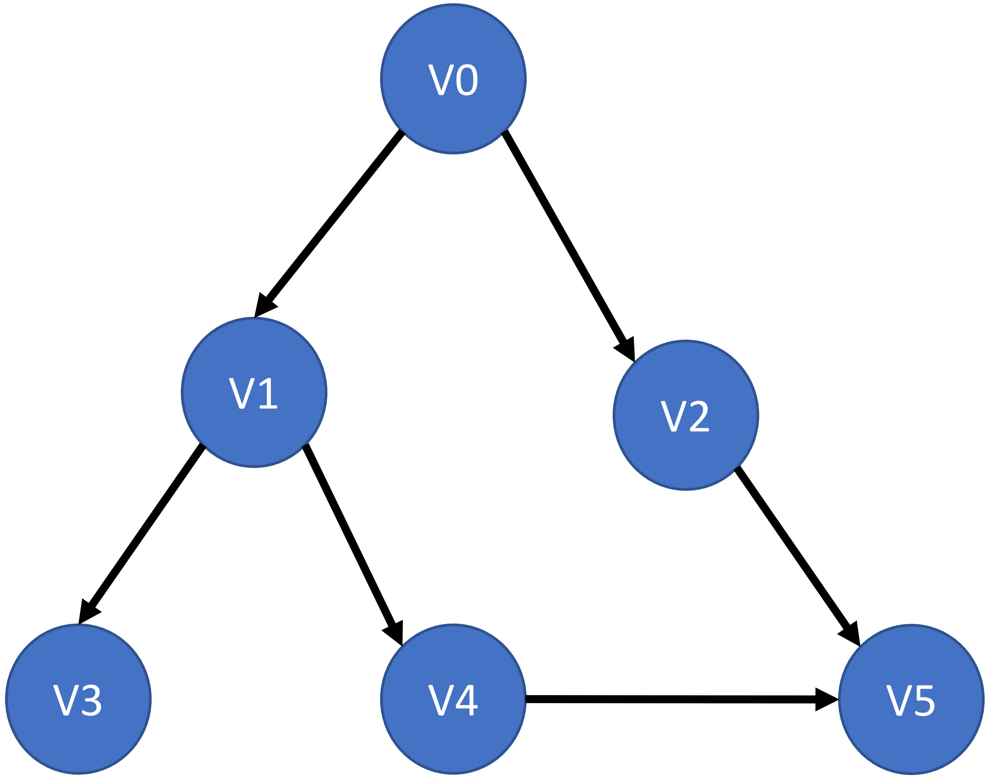
Path-finding

Daniel Nogueira

dnogueira@ipca.pt

Algorithms

Depth-First Search (DFS)

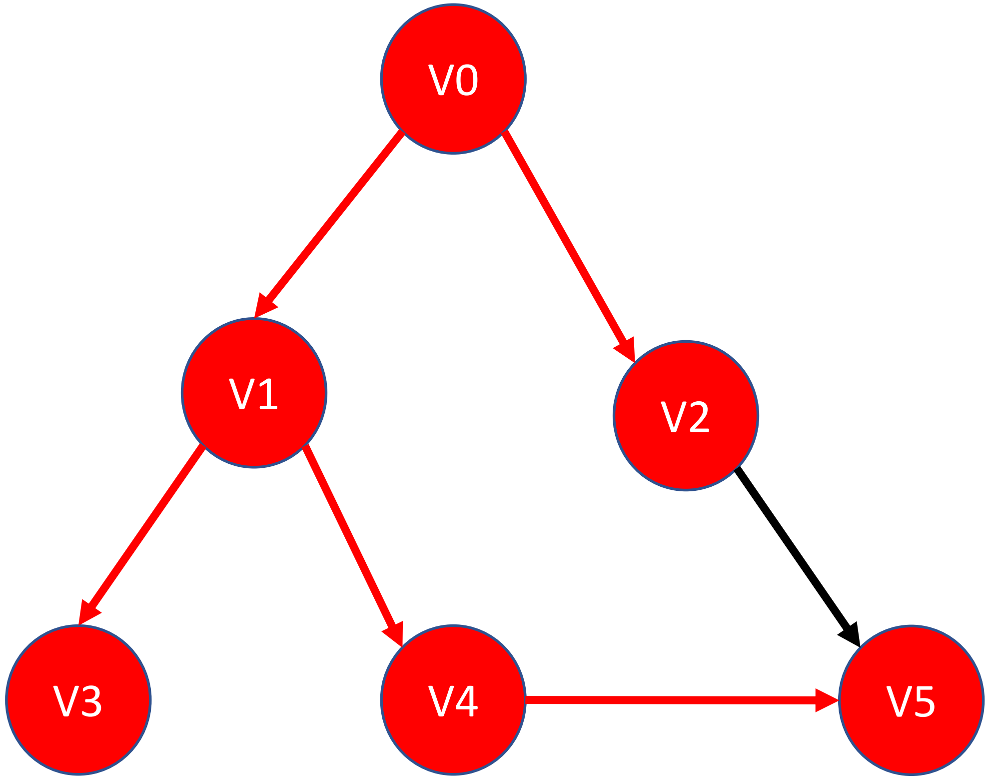


The Algorithm

1.Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
 - Choose an adjacent node not yet visited
 - Visit it
3. If it is a non-objective end node:
 - Return to this father
 - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

Depth-First Search (DFS)

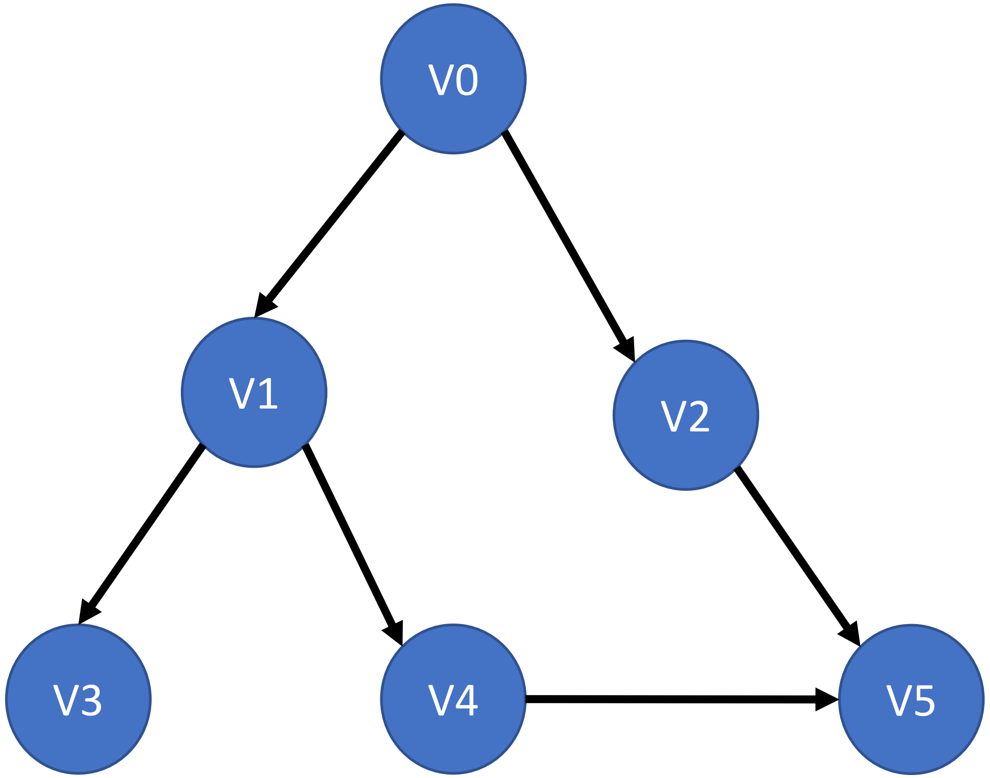


The Algorithm

1.Set a start node
2. While this is not an objective or final node (node whose adjacency has already been visited):
  - Choose an adjacent node not yet visited
  - Visit it
3. If it is a non-objective end node:
  - Return to this father
  - If there is a father, repeat. If there is no parent, choose another start node

Algorithms

Breadth-First Search (BFS)

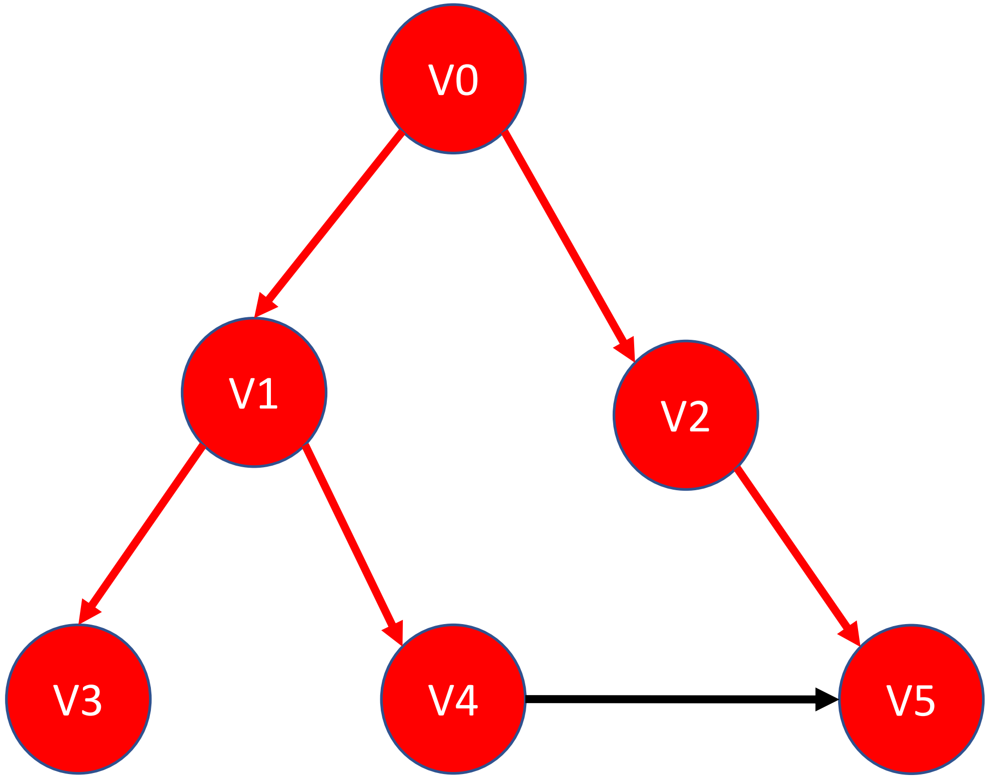


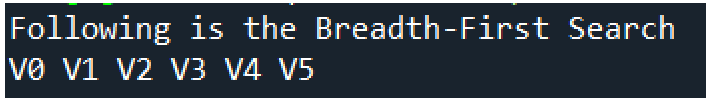
The Algorithm

1.Define an initial node, marking it as explored
2. Put it on the list
3. As long as the queue is not empty:
  - Remove the 1st node from the queue, u
  - For each neighbour v of u:
    * If v is not explored:
      ** Mark v as explored
      ** Put v at the end of the queue
4. Repeat from another starting node, if there is one

Algorithms

Breadth-First Search (BFS)



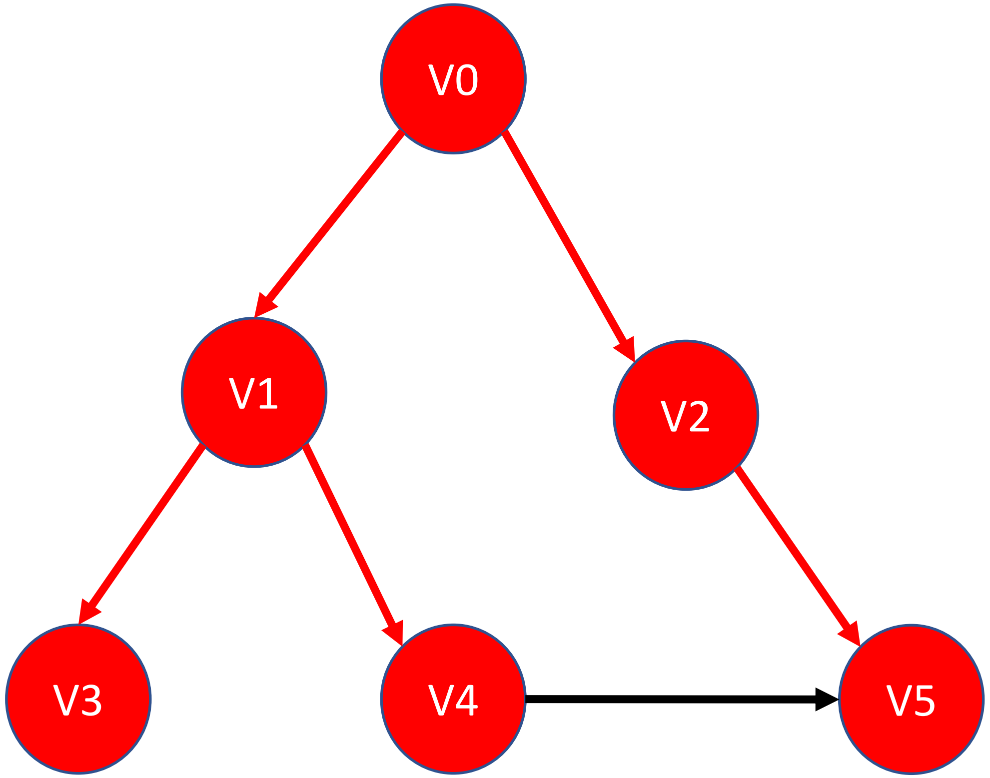


DFS V0 V1 V3 V4 V5 V2

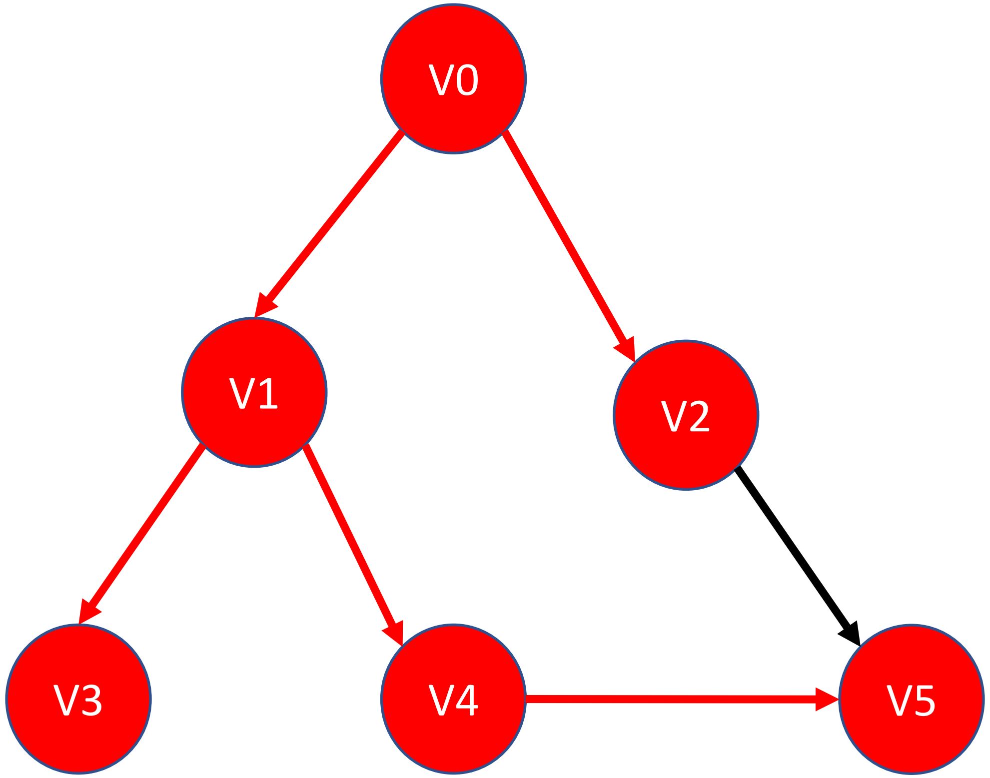
BFS V0 V1 V2 V3 V4 V5

Algorithms

Breadth-First Search (BFS)



Depth-First Search (DFS)



Algorithms

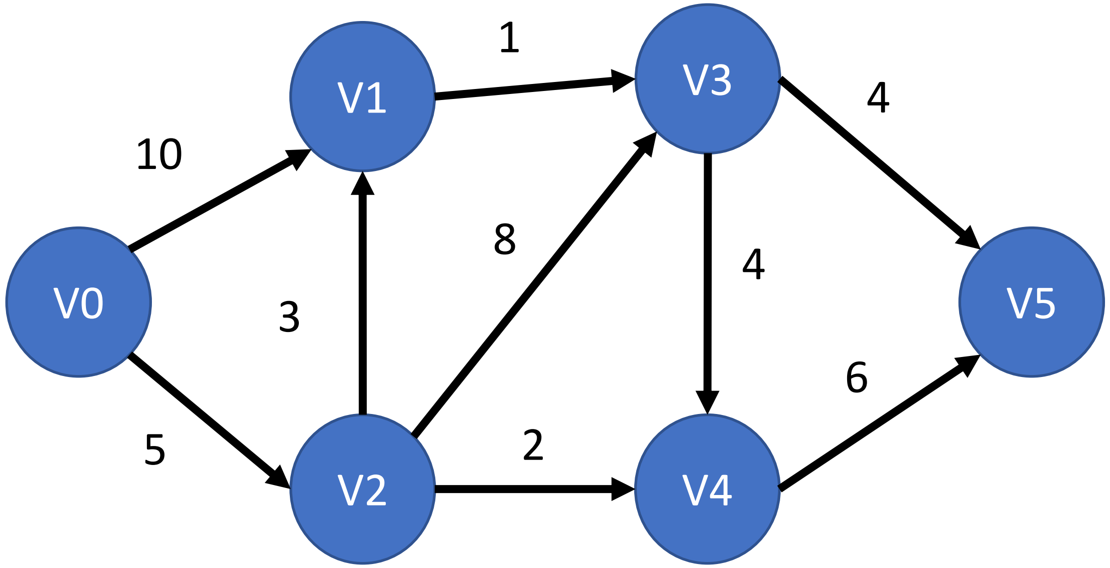
Dijkstra → Dijkstra's algorithm can be used to find the **shortest distance** from the source vertex to all other vertices in a weighted graph. For each vertex *v* of the graph, we maintain an attribute ***d(v)*** which is an upper bound on the weight of the shortest path from the initial **node *s* to *v*.**

