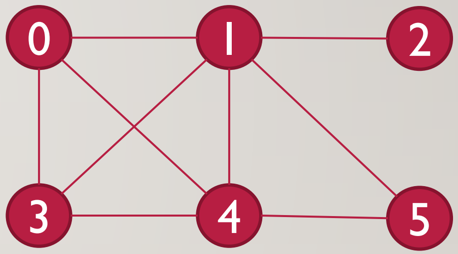
ANALYSIS OF ALGORITHMS

LECTURE 10 : SHORTEST PATH TREES

BASED ON SECTION 6.1

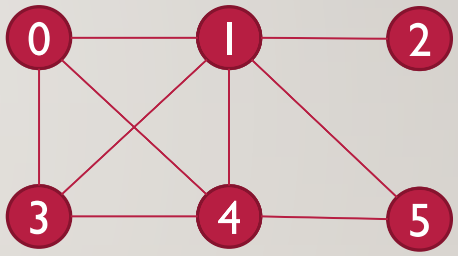
SHORTEST PATH

- Let's say you wanted to find the shortest path from some vertex to every other vertex
- What would you do?

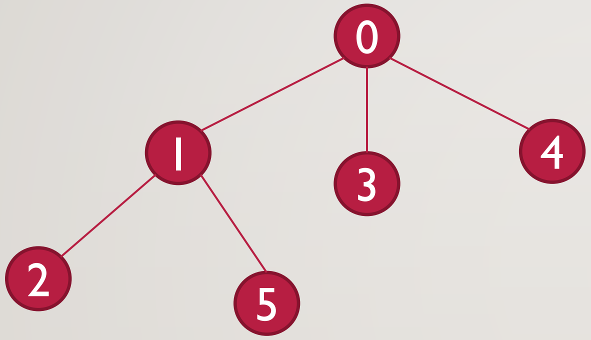


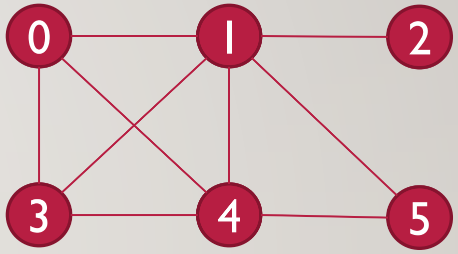
SHORTEST PATH

- Let's say you wanted to find the shortest path from some vertex to every other vertex
- What would you do?
- Breadth first search!
  - Because it tries to make shorter, wider trees, it is actually finding the shortest path from the root to anywhere else



BREADTH FIRST SEARCH TREE

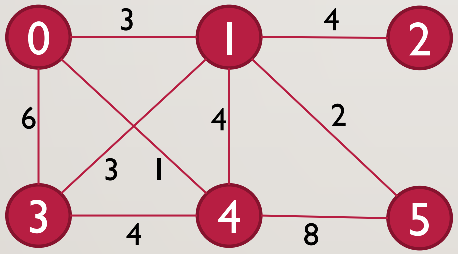




- Consider if there was a shorter path from 0 to 2
- That would have to be the edge 0,2
- But then that would have been in the tree
- The tree contains the shortest path from the root to any vertex

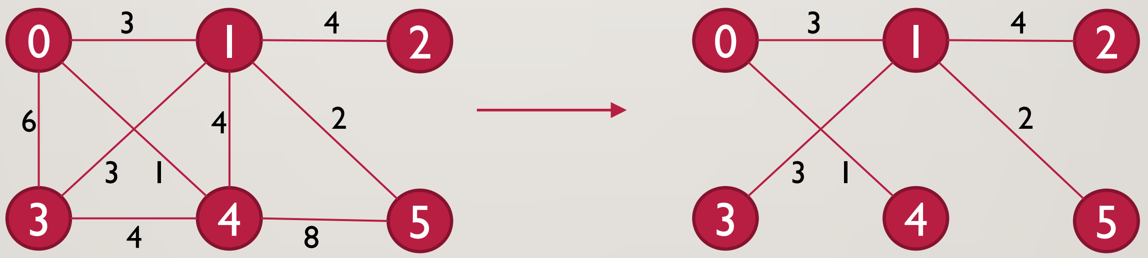
WEIGHTED GRAPHS

- What do we do if the graph is weighted?
- Minimum Weighted Spanning Tree?



