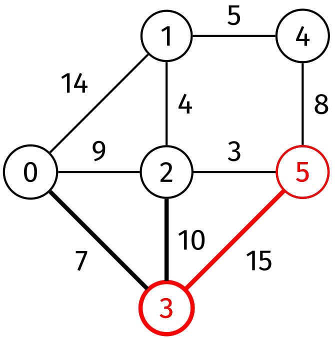
Relax along (3, 5, 15)
dist[3] + 15 = 22 < dist[5]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	∞
pred	-1	0	0	0	-1	-1

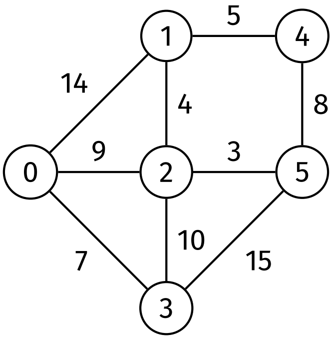
Relax along (3, 5, 15)
dist[3] + 15 = 22 < dist[5]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

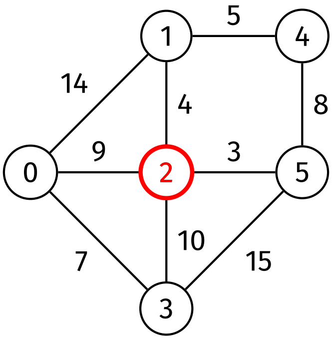
Done with exploring 3




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

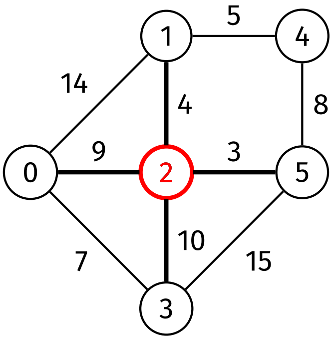
Remove 2 from vSet




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

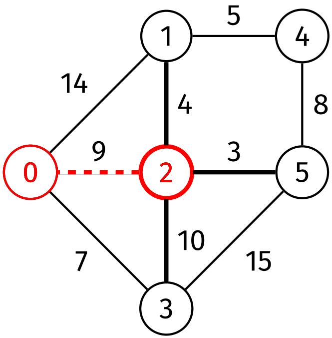
Explore 2




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

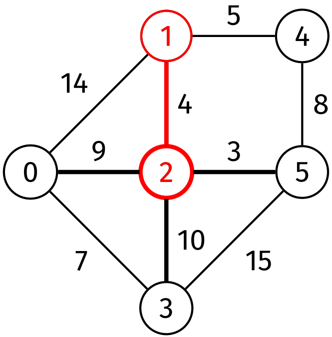
No need to consider (2, 0, 9)
(0 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

Relax along (2, 1, 4)

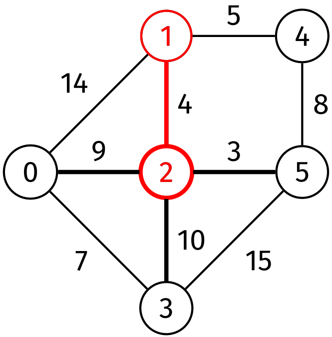



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

Relax along (2, 1, 4)

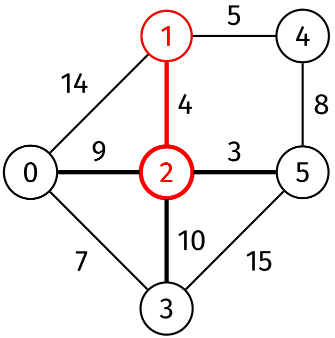
dist[2] + 4




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

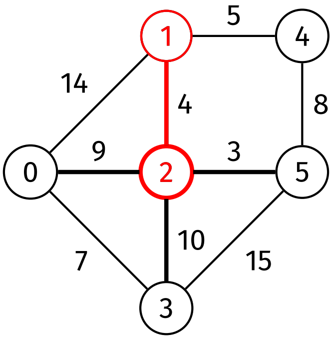
Relax along (2, 1, 4)
dist[2] + 4 = 13




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

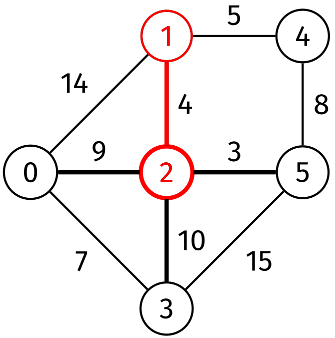
Relax along (2, 1, 4)
dist[2] + 4 = 13 < dist[1]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	14	9	7	∞	22
pred	-1	0	0	0	-1	3