The Algorithm

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all *v*
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose *u* whose estimate is the smallest among the open
   * Close *u*
   * For every open node *v* adjacent to *u*: relax edge (*u,v*)

Algorithms

Dijkstra

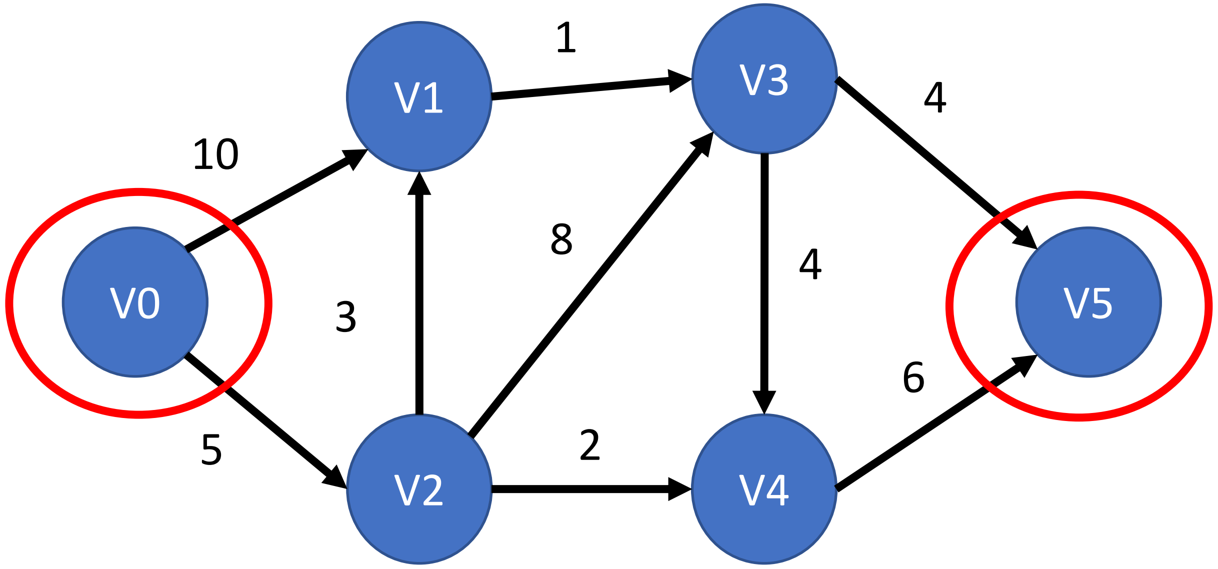


1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose <u>*u*</u> whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

Dijkstra

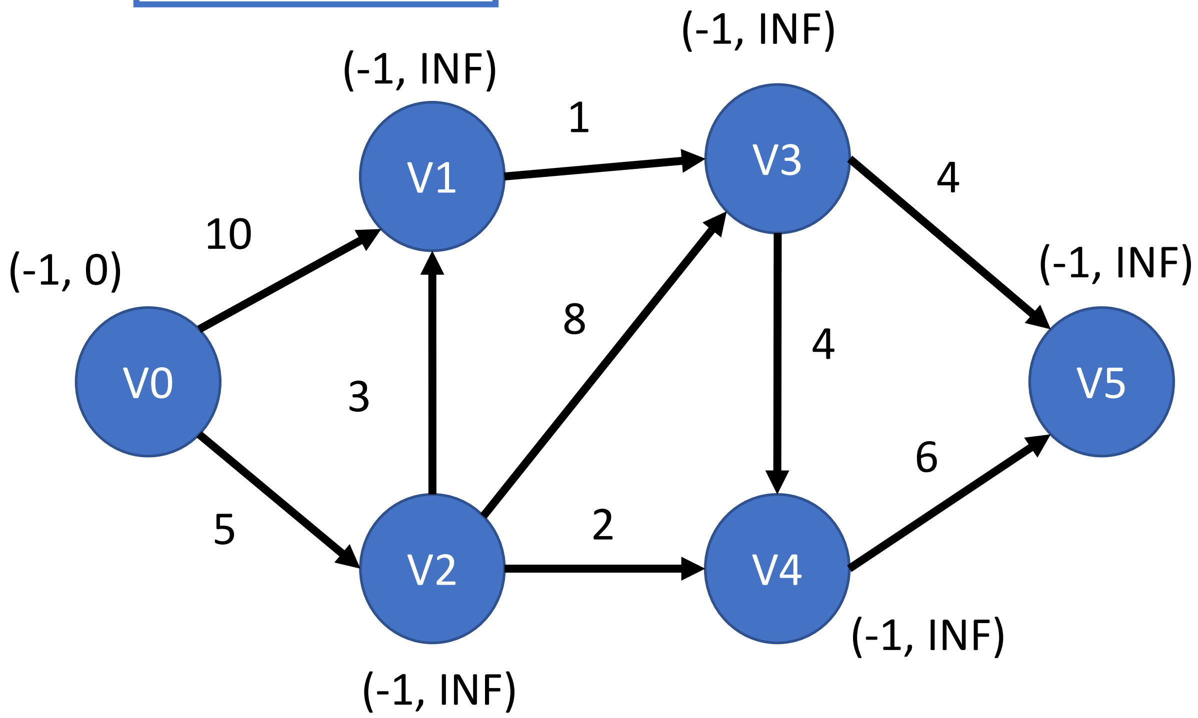
NOTATION → (father node, weigth)



1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose <u>*u*</u> whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

Dijkstra

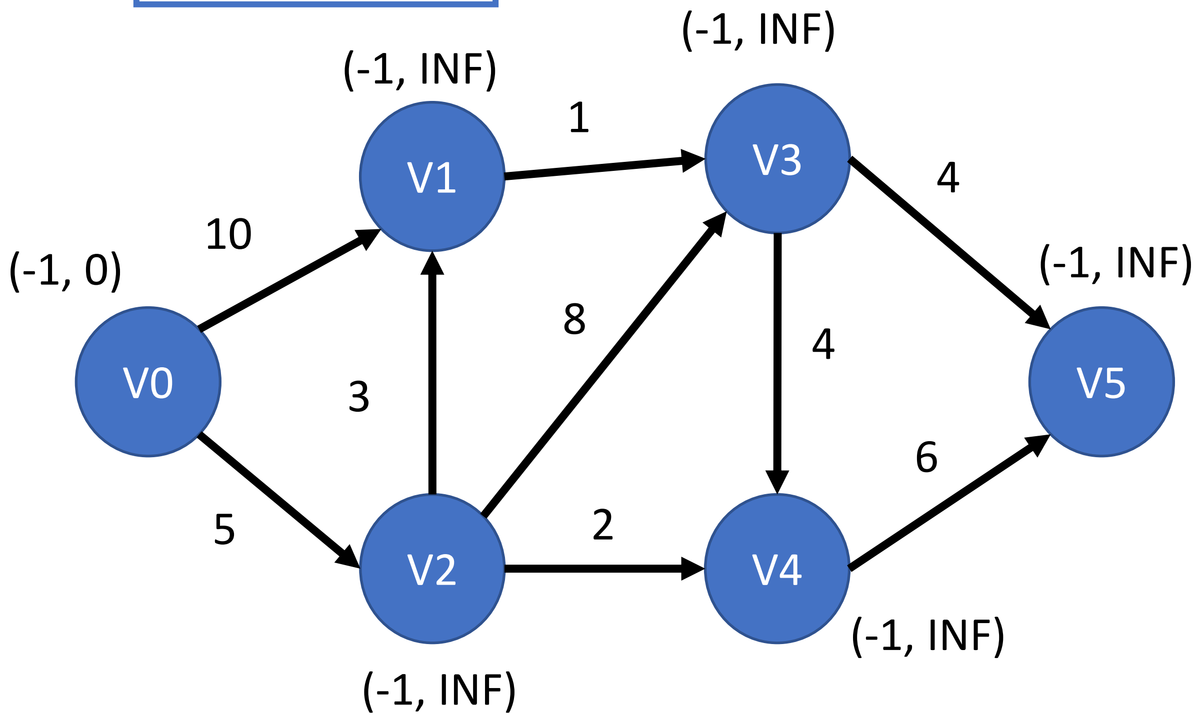


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose <u>*u*</u> whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

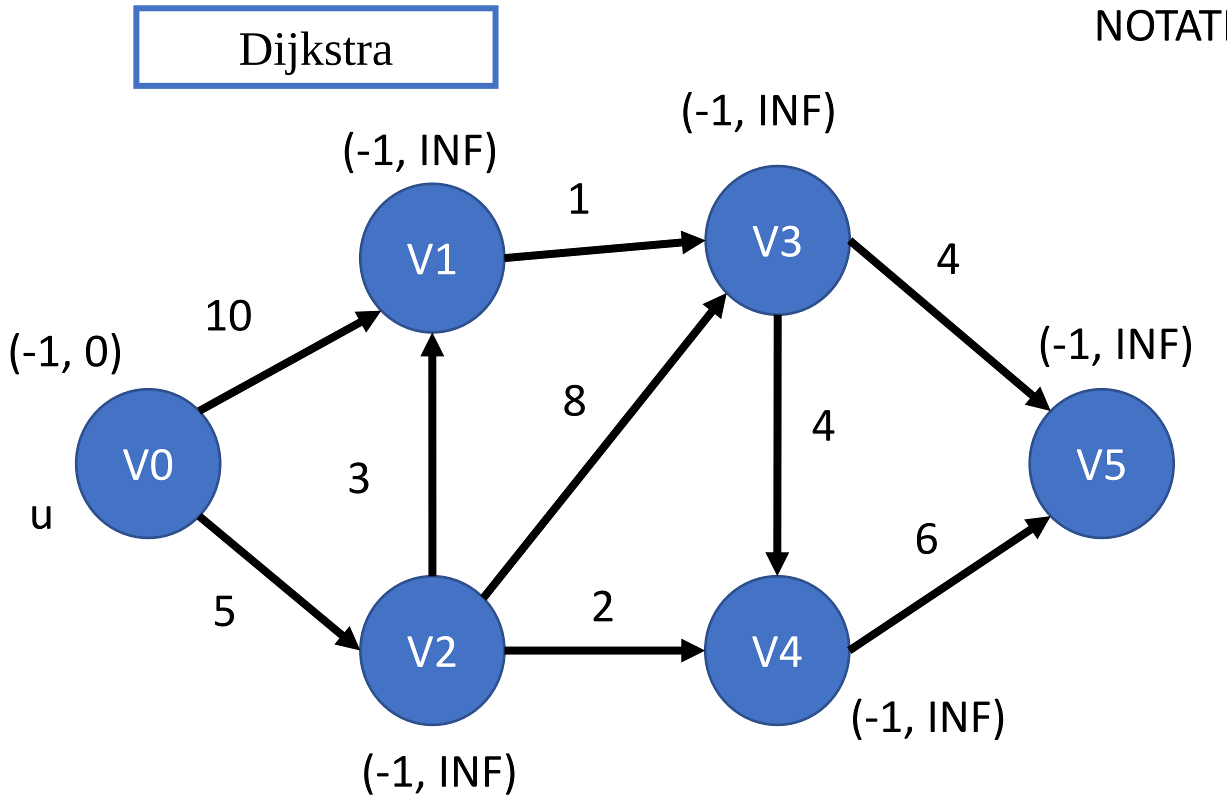
Dijkstra



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all v
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close u
   * For every open node v adjacent to u: relax edge (u,v)