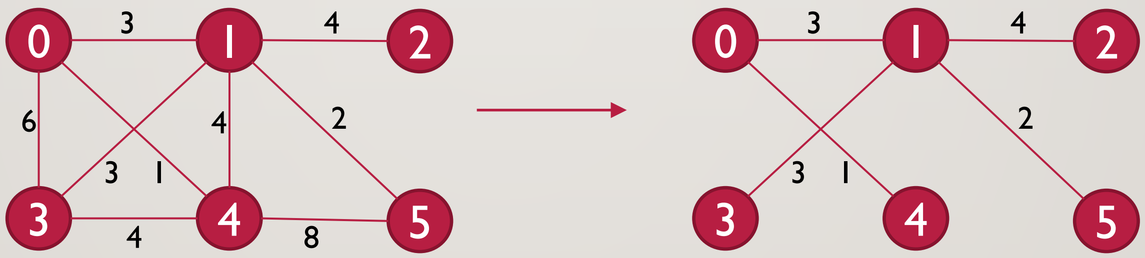
WEIGHTED GRAPHS

- What do we do if the graph is weighted?
- Minimum Weighted Spanning Tree?



WEIGHTED GRAPHS

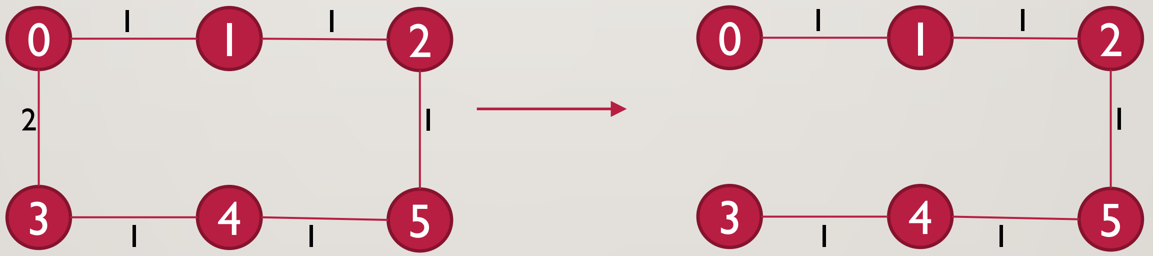
- What do we do if the graph is weighted?
- Minimum Weighted Spanning Tree?



- Finds the smallest total cost for a connected subtree, not the shortest path
- Consider the path between vertex 3 and vertex 4

WEIGHTED GRAPHS

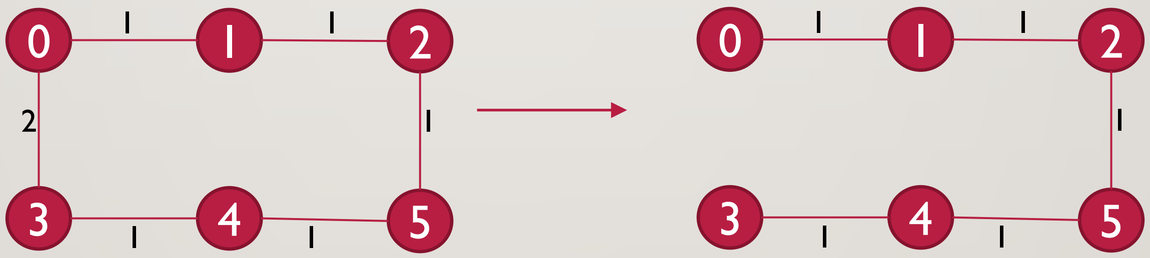
- What do we do if the graph is weighted?
- Minimum Weighted Spanning Tree?