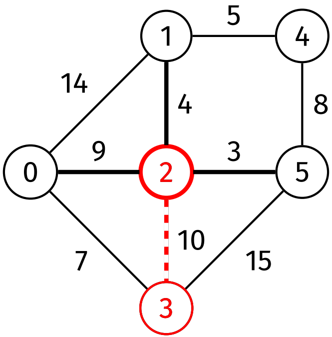
Relax along (2, 1, 4)
dist[2] + 4 = 13 < dist[1]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	22
pred	-1	2	0	0	-1	3

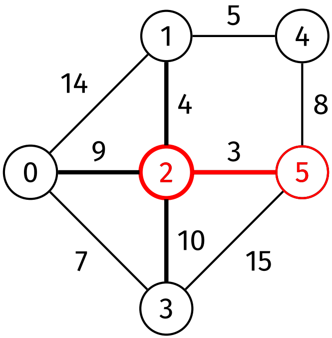
No need to consider (2, 3, 10)
(3 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	22
pred	-1	2	0	0	-1	3

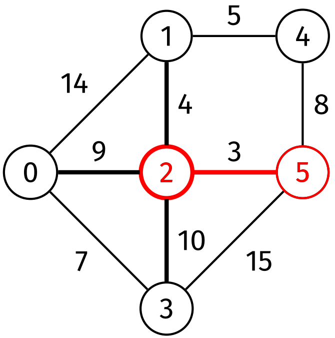
Relax along (2, 5, 3)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	22
pred	-1	2	0	0	-1	3

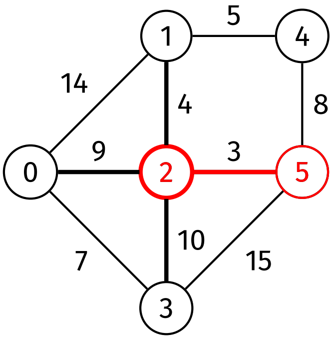
Relax along (2, 5, 3)
dist[2] + 3




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	22
pred	-1	2	0	0	-1	3

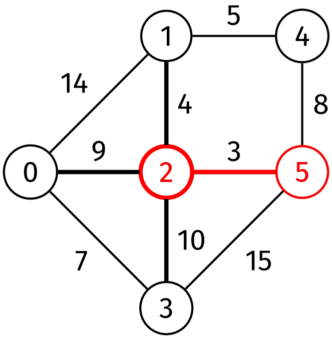
Relax along (2, 5, 3)
dist[2] + 3 = 12




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	22
pred	-1	2	0	0	-1	3

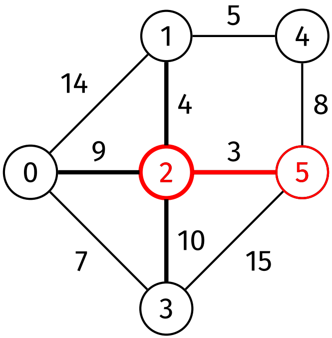
Relax along (2, 5, 3)
dist[2] + 3 = 12 < dist[5]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	22
pred	-1	2	0	0	-1	3

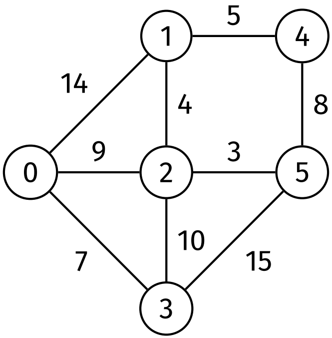
Relax along (2, 5, 3)
dist[2] + 3 = 12 < dist[5]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