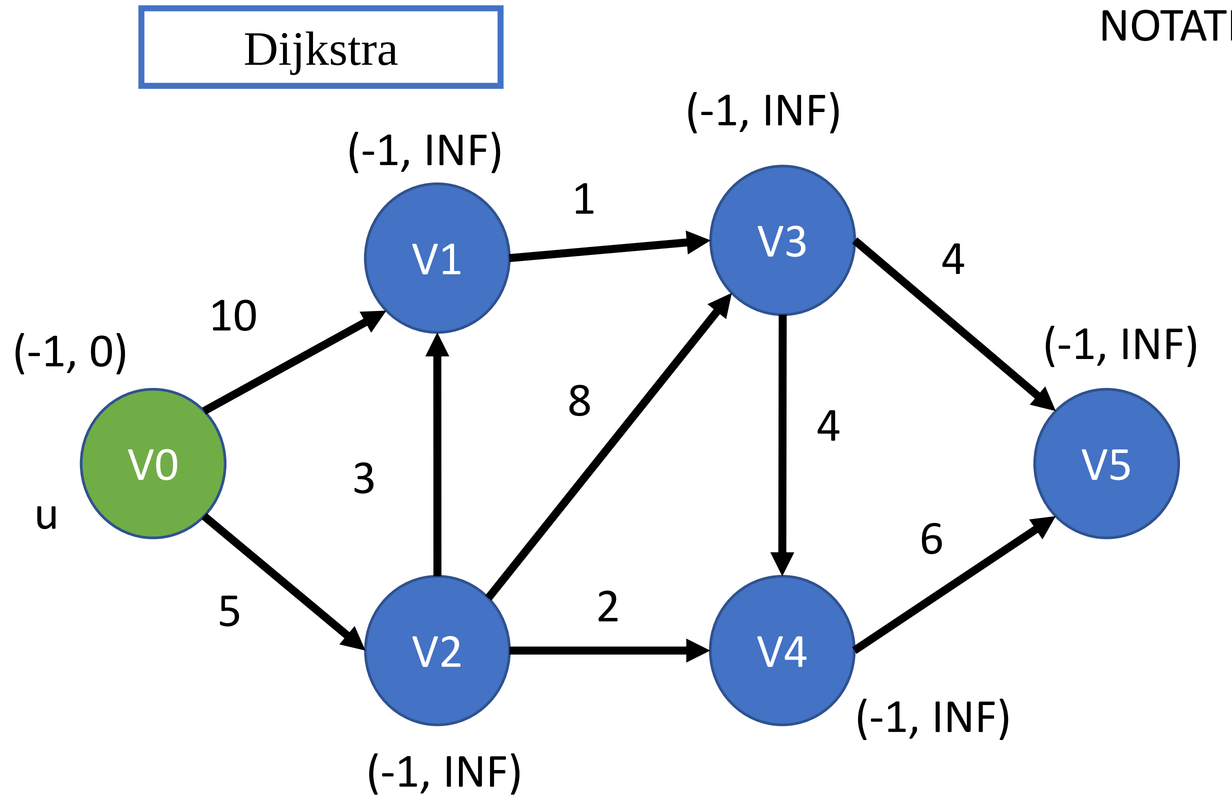
Algorithms



NOTATION ➡ (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

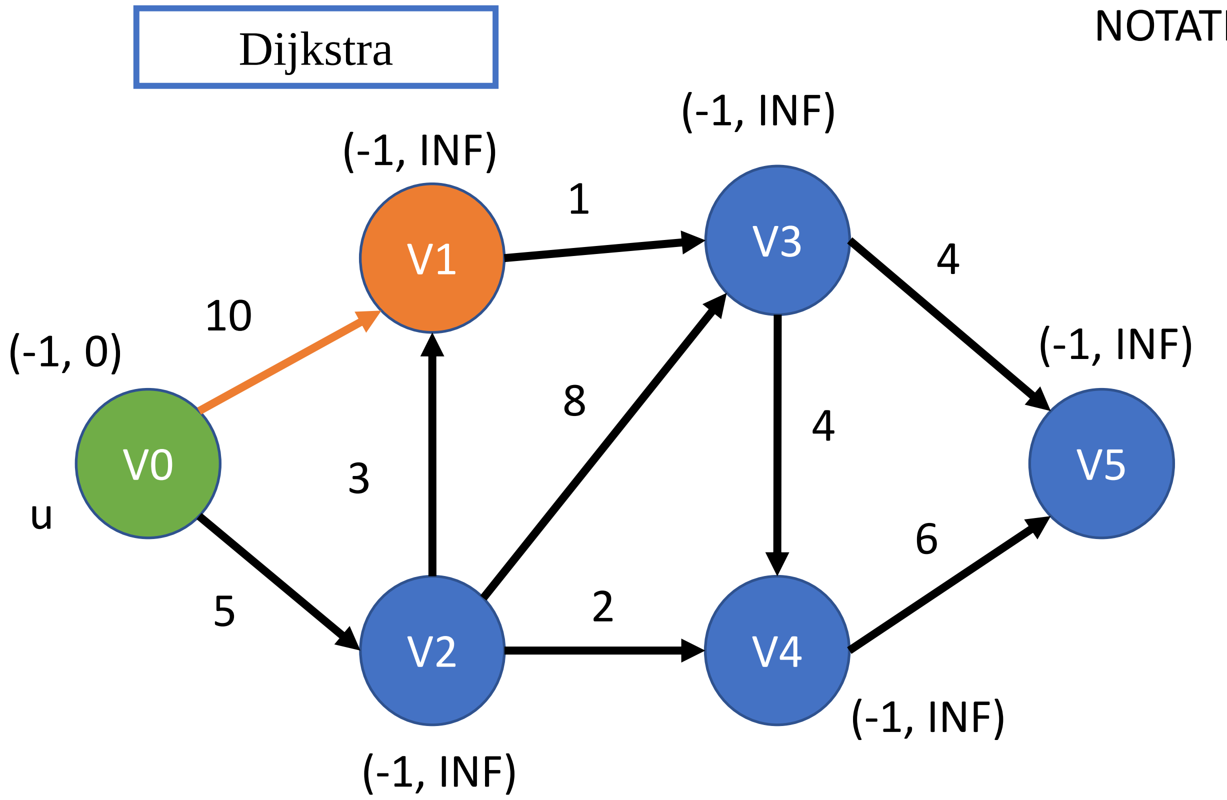
Algorithms



NOTATION ➡ (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

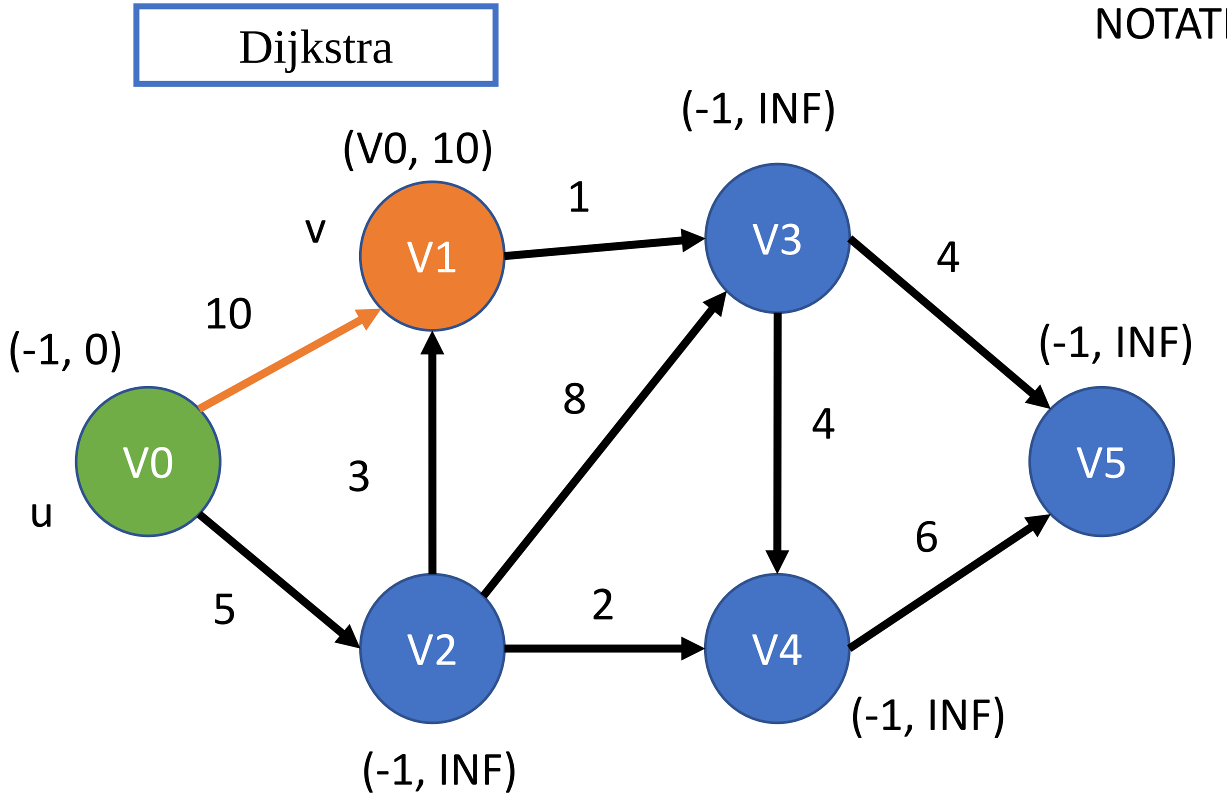
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

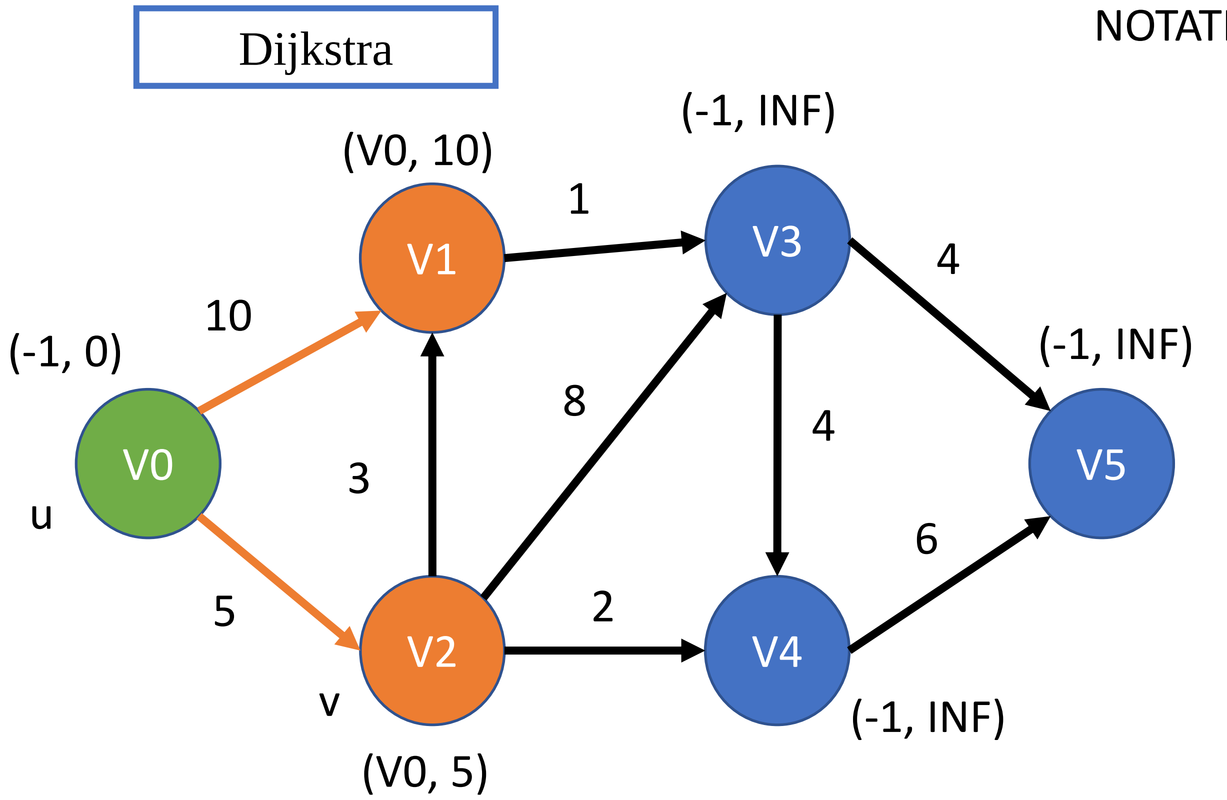
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

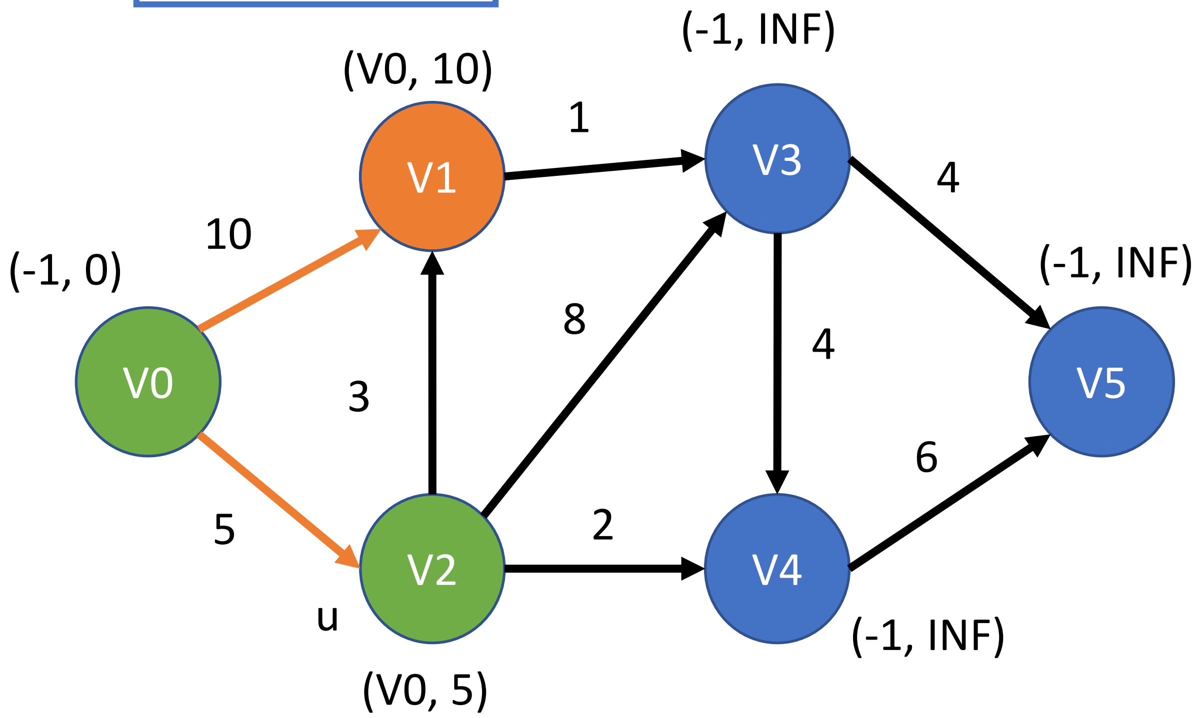


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all v
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close u
   * For every open node v adjacent to u: relax edge (u,v)

Algorithms

Dijkstra

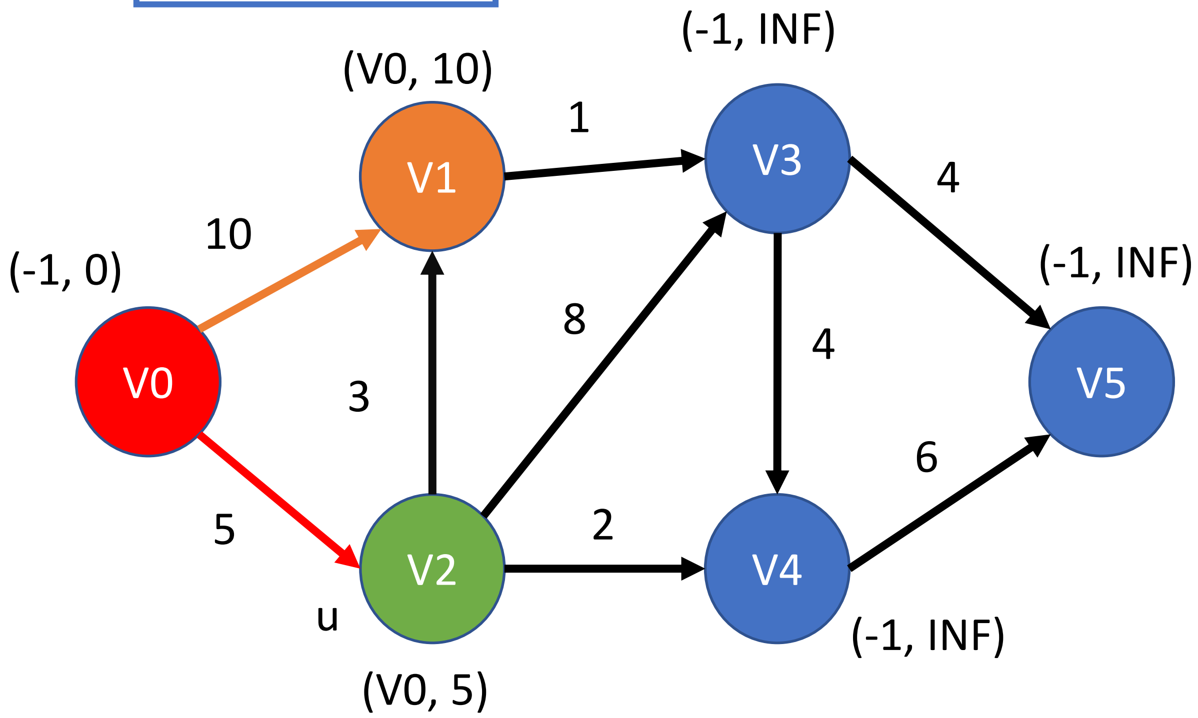


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

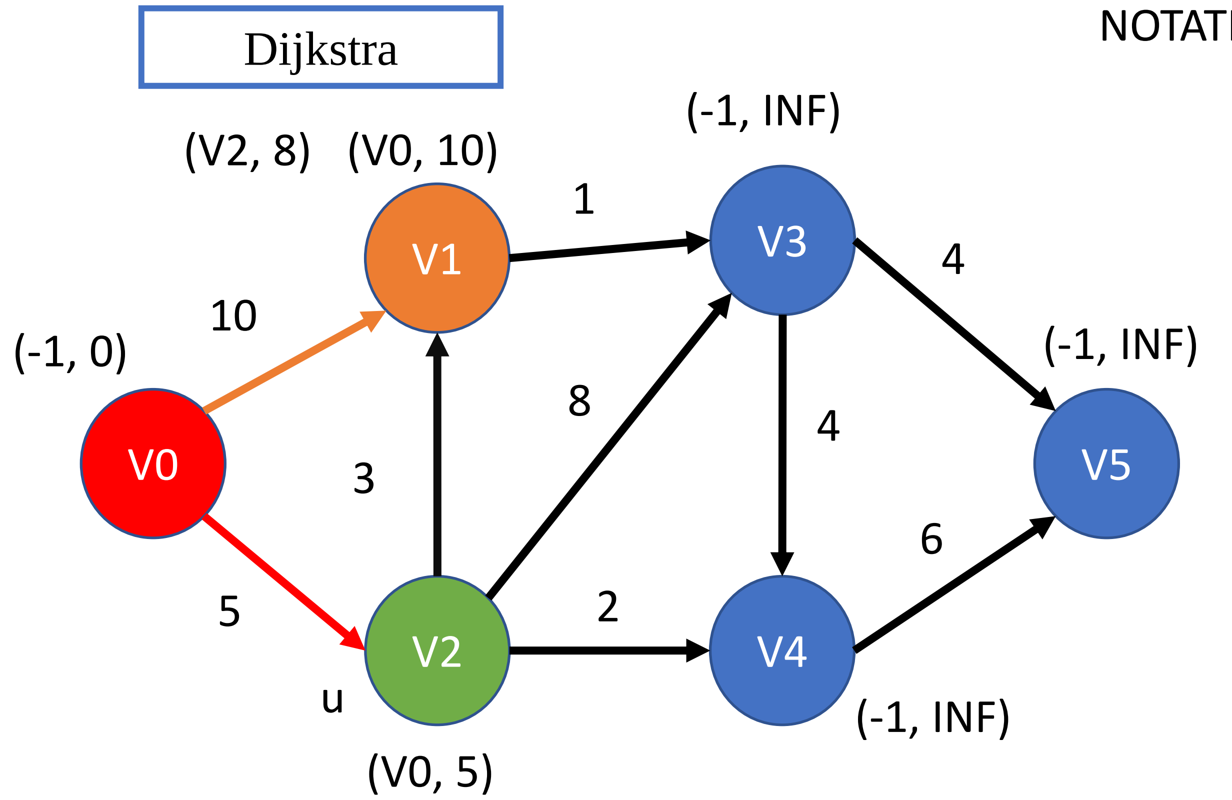
Dijkstra



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

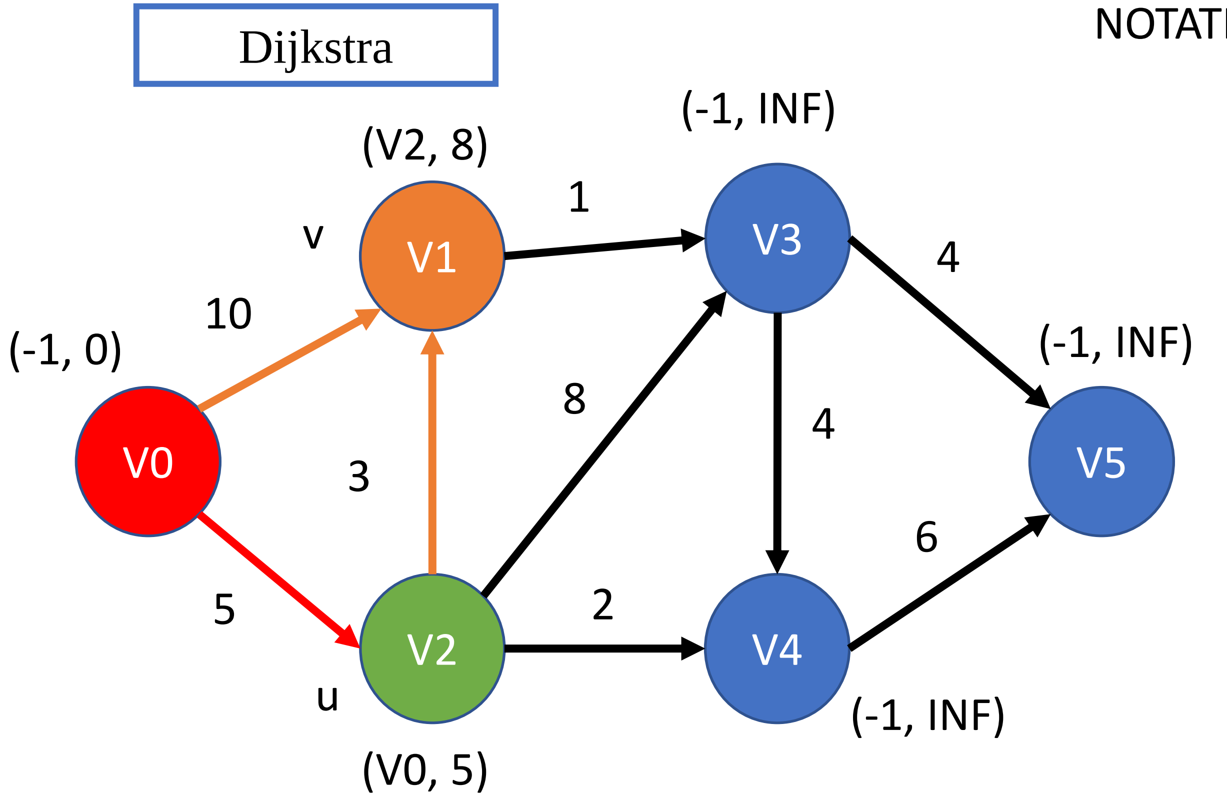
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all v
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close u
   * For every open node v adjacent to u: relax edge (u,v)