- Finds the smallest total cost for a connected subtree, not the shortest path
- Consider the path between vertex 0 and vertex 3

WEIGHTED GRAPHS

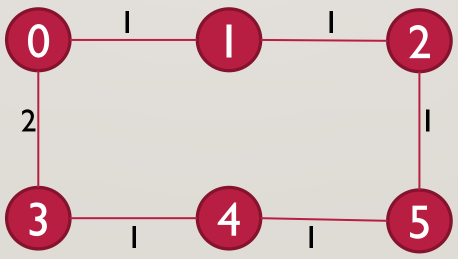
- What's wrong and how do we fix it?



- The MWST algorithm is greedy. It only cares what the cheapest edge right now is
- We need to keep track of the cheapest edge **cumulatively**

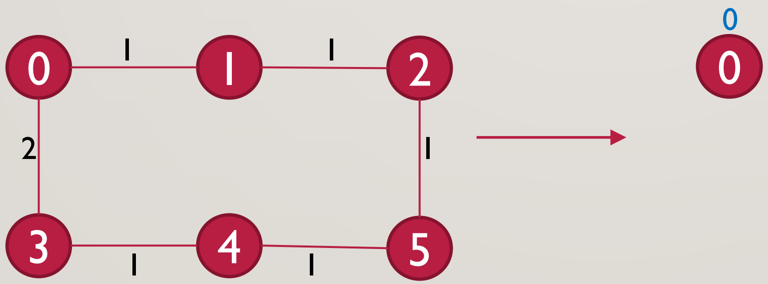
DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



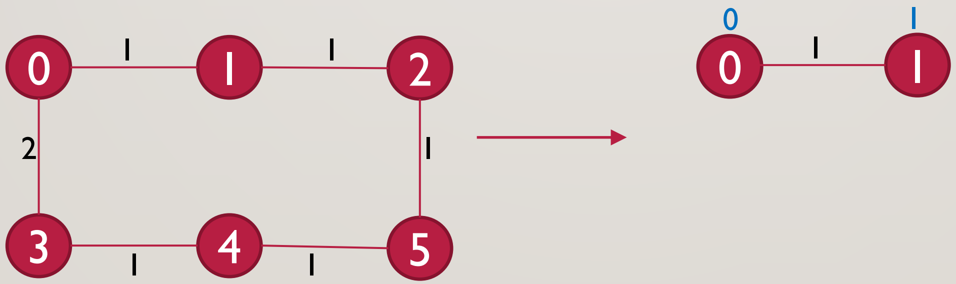
DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



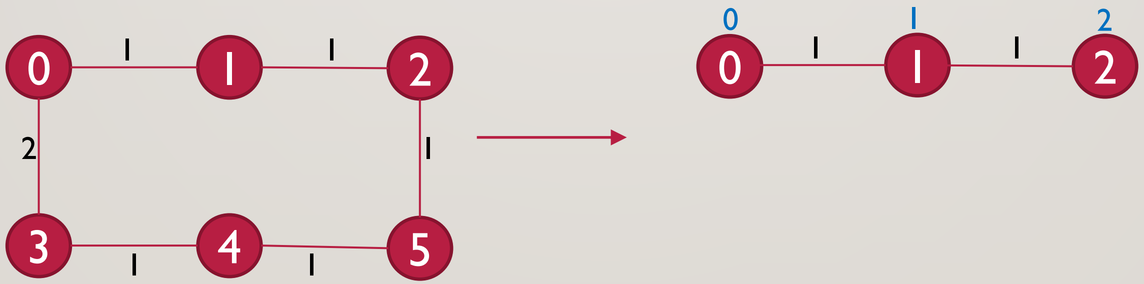
DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