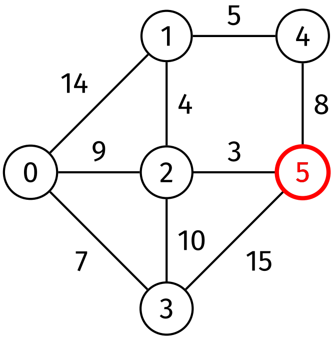
Done with exploring 2




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

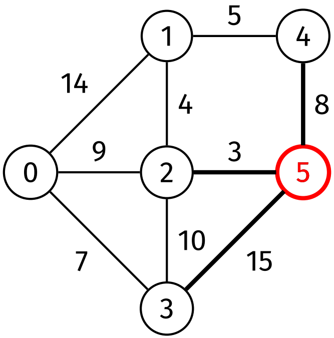
Remove 5 from vSet




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

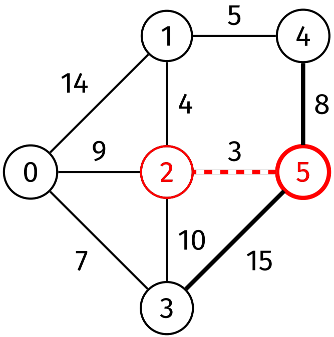
Explore 5




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

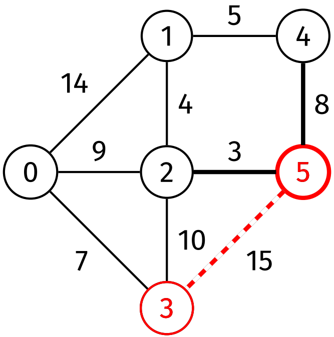
No need to consider (5, 2, 3)
(2 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

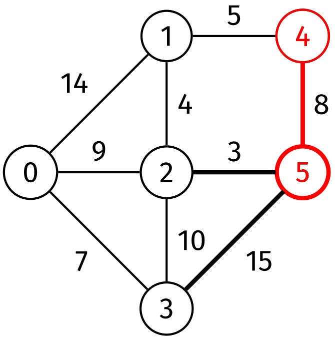
No need to consider (5, 3, 15)
(3 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

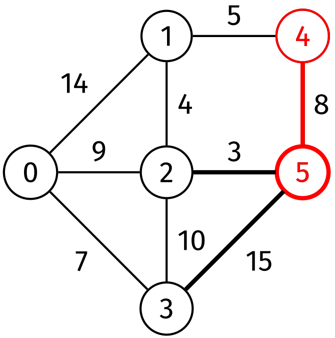
Relax along (5, 4, 8)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

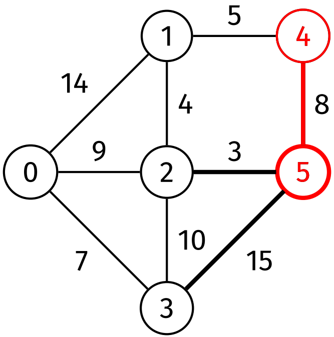
Relax along (5, 4, 8)
dist[5] + 8




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

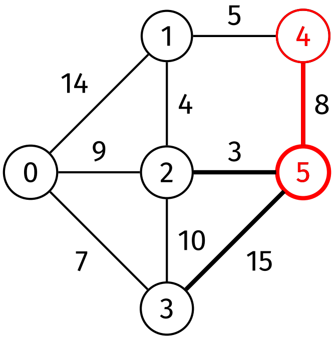
Relax along (5, 4, 8)
dist[5] + 8 = 20




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

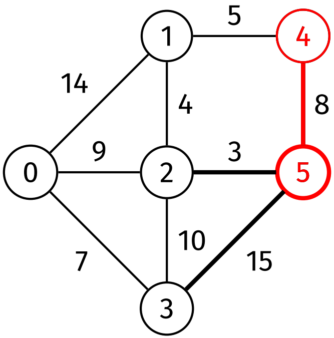
Relax along (5, 4, 8)
dist[5] + 8 = 20 < dist[4]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	∞	12
pred	-1	2	0	0	-1	2

