CMSC 132:
Object-Oriented Programming II

Single Source Shortest Path Algorithm

Department of Computer Science

University of Maryland, College Park

Single Source Shortest Path

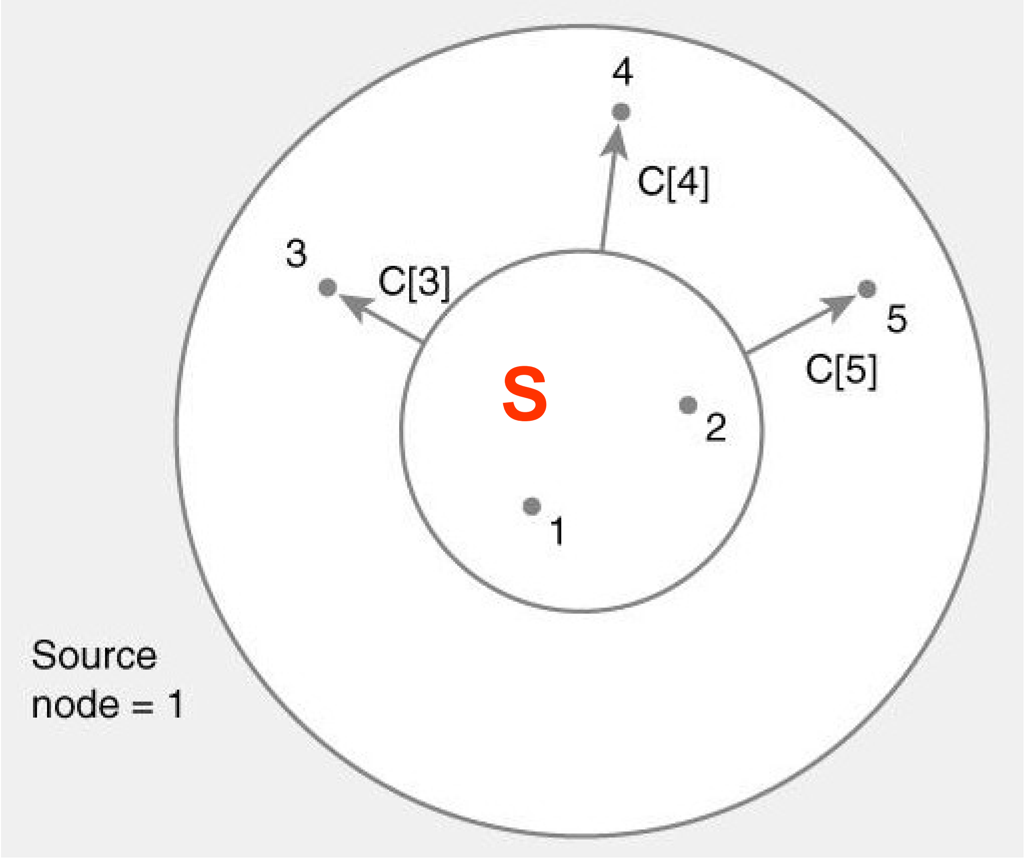
- **Common graph problem**
  1. Find path from X to Y with lowest edge weight
  2. Find path from X to any Y with lowest edge weight
- **Useful for many applications**
  - Shortest route in map
  - Lowest cost trip
  - Most efficient internet route
- **Dijkstra's algorithm solves problem 2**
  - Can also be used to solve problem 1
  - Would use different algorithm if only interested in a single destination

Shortest Path – Dijkstra’s Algorithm

- Maintain
  - Nodes with known shortest path from start ≅ S
  - Cost of shortest path to node K from start ≅ C[K]
    - Only for paths through nodes in S
  - Predecessor to K on shortest path ≅ P[K]
    - Updated whenever new (lower) C[K] discovered
    - Remembers actual path with lowest cost