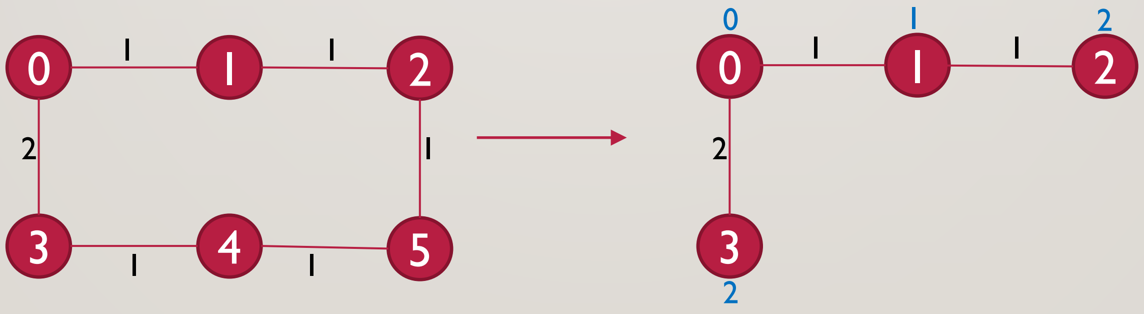
DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



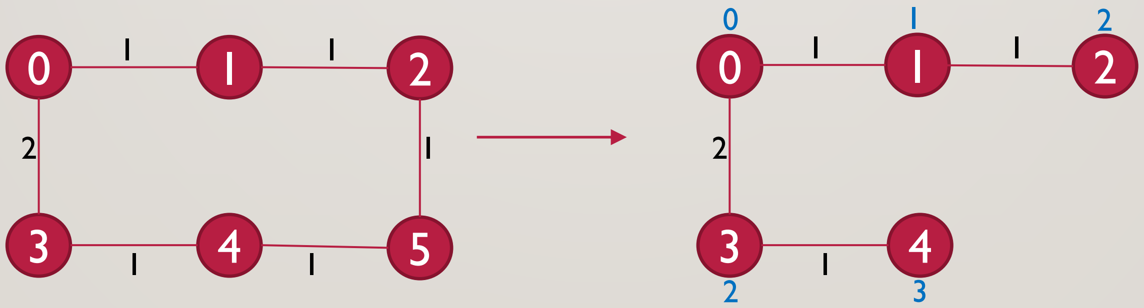
DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



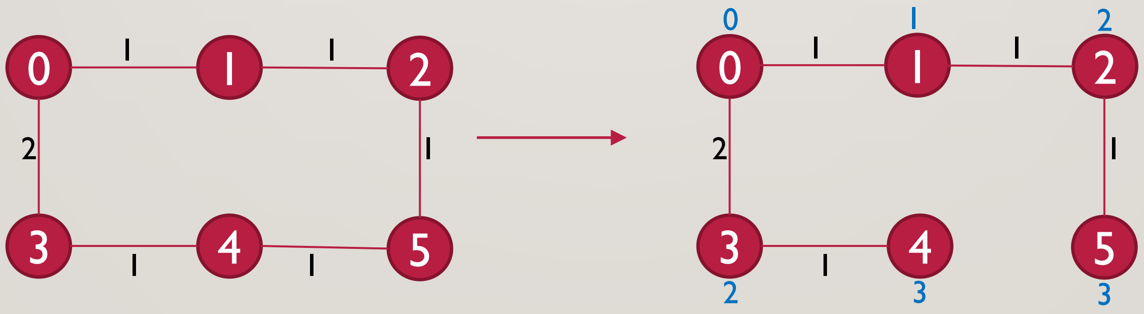
DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



DIJKSTRA'S ALGORITHM

- Pretty much the same as the Prim's algorithm (The minimum weighted spanning tree algorithm we covered in class)
- But now instead of just picking the edge with the lowest cost, we pick the edge with the lowest cumulative cost.
- That means we need to keep track of the cumulative cost to each vertex



DIJKSTRA'S ALGORITHM

- Start off with the root vertex
- Repeatedly
  - Add the edge with the lowest cumulative cost from the original graph going from a vertex in the tree to one not in the tree
  - Stop when you have all the vertices from original tree

DIJKSTRA'S ALGORITHM

- Start off with the root vertex
- Repeatedly
  - Add the edge with the lowest cumulative cost from the original graph going from a vertex in the tree to one not in the tree
  - Stop when you have all the vertices from original tree
- Data structures needed
  - Parent array – stores the parent of each vertex
  - Cost array – keeps best known cost to each vertex
  - Marked array – stores whether a vertex is in the tree