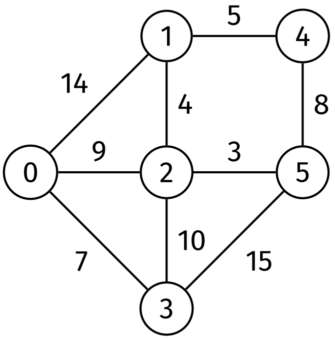
Relax along (5, 4, 8)
dist[5] + 8 = 20 < dist[4]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

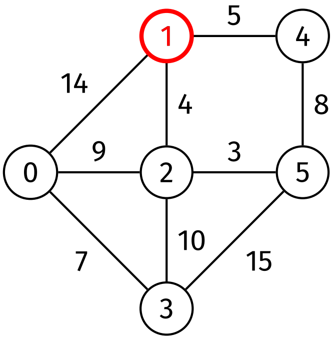
Done with exploring 5




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

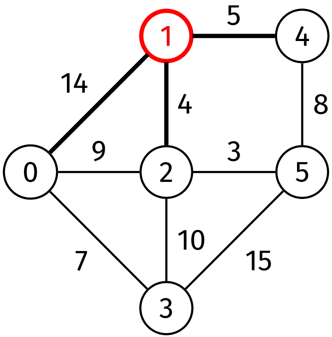
Remove 1 from vSet




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

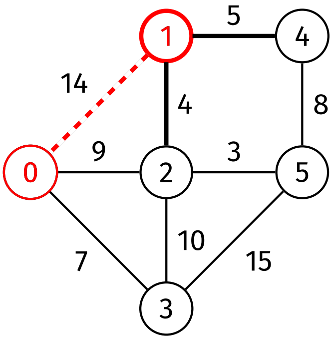
Explore 1




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

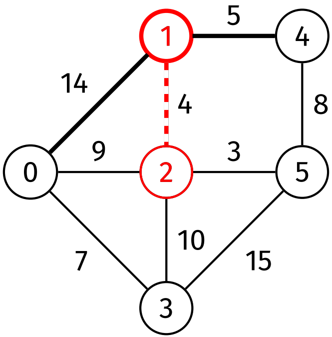
No need to consider (1, 0, 14)
(0 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

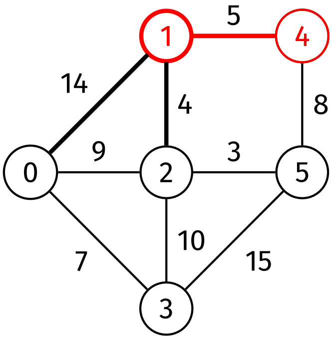
No need to consider (1, 2, 4)
(2 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

Relax along (1, 4, 5)

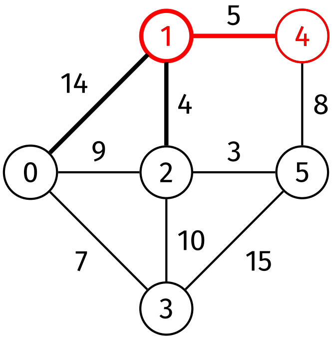



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

Relax along (1, 4, 5)

dist[1] + 5

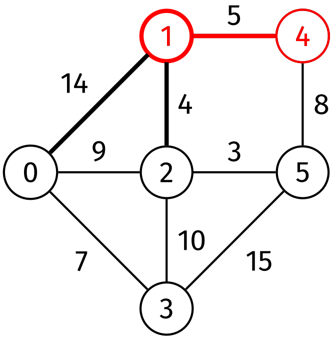



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

Relax along (1, 4, 5)