Algorithms

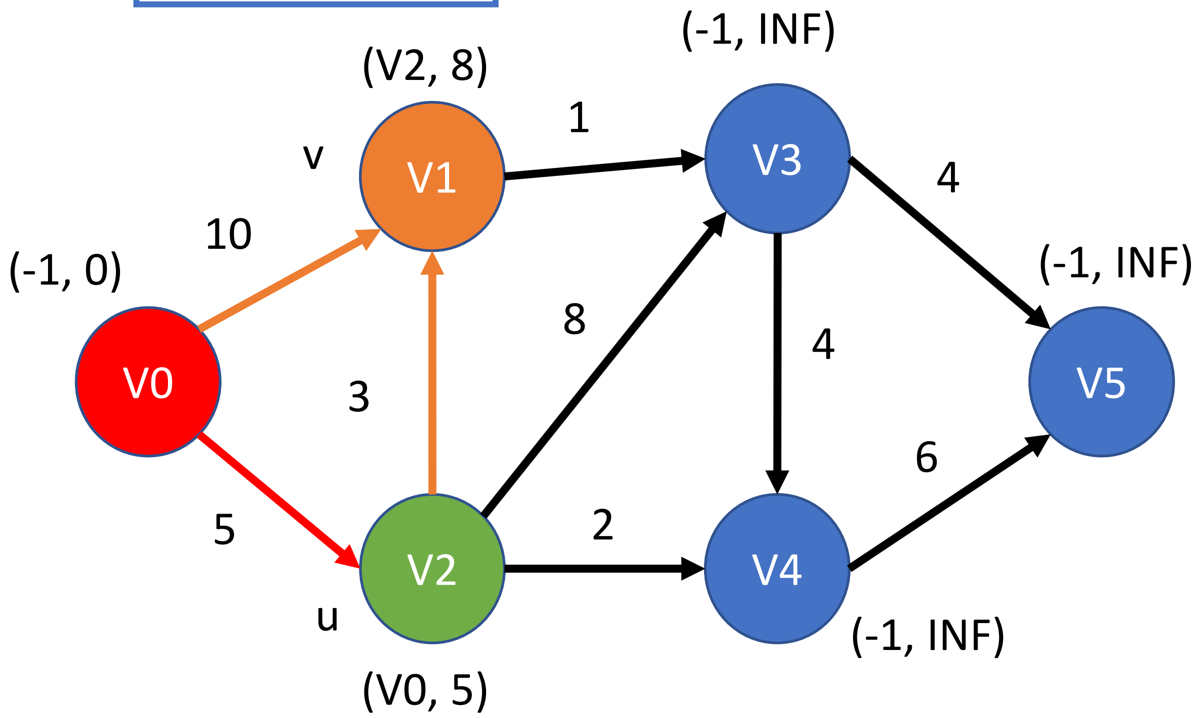


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

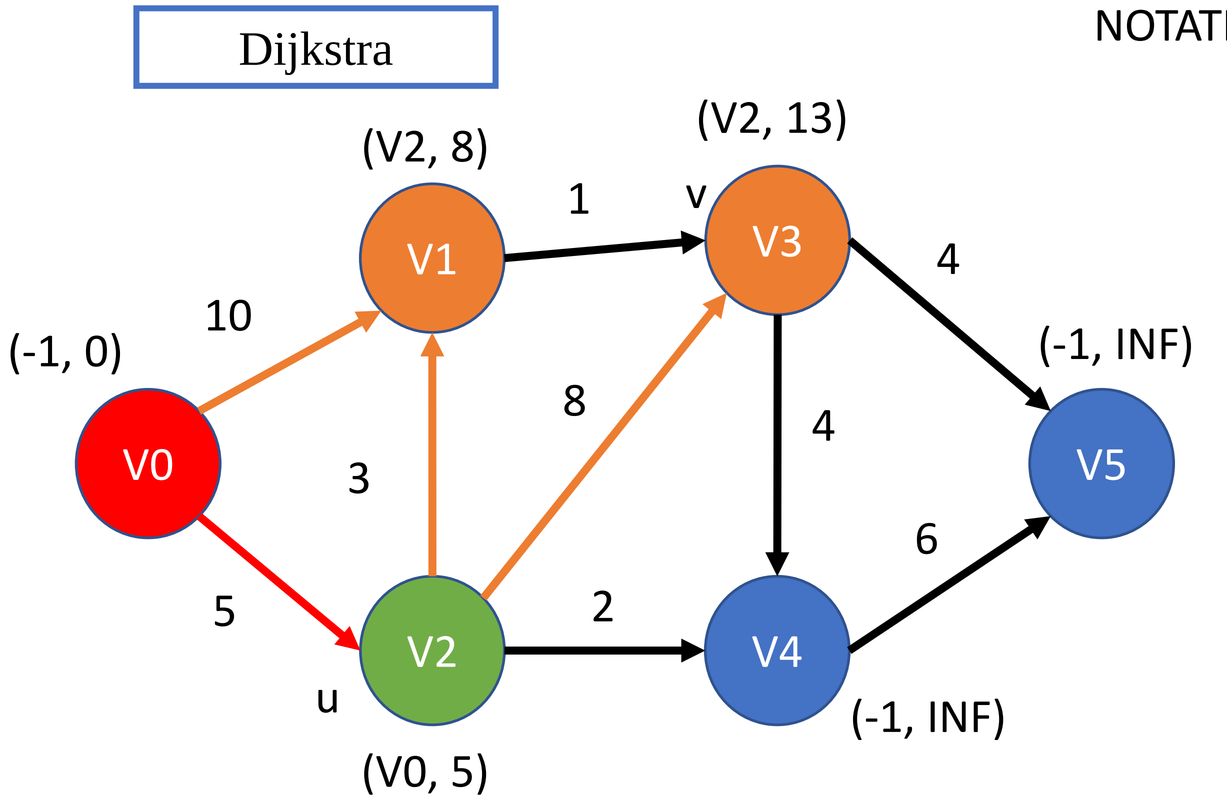
Dijkstra



NOTATION ➡ (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all v
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close u
   * For every open node v adjacent to u: relax edge (u,v)

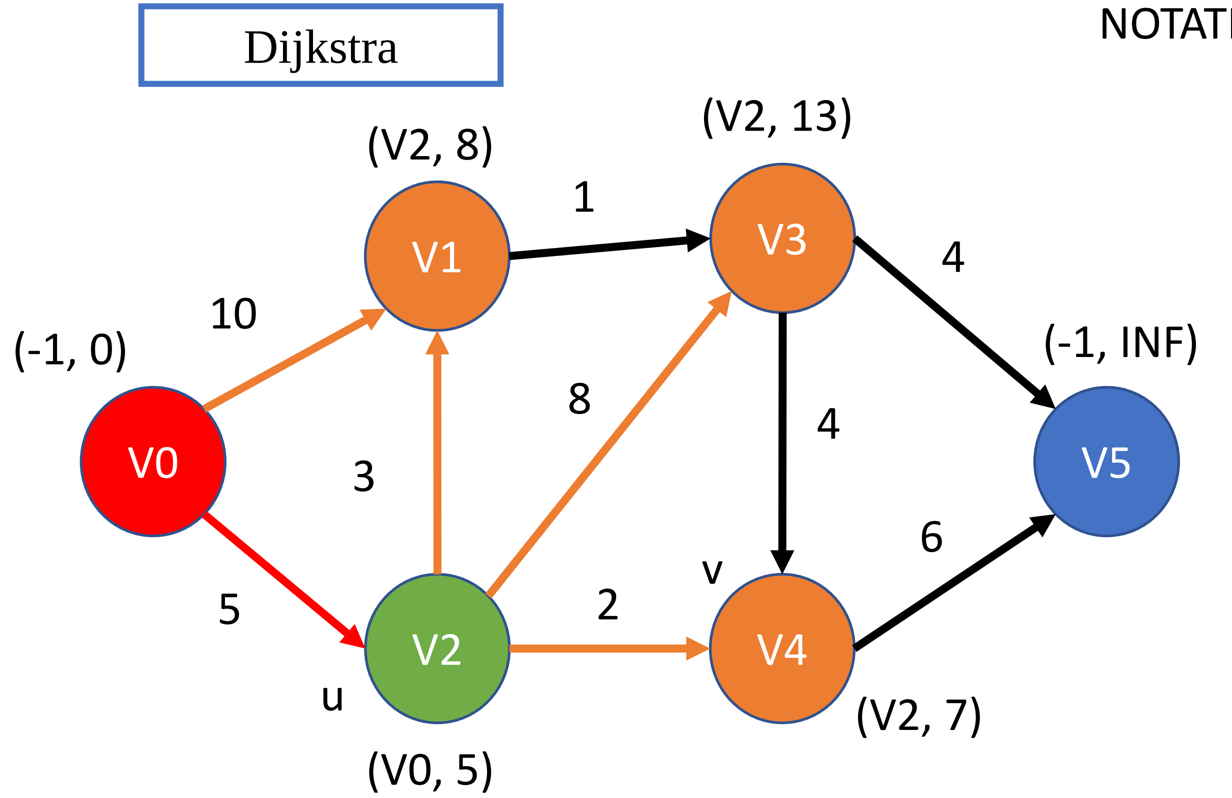
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

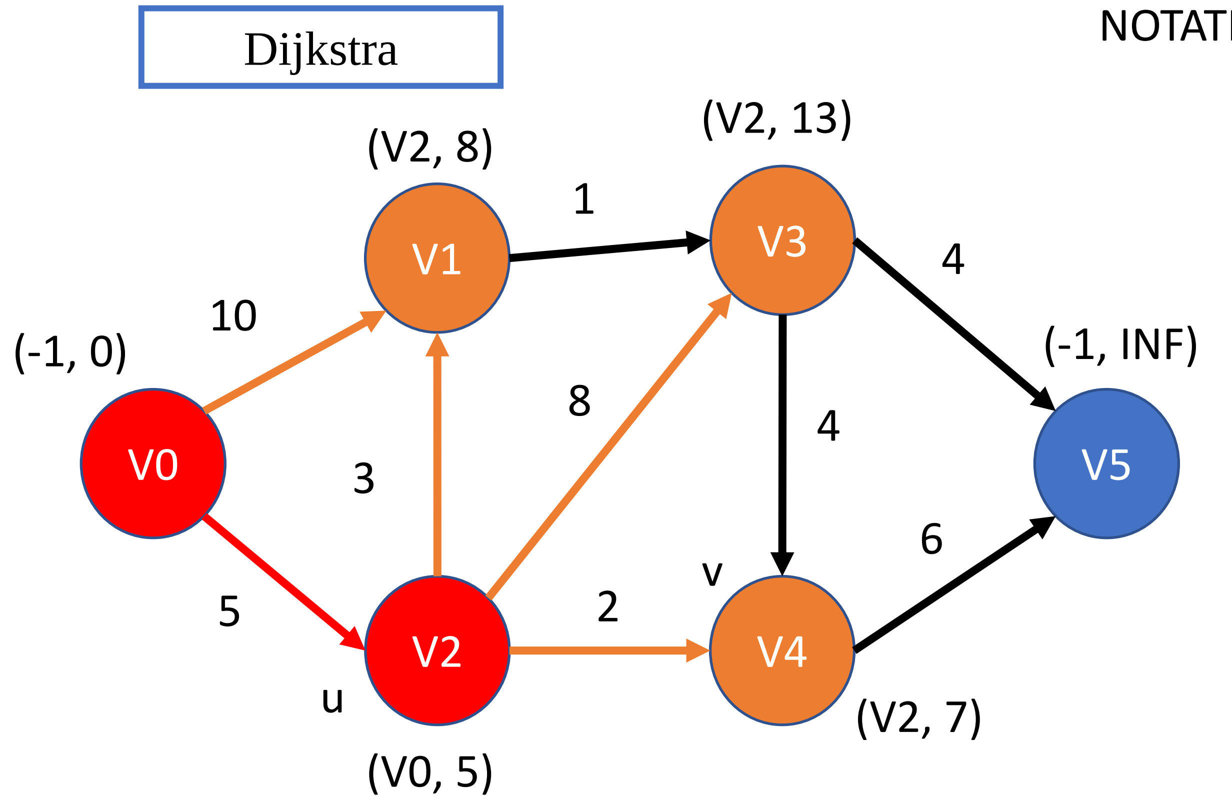
Algorithms



NOTATION ➡ (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>v</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>u</u>
   * For every open node <u>v</u> adjacent to <u>u</u>: relax edge (<u>u,v</u>)

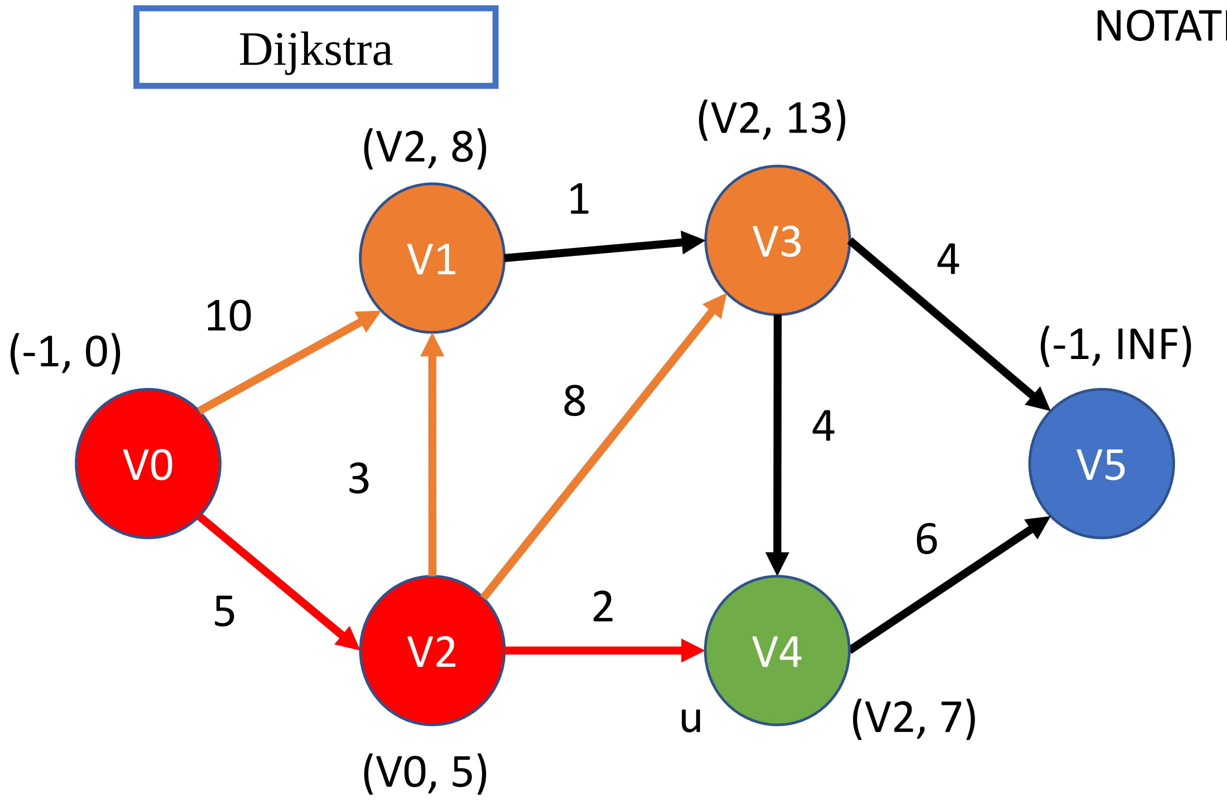
Algorithms



NOTATION ➡ (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all v
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close u
   * For every open node v adjacent to u: relax edge (u,v)