Cost	0	1	2	3	4	5

Parent	0	1	2	3	4	5

Marked	0	1	2	3	4	5

DIJKSTRA'S ALGORITHM

- marked[root]=true, parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	-1	-1	-1	-1	-1	-1

Parent	0	1	2	3	4	5
	-1	-1	-1	-1	-1	-1

Marked	0	1	2	3	4	5
	f	f	f	f	f	f

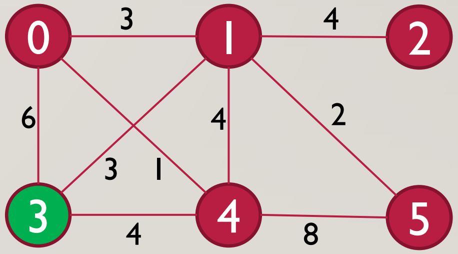
DIJKSTRA'S ALGORITHM

- marked[root]=true, parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	-1	-1	-1	-1	-1	-1

Parent	0	1	2	3	4	5
	-1	-1	-1	-1	-1	-1

Marked	0	1	2	3	4	5
	f	f	f	f	f	f



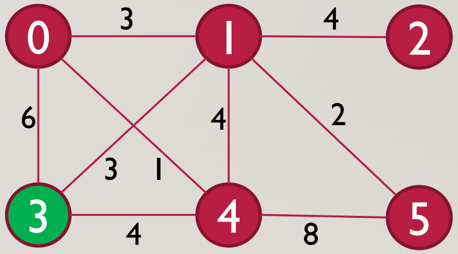
DIJKSTRA'S ALGORITHM

- marked[root]=true, parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
    - Find the unmarked vertex **y** with the lowest cost
    - Set marked[y]=true
    - For every unmarked neighbour **z**
        - The current cost of getting to z is in the cost array
        - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
        - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
    - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	-1	-1	-1	0	-1	-1

Parent	0	1	2	3	4	5
	-1	-1	-1	3	-1	-1

Marked	0	1	2	3	4	5
	f	f	f	f	f	f



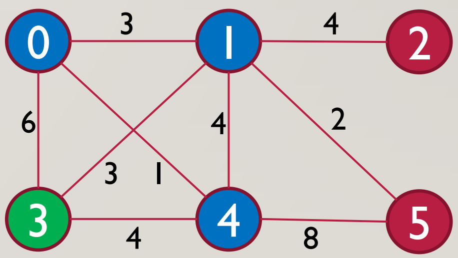
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	**0**	**1**	**2**	**3**	**4**	**5**
	-1	-1	-1	0	-1	-1

Parent	**0**	**1**	**2**	**3**	**4**	**5**
	-1	-1	-1	3	-1	-1

Marked	**0**	**1**	**2**	**3**	**4**	**5**
	f	f	f	t	f	f



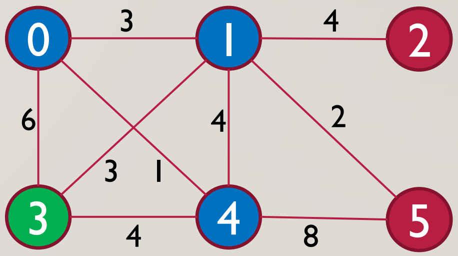
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	**0**	**1**	**2**	**3**	**4**	**5**
	6	-1	-1	0	-1	-1

Parent	**0**	**1**	**2**	**3**	**4**	**5**
	3	-1	-1	3	-1	-1

Marked	**0**	**1**	**2**	**3**	**4**	**5**
	f	f	f	t	f	f



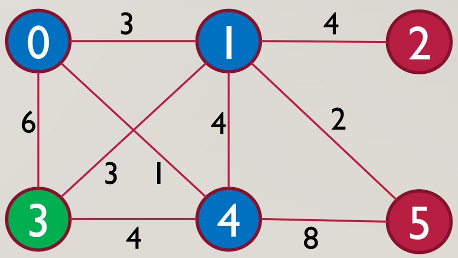
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	6	3	-1	0	-1	-1