dist[1] + 5 = 18

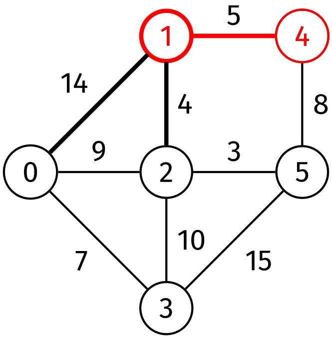



```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

Dijkstra's Algorithm
Example

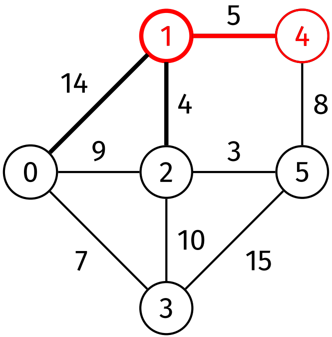
Relax along (1, 4, 5)
dist[1] + 5 = 18 < dist[4]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	20	12
pred	-1	2	0	0	5	2

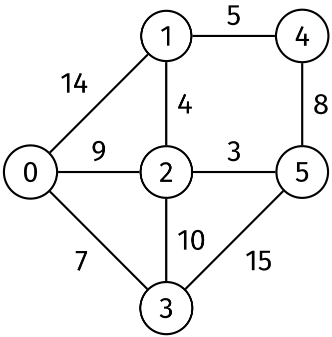
Relax along (1, 4, 5)
dist[1] + 5 = 18 < dist[4]




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

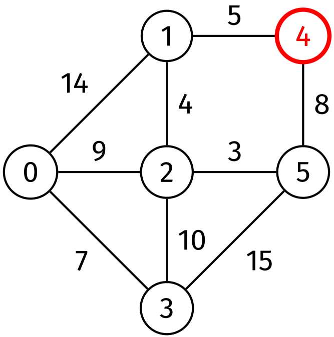
Done with exploring 1




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

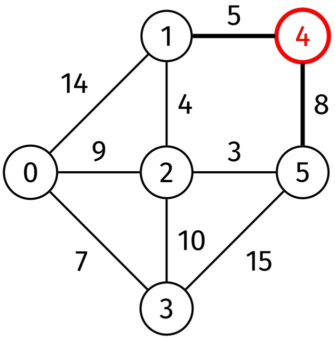
Remove 4 from vSet




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

Explore 4



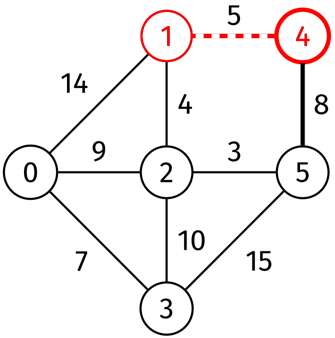

```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

Dijkstra's Algorithm

Example

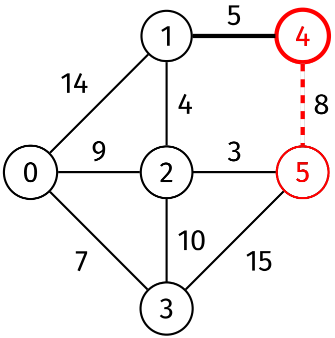
No need to consider (4, 1, 5)
(1 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

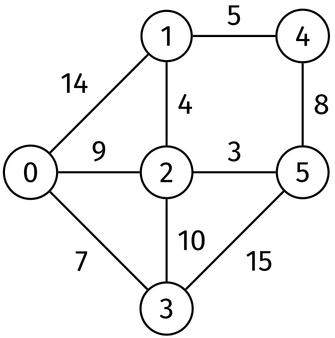
No need to consider (4, 5, 8)
(5 has already been explored)




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

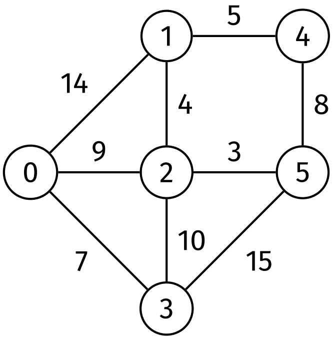
Done with exploring 4




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2

Finished




```
while vSet is not empty:
    find vertex v in vSet such that
            dist[v] is minimal
            and remove it from vSet

    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

	[0]	[1]	[2]	[3]	[4]	[5]
dist	0	13	9	7	18	12
pred	-1	2	0	0	1	2