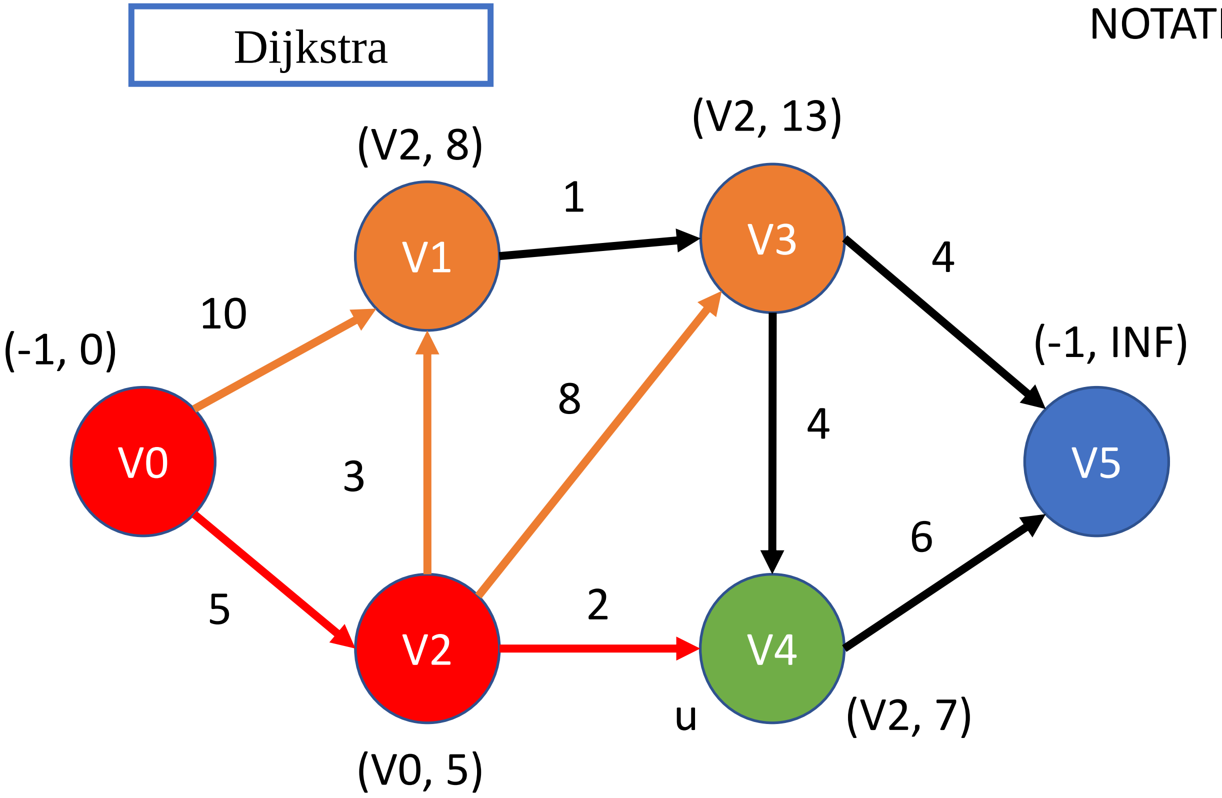
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

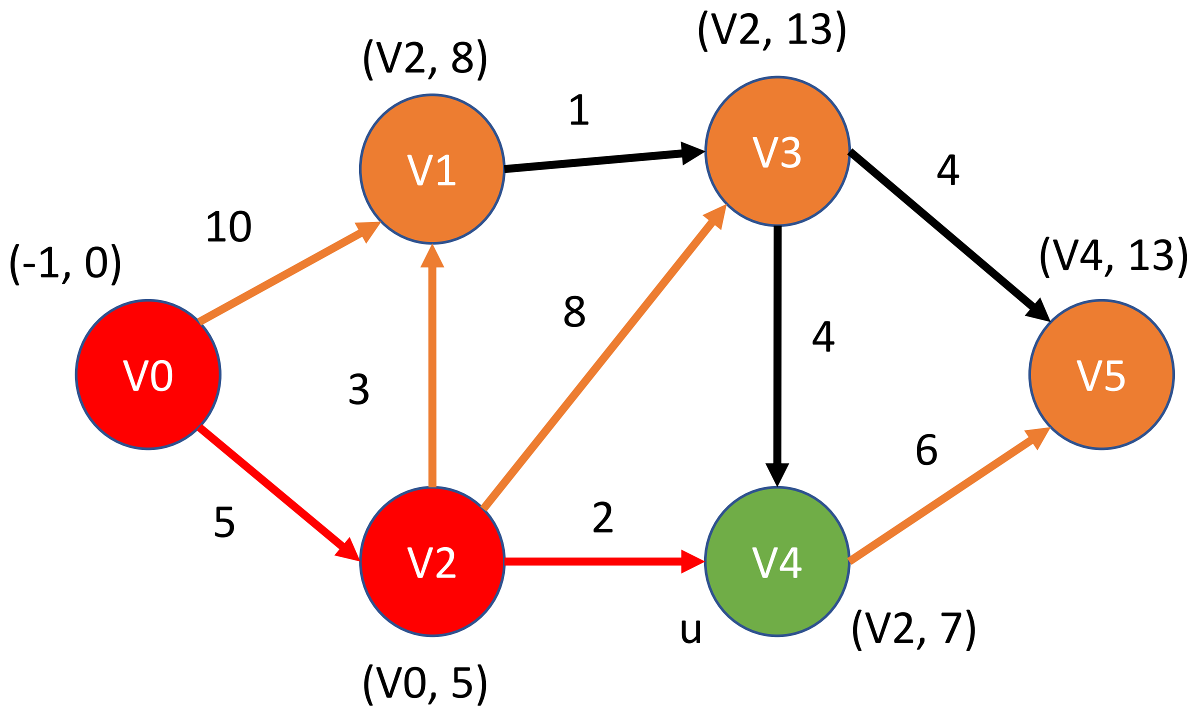


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

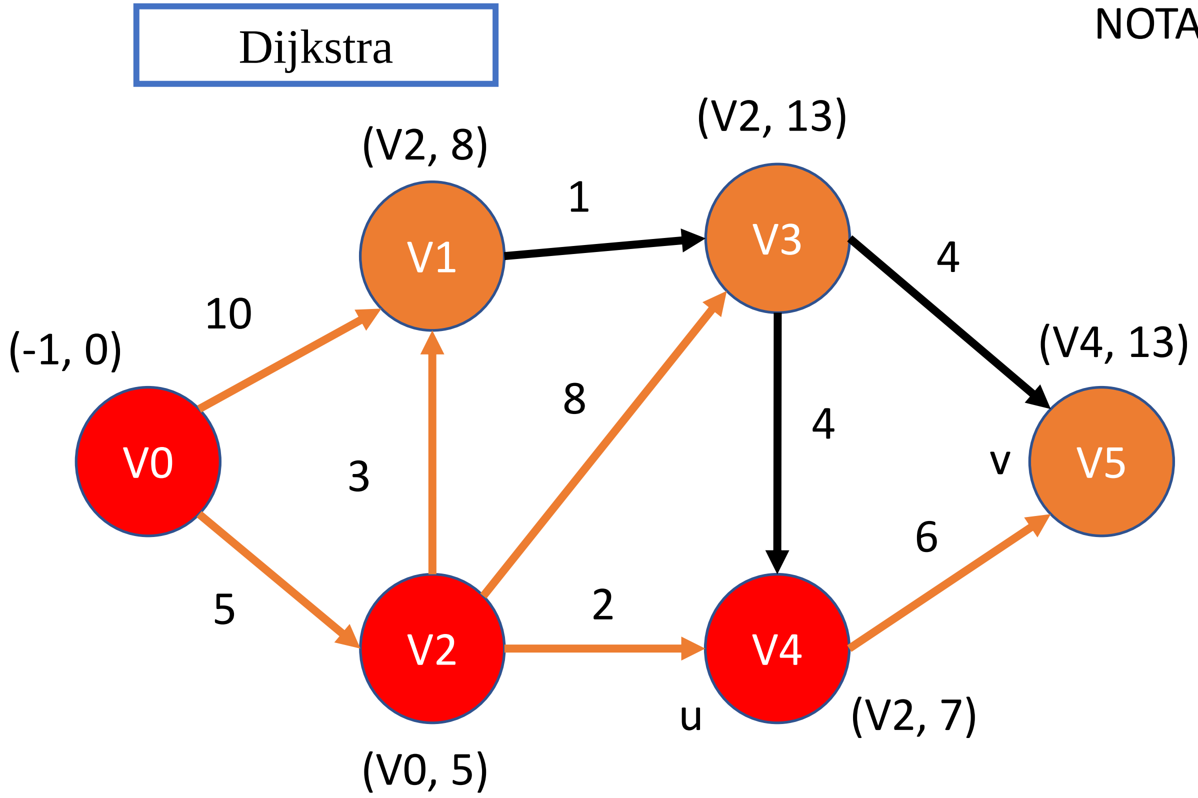
Dijkstra



NOTATION ➡ (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all *v*
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close *u*
   * For every open node *v* adjacent to *u*: relax edge (*u,v*)

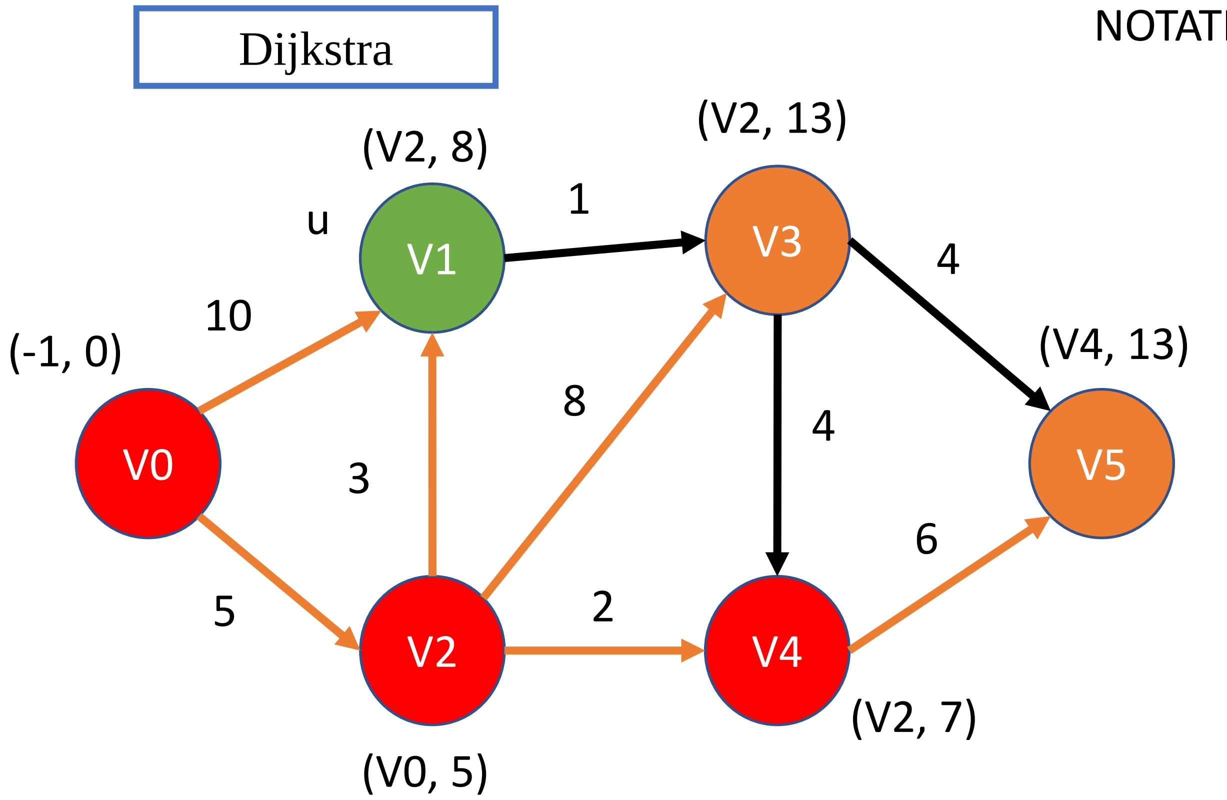
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>v</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

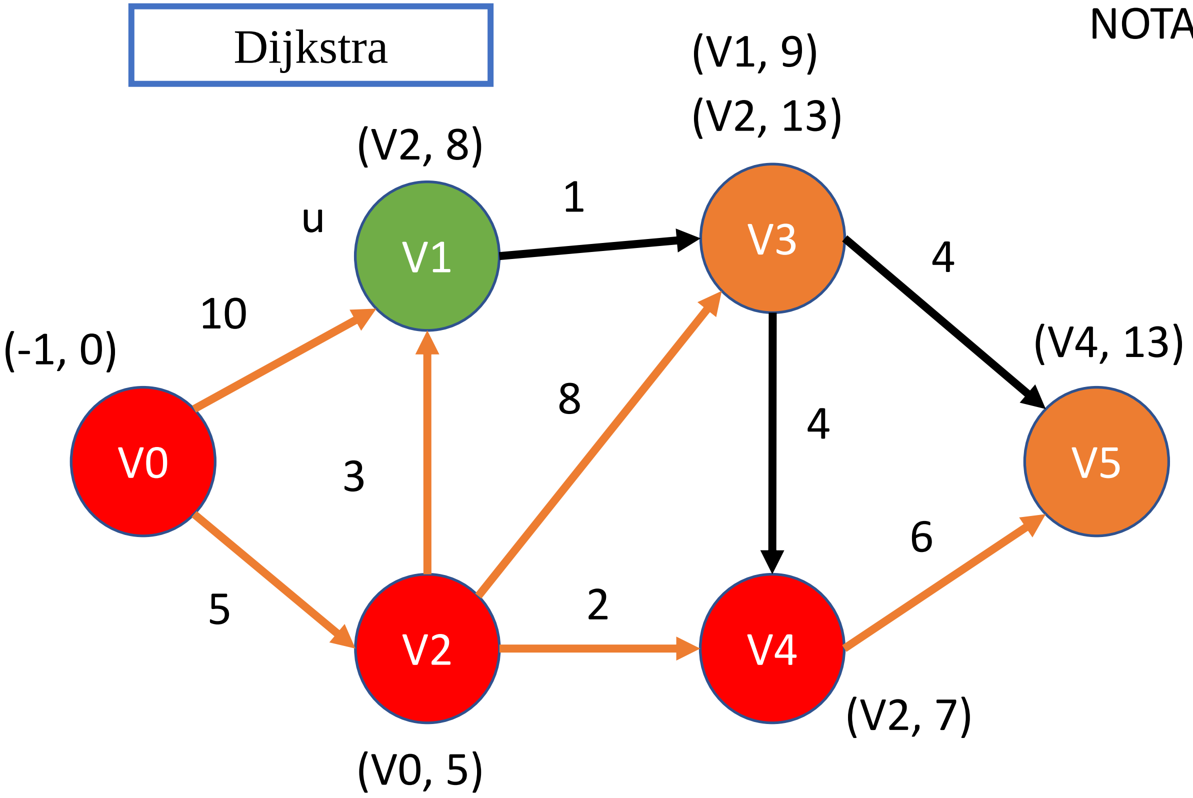
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>v</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>u</u>
   * For every open node <u>v</u> adjacent to <u>u</u>: relax edge (<u>u,v</u>)

Algorithms

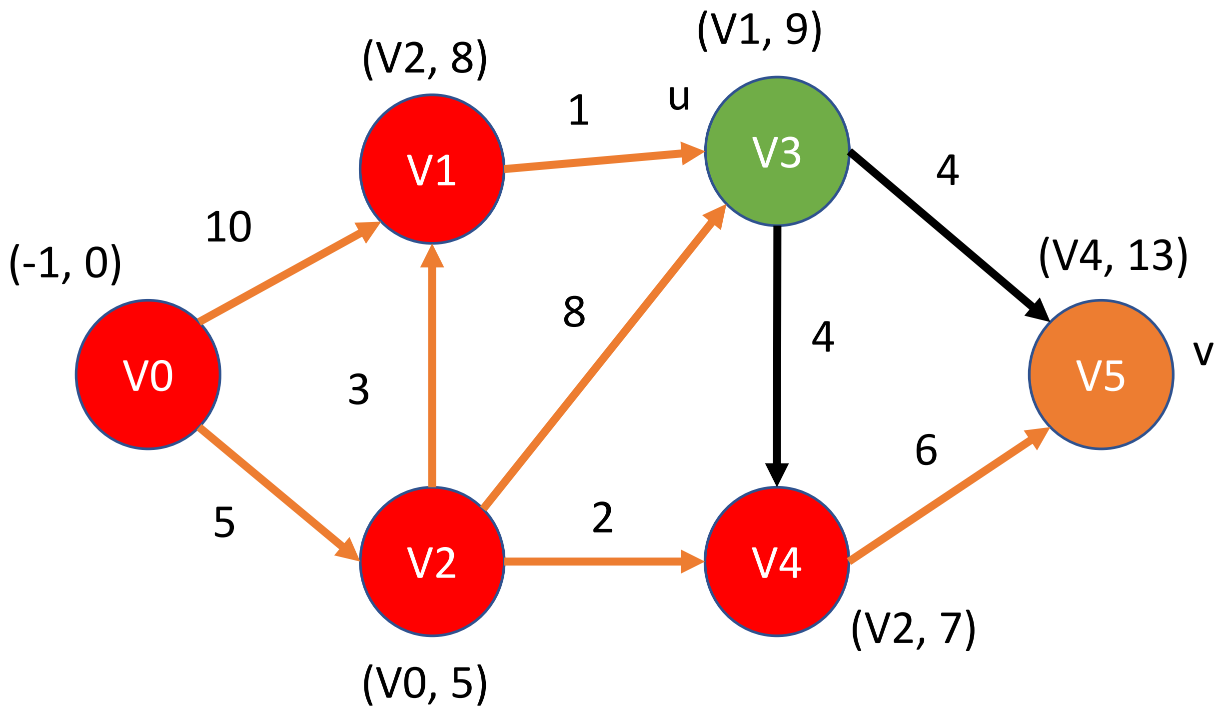


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

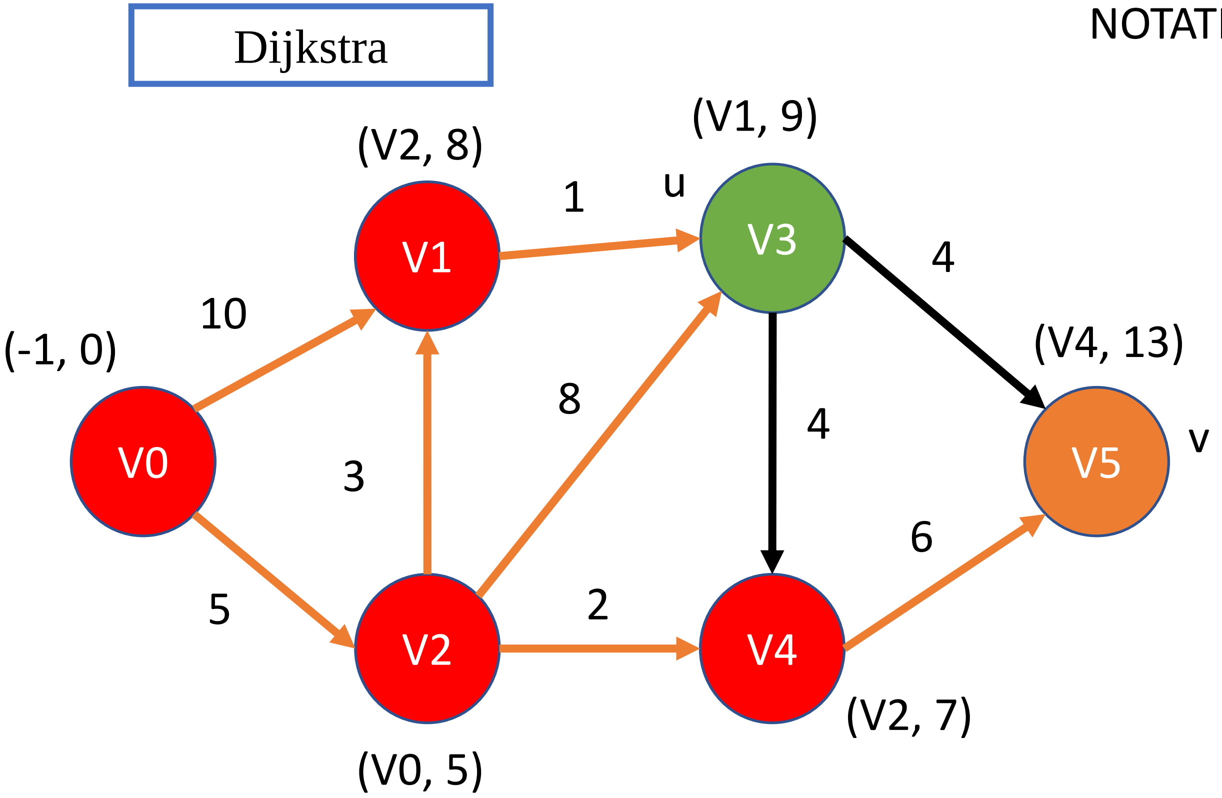
Dijkstra



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all v ≠ s, and p(v) = -1 for all *v*
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close *u*
   * For every open node *v* adjacent to *u*: relax edge (*u,v*)