Parent	0	1	2	3	4	5
	3	3	-1	3	-1	-1

Marked	0	1	2	3	4	5
	f	f	f	t	f	f



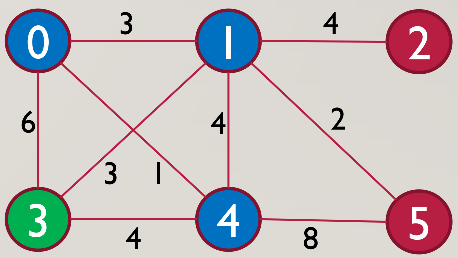
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	6	3	-1	0	4	-1

Parent	0	1	2	3	4	5
	3	3	-1	3	3	-1

Marked	0	1	2	3	4	5
	f	f	f	t	f	f



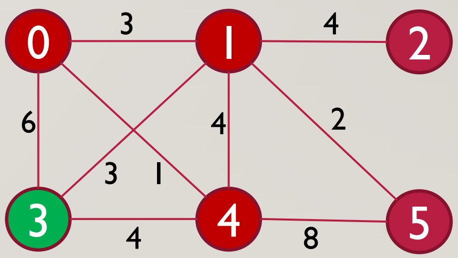
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	6	3	-1	0	4	-1

Parent	0	1	2	3	4	5
	3	3	-1	3	3	-1

Marked	0	1	2	3	4	5
	f	f	f	t	f	f



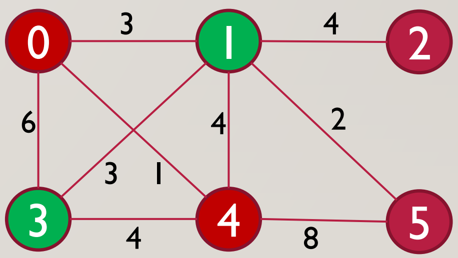
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	**0**	**1**	**2**	**3**	**4**	**5**
	6	3	-1	0	4	-1

Parent	**0**	**1**	**2**	**3**	**4**	**5**
	3	3	-1	3	3	-1

Marked	**0**	**1**	**2**	**3**	**4**	**5**
	f	t	f	t	f	f



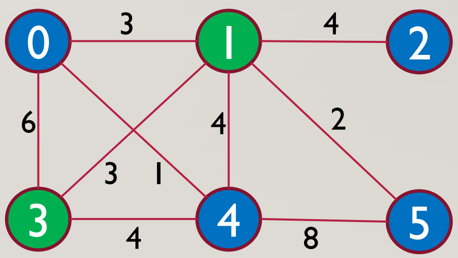
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	6	3	-1	0	4	-1

Parent	0	1	2	3	4	5
	3	3	-1	3	3	-1

Marked	0	1	2	3	4	5
	f	t	f	t	f	f



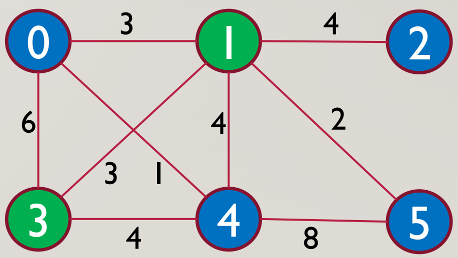
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	6	3	7	0	4	-1

Parent	0	1	2	3	4	5
	3	3	1	3	3	-1

Marked	0	1	2	3	4	5
	f	t	f	t	f	f



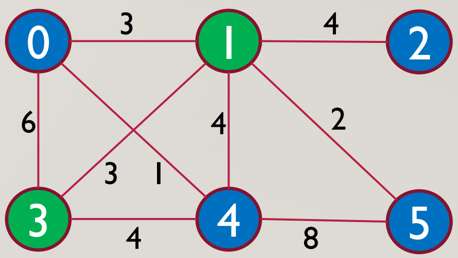
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