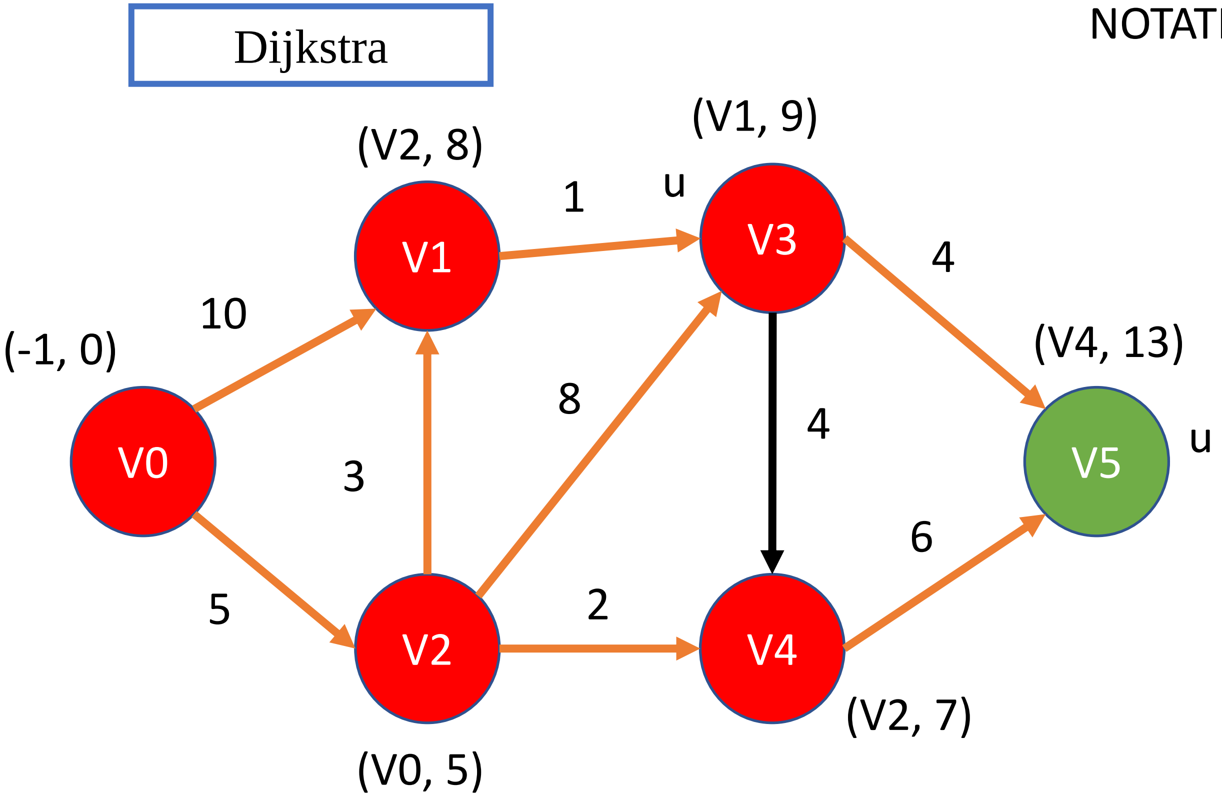
Algorithms



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

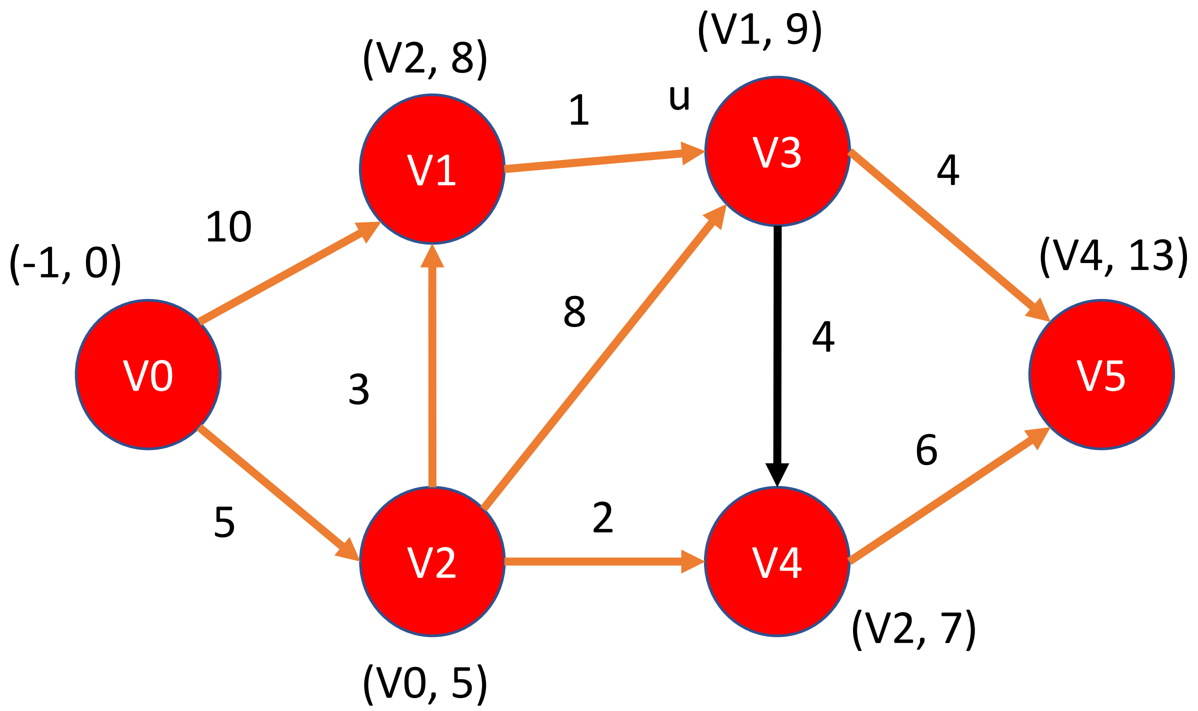


NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms

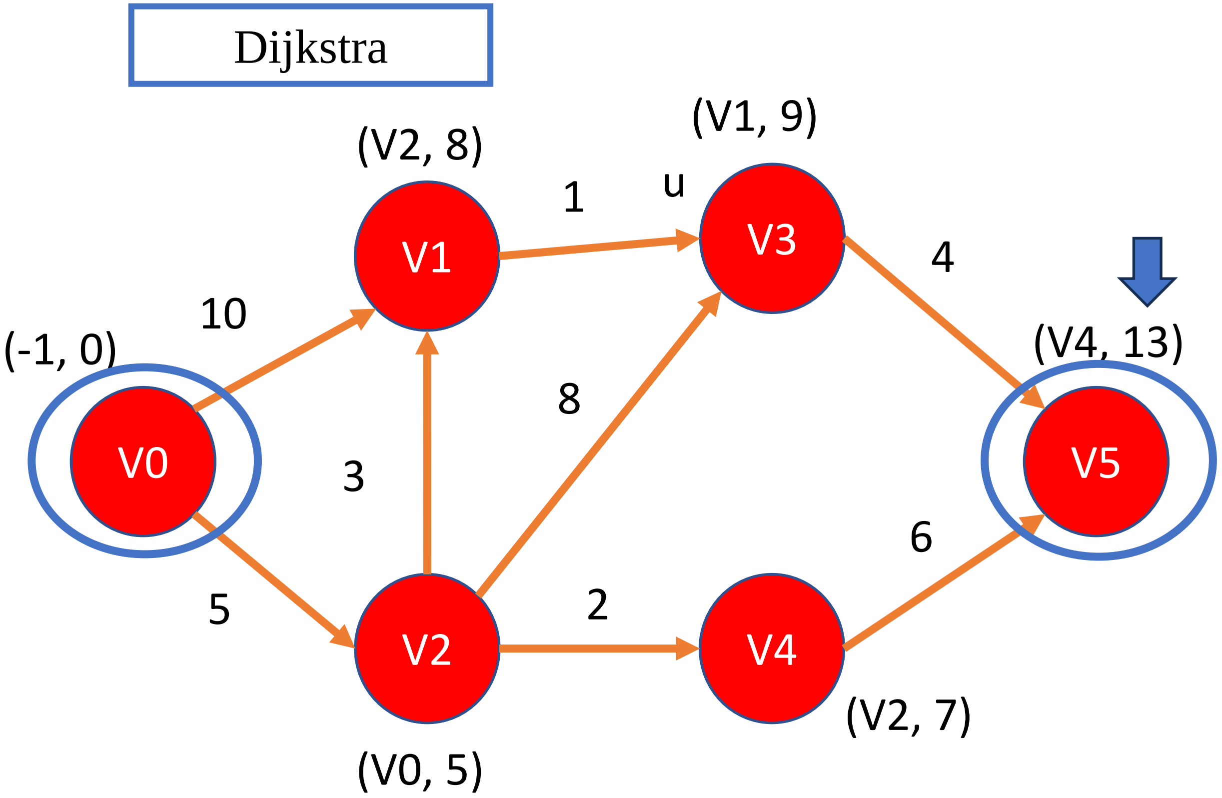
Dijkstra



NOTATION → (father node, weigth)

1. Initialize the graph with d(s) = 0, d(v) = INF, for all <u>v ≠ s</u>, and p(v) = -1 for all <u>*v*</u>
2. Make open(v) = True for every v in the graph
3. As long as there is an open vertex:
   * Choose u whose estimate is the smallest among the open
   * Close <u>*u*</u>
   * For every open node <u>*v*</u> adjacent to <u>*u*</u>: relax edge (<u>*u,v*</u>)

Algorithms



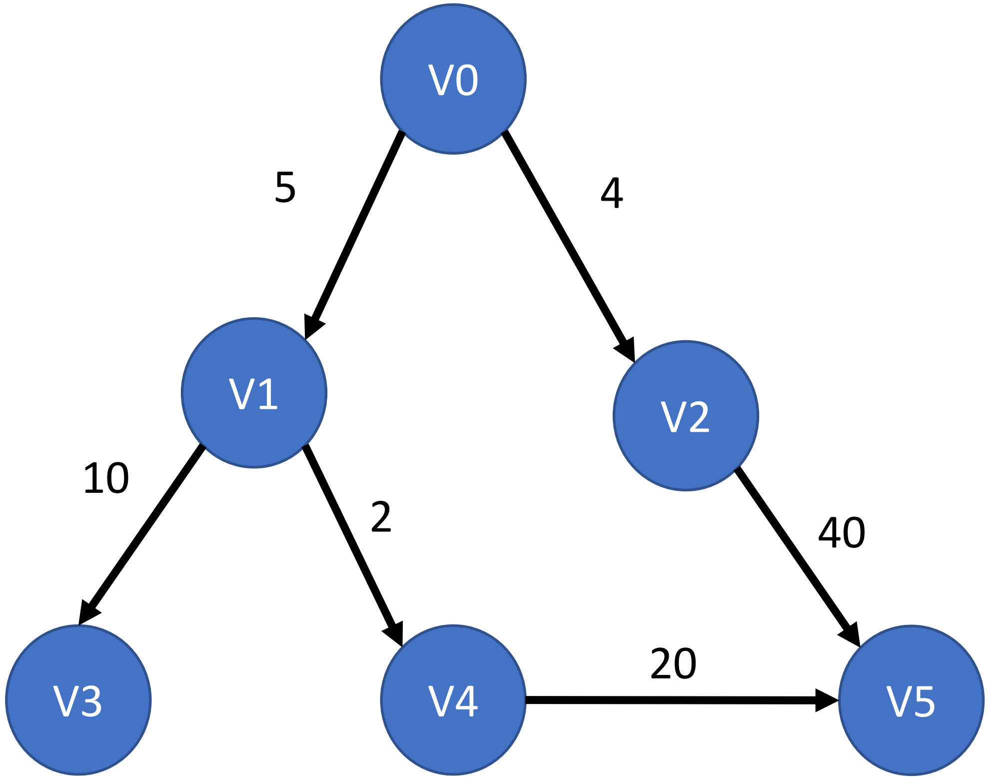
the shortest distance V0 => V5

V5, V4, V2, V0

TOTAL = 13

Algorithms

Dijkstra

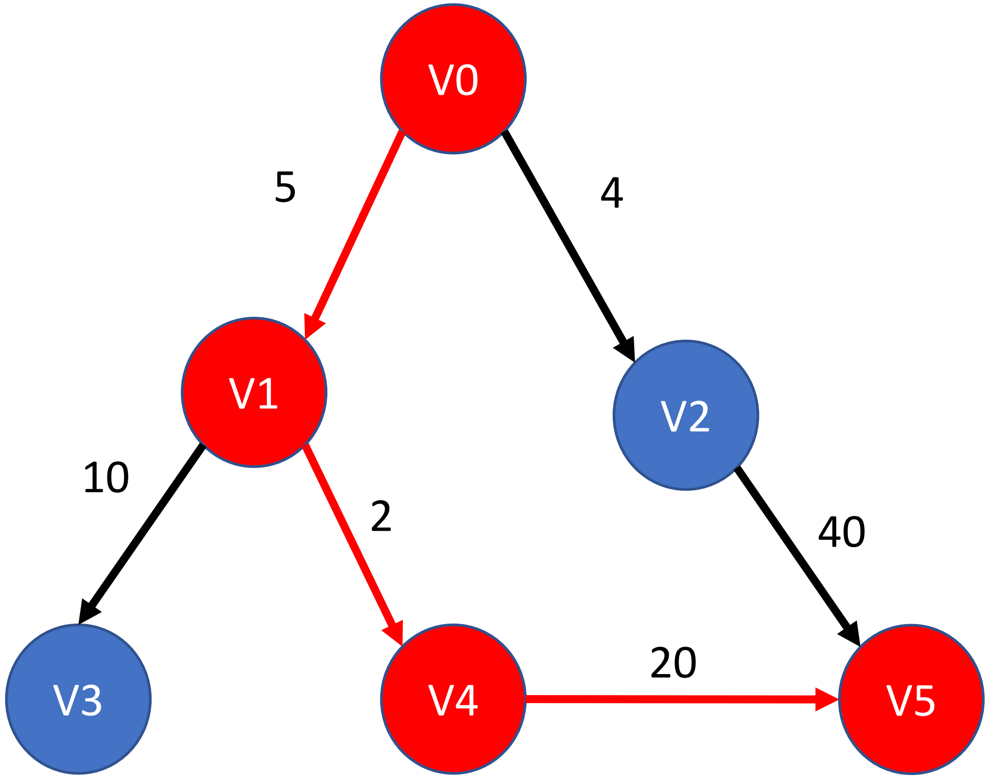



```python
def dijkstra_algorithm(graph, start_node):
    unvisited_nodes = list(graph.get_nodes())
    shortest_path = {}
    previous_nodes = {}
    # We'll use max_value to initialize the "infinity" value of the unvisited nodes
    max_value = sys.maxsize
    for node in unvisited_nodes:
        shortest_path[node] = max_value
    # However, we initialize the starting node's value with 0
    shortest_path[start_node] = 0
    while unvisited_nodes:
        current_min_node = None
        for node in unvisited_nodes: # Iterate over the nodes
            if current_min_node == None:
                current_min_node = node
            elif shortest_path[node] < shortest_path[current_min_node]:
                current_min_node = node
        # The code block below retrieves the current node's neighbors and updates their distances
        neighbors = graph.get_outgoing_edges(current_min_node)
        for neighbor in neighbors:
            tentative_value = shortest_path[current_min_node] + graph.value(current_min_node, neighbor)
            if tentative_value < shortest_path[neighbor]:
                shortest_path[neighbor] = tentative_value
                # We also update the best path to the current node
                previous_nodes[neighbor] = current_min_node
        unvisited_nodes.remove(current_min_node)

    return previous_nodes, shortest_path
```

Algorithms

Dijkstra




```
We found the following best path with a value of 27.
V0 -> V1 -> V4 -> V5
```

Algorithms

A*

A* is an informed search algorithm. Informed Search signifies that the algorithm has extra information, to begin with. It is a complete as well as an optimal solution for solving path and grid problems.

Algorithms

A*

A* is an informed search algorithm. Informed Search signifies that the algorithm has extra information, to begin with. It is a **complete** as well as an **optimal** solution for solving path and grid problems.

Complete – It means that it will find all the available paths from start to end.
Optimal – find the least cost from the starting point to the ending point.

Algorithms

A*

A* is an **informed search** algorithm. **Informed Search** signifies that the algorithm has extra information, to begin with. It is a **complete** as well as an **optimal** solution for solving path and grid problems.

$$f(n) = g(n) + h(n)$$

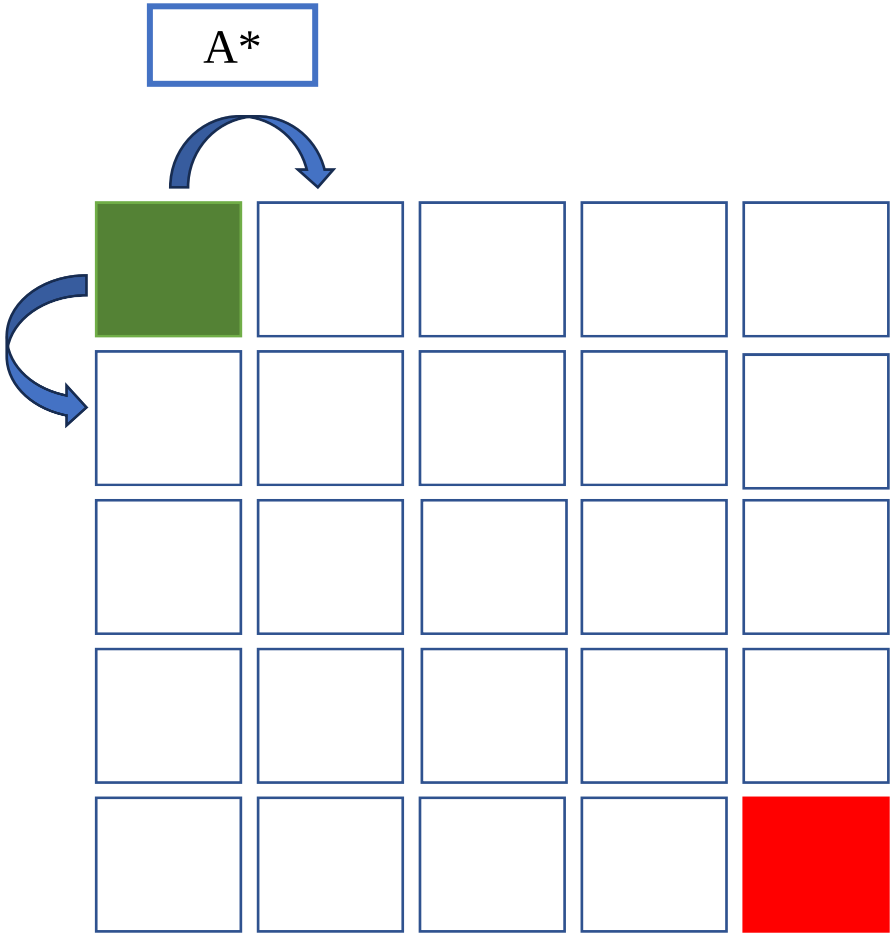
The Algorithm

1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

Algorithms

manhattan((x1, y1), (x2, y2)) = |x1 — x2| + |y1 — y2|

euclidean((x1, y1), (x2, y2)) = sqrt(x^2 + y^2)

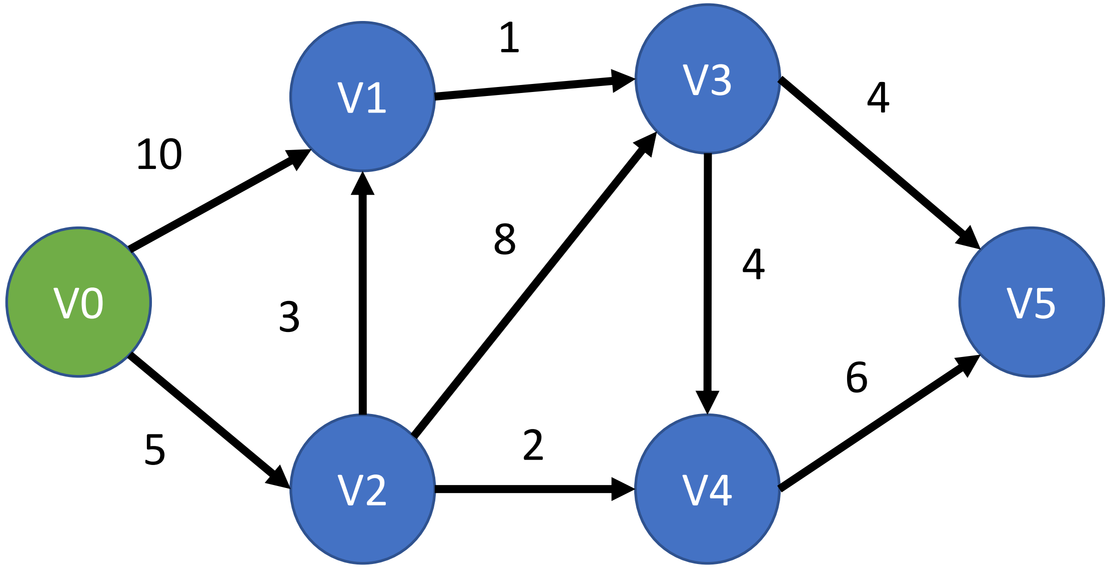


h(n)

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

Algorithms

A*



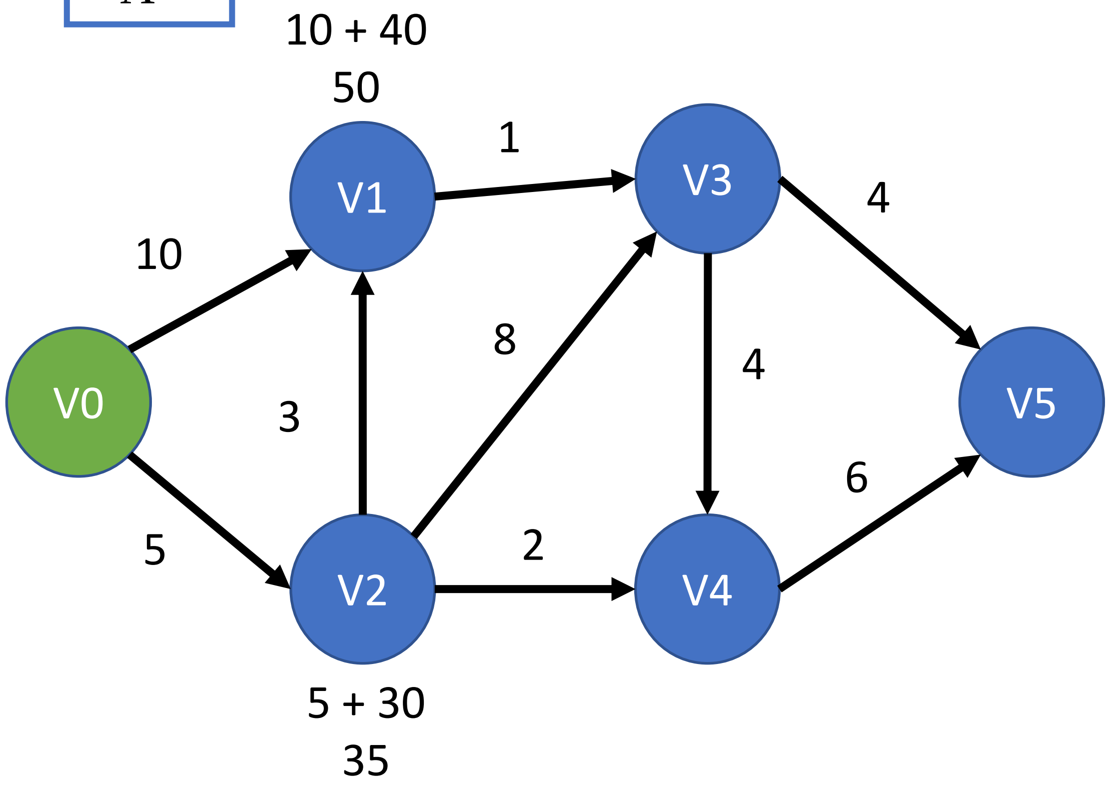
1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n) →

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms

A*

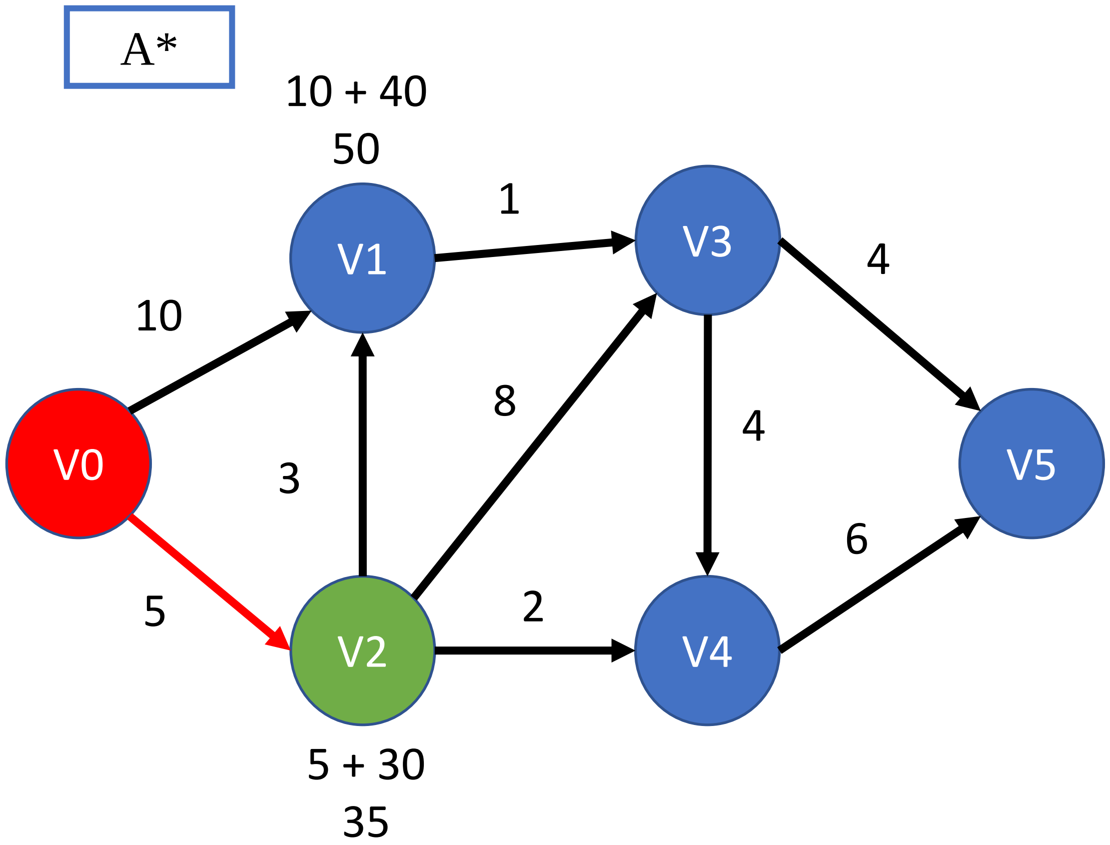


1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n)

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms



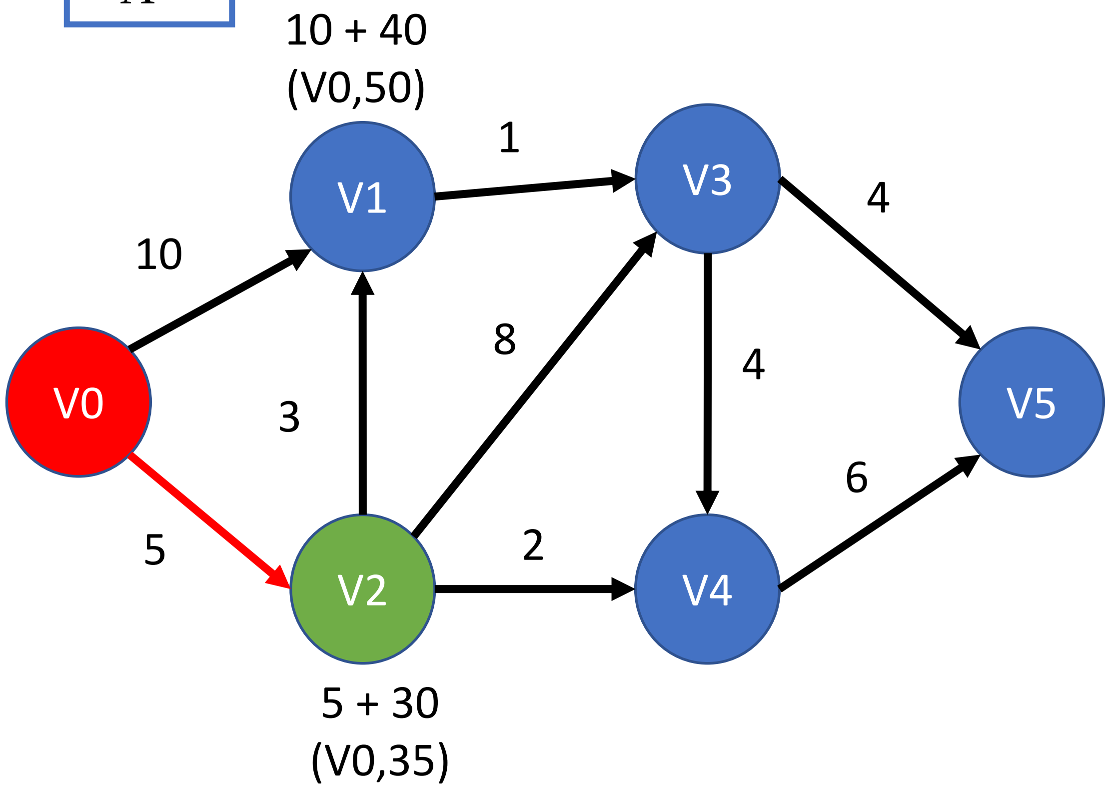
1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n)

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms

A*

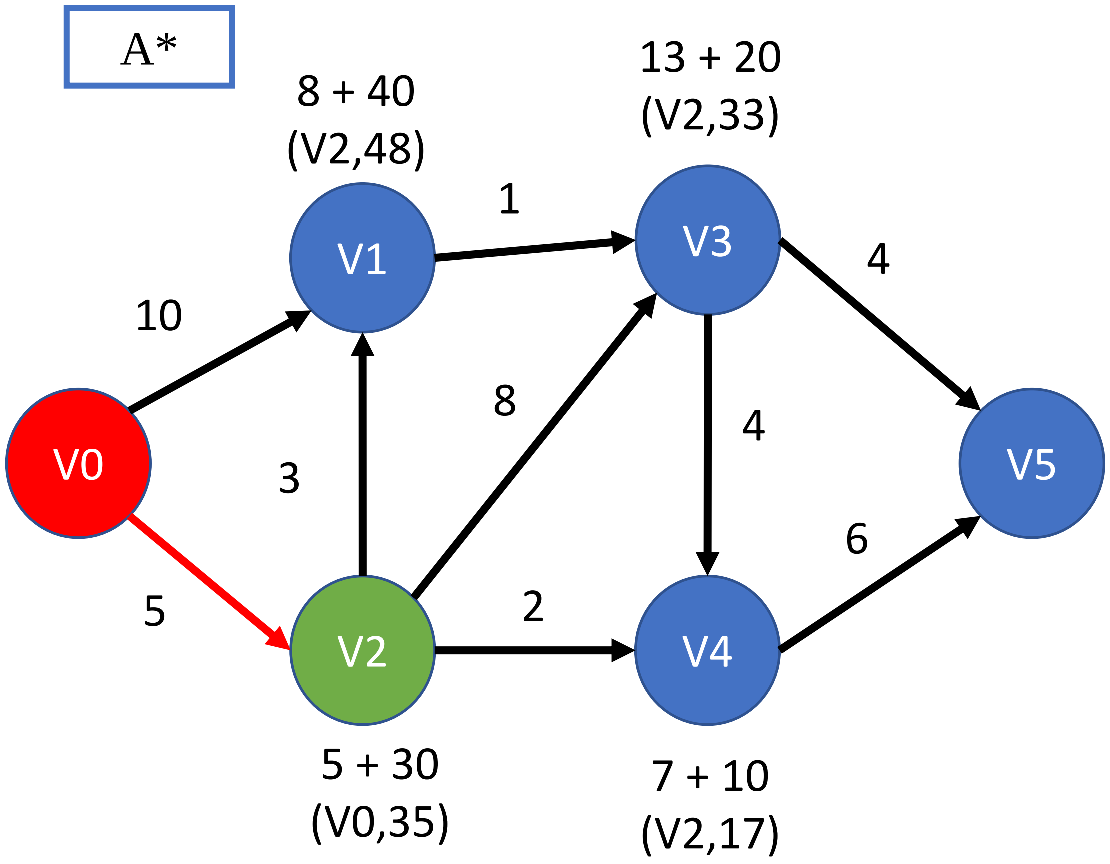


1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n)