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

Parent	0	1	2	3	4	5
	3	3	1	3	3	1

Marked	0	1	2	3	4	5
	f	t	f	t	f	f



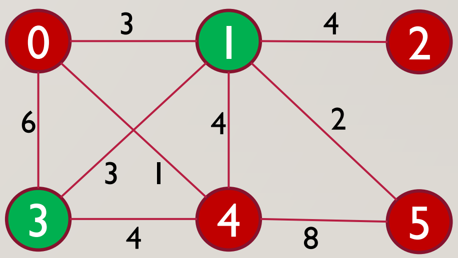
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

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

Parent	0	1	2	3	4	5
	3	3	1	3	3	1

Marked	0	1	2	3	4	5
	f	t	f	t	f	f



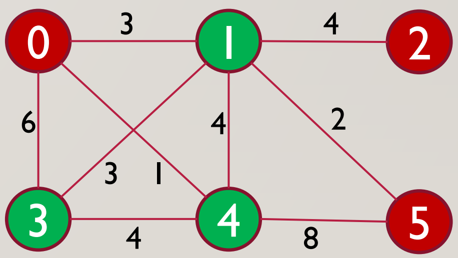
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

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

Parent	0	1	2	3	4	5
	3	3	1	3	3	1

Marked	0	1	2	3	4	5
	f	t	f	t	t	f



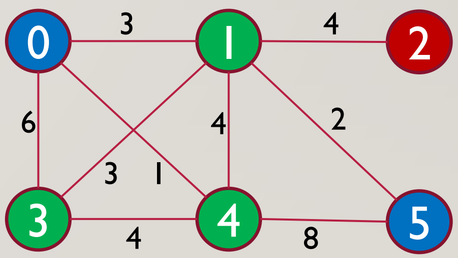
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

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

Parent	0	1	2	3	4	5
	3	3	1	3	3	1

Marked	0	1	2	3	4	5
	f	t	f	t	t	f



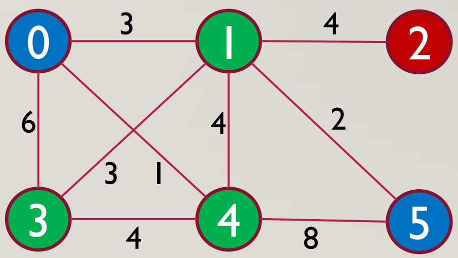
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	f	t	f	t	t	f



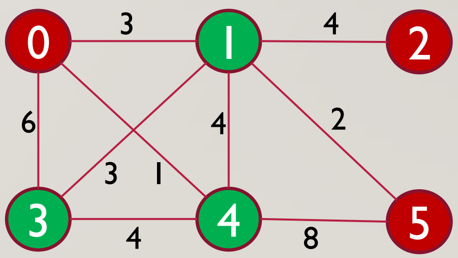
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	f	t	f	t	t	f



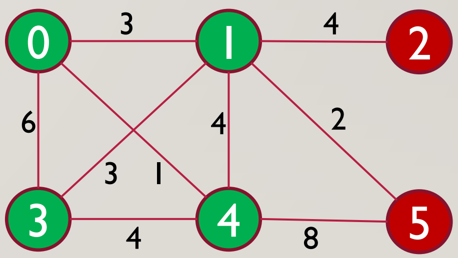
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	f	t	t	f



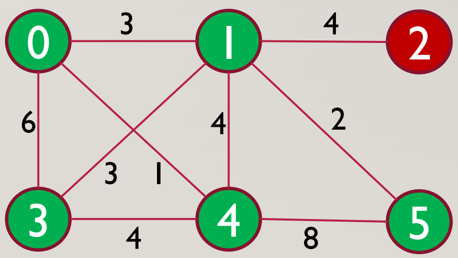
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	f	t	t	t