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms

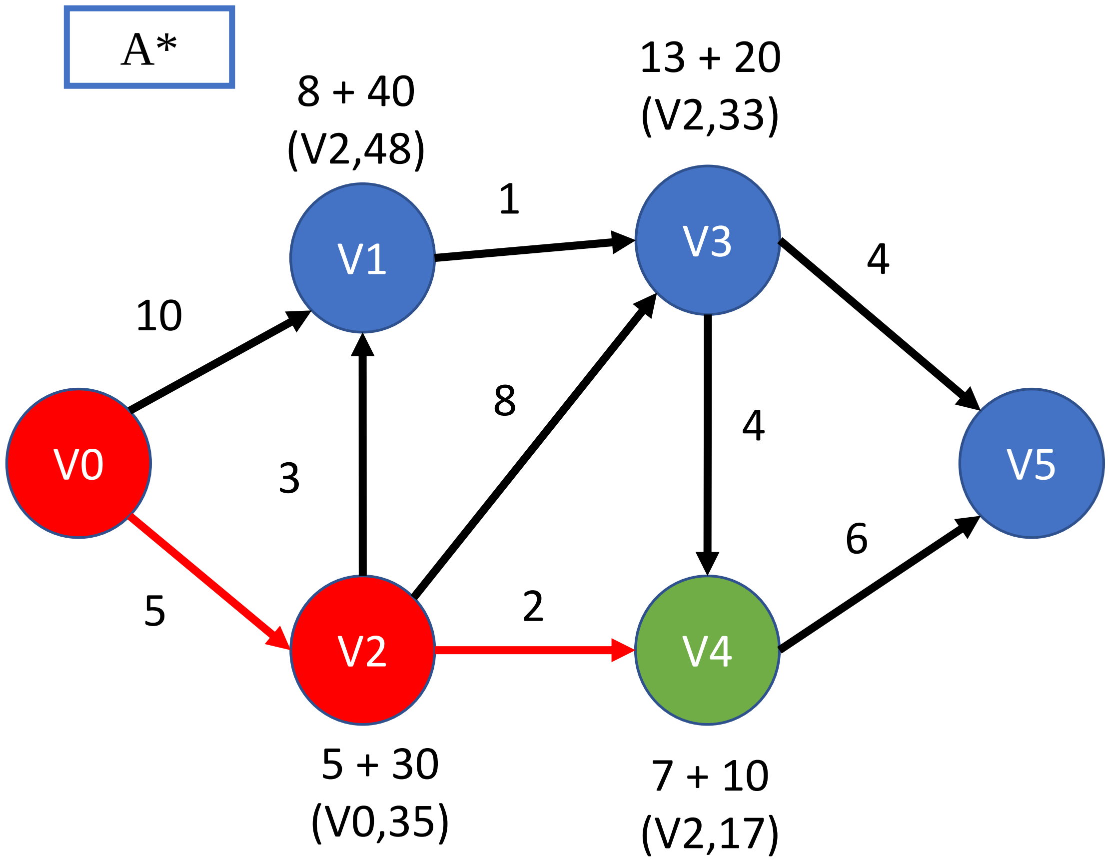


1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n) →

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms

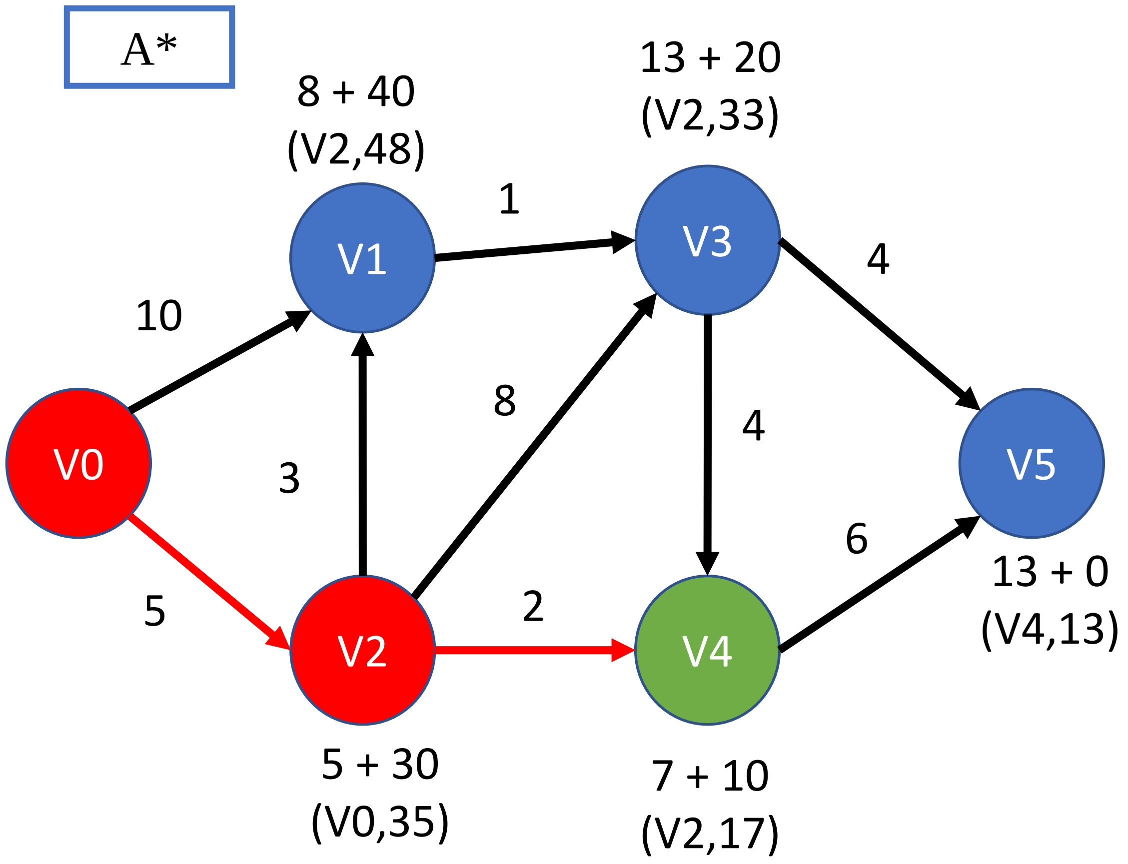


1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n)

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms

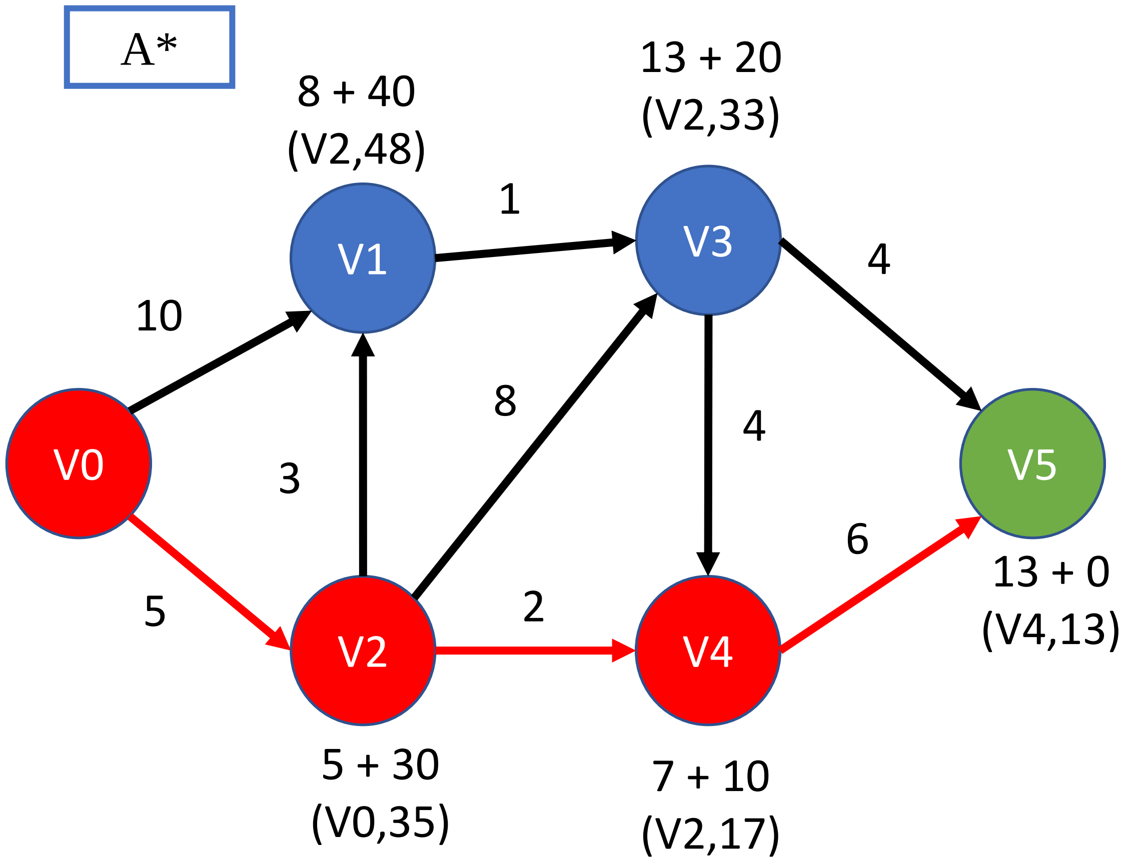


1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
   * If it is a goal node, then stop and return to success.
   * Else remove the node from OPEN, and find all its successors.
   * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n) →

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms



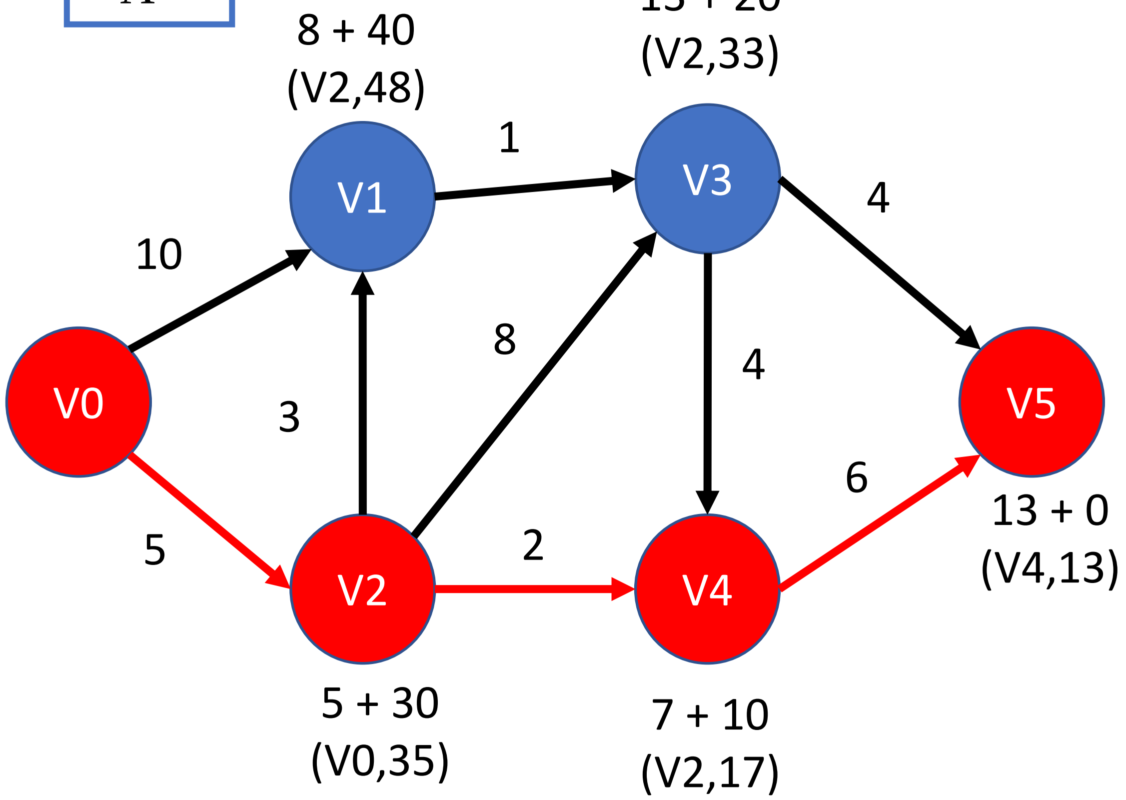
1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
    * If it is a goal node, then stop and return to success.
    * Else remove the node from OPEN, and find all its successors.
    * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n) →

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms

A*

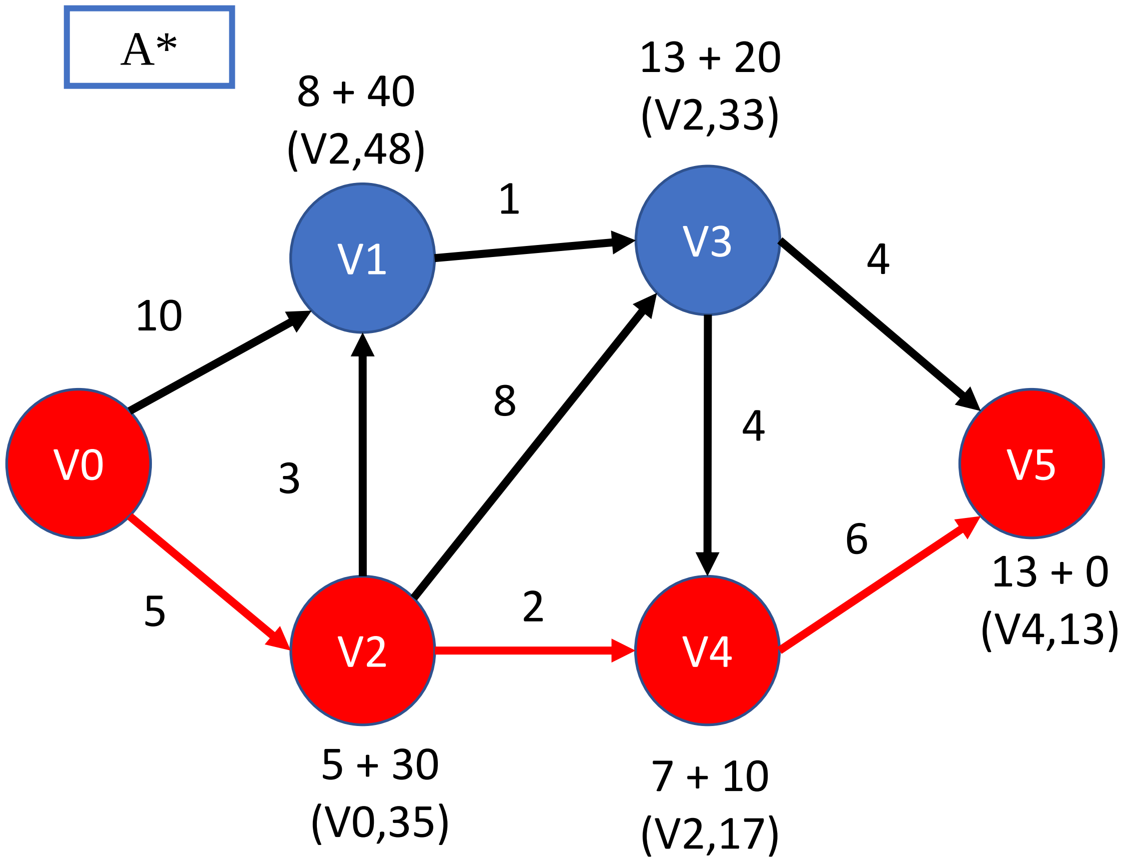


1. Place the starting node into OPEN and find its f (n) value.
2. Remove the node from OPEN, having the smallest f (n) value.
    * If it is a goal node, then stop and return to success.
    * Else remove the node from OPEN, and find all its successors.
    * Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.

h(n)

	V5
V0	50
V1	40
V2	30
V3	20
V4	10

Algorithms



the shortest distance V0 => V5

V5, V4, V2, V0

Algorithms

manhattan((x1, y1), (x2, y2)) = |x1 — x2| + |y1 — y2|

euclidean((x1, y1), (x2, y2)) = sqrt(x^2 + y^2)

A*

0	1	1	1	1
1	1	1	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

0	1+7 8	1	1	1
1+7 8	1	1	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

0	1+7 8	1	1	1
1+7 8	1	1	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

0	1+7 8	2+6 8	1	1
1+7 8	2+6 8	1	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

0	1+7 8	2+6 8	1	1
1+7 8	2+6 8	1	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

0	1+7 8	2+6 8	3+5 8	1
1+7 8	2+6 8	3+5 8	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

0	1+7 8	2+6 8	3+5 8	1
1+7 8	2+6 8	3+5 8	1	1
1				1
1	1	1	1	1
1	1	1	1	1

h(n)

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

Algorithms

A*

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

h(n)

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

Algorithms

A*

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

h(n)

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

Algorithms

A*

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

h(n)

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

Algorithms

A*

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

h(n)

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

Algorithms

A*

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

h(n)

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

Algorithms

A*

1	1+7 8	2+6 8	3+5 8	4+4 8
1+7 8	2+6 8	3+5 8	4+4 8	5+3 8
1				6+2 8
1	1	1	1	7+1 8
1	1	1	1	1

h(n)

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

Algorithms

A*

1	1+7 8	2+6 8	3+5 8	4+4 8
1+7 8	2+6 8	3+5 8	4+4 8	5+3 8
1				6+2 8
1	1	1	8+2 10	7+1 8
1	1	1	1	8+0 8

h(n)

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

Algorithms

A*

0	1+7 8	2+6 8	3+5 8	4+4 8
1+7 8	2+6 8	3+5 8	4+4 8	5+3 8
1				6+2 8
1	1	1	8+2 10	7+1 8
1	1	1	1	8+0 8

h(n)

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

Daniel Nogueira

dnogueira@ipca.pt