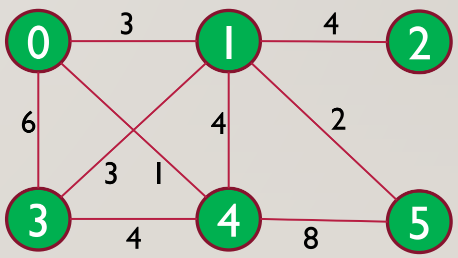
DIJKSTRA'S ALGORITHM

- parent[root]=root, cost[root]=0
- Set cost[v]=cost(root,v) for all v
- Repeatedly
  - Find the unmarked vertex **y** with the lowest cost
  - Set marked[y]=true
  - For every unmarked neighbour **z**
    - The current cost of getting to z is in the cost array
    - By finding the edge (y,z) we've found another possible cost. In this case cost(z)=cost(y)+cost(y,z)
    - Update the cost and parent of z if it's lower than the current known cost or the cost is unknown
  - Stop when all vertices are in the tree

Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	t	t	t	t

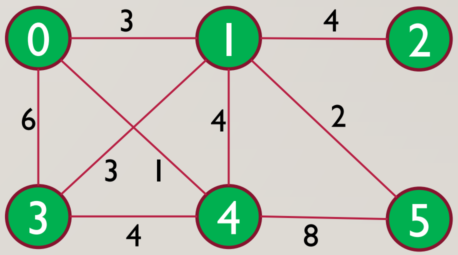


DIJKSTRA'S ALGORITHM

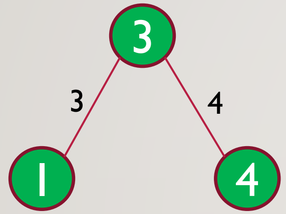
Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	t	t	t	t



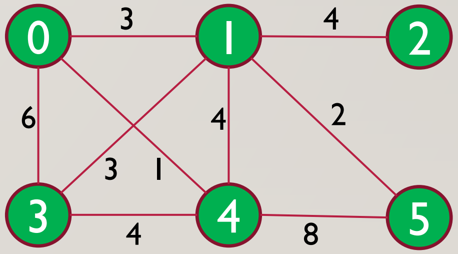
DIJKSTRA'S ALGORITHM



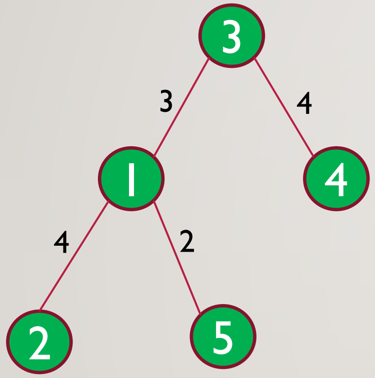
Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	t	t	t	t



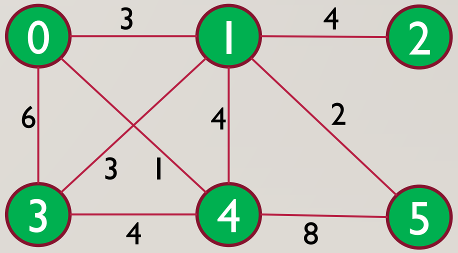
DIJKSTRA'S ALGORITHM



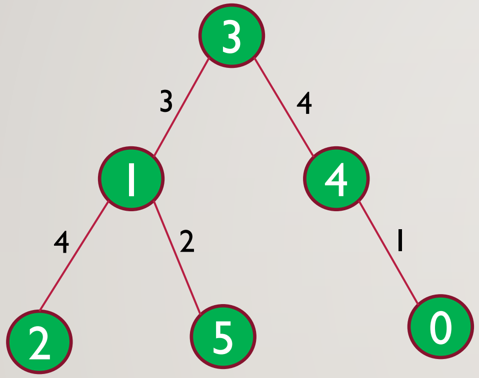
Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	t	t	t	t



DIJKSTRA'S ALGORITHM



Cost	0	1	2	3	4	5
	5	3	7	0	4	5

Parent	0	1	2	3	4	5
	4	3	1	3	3	1

Marked	0	1	2	3	4	5
	t	t	t	t	t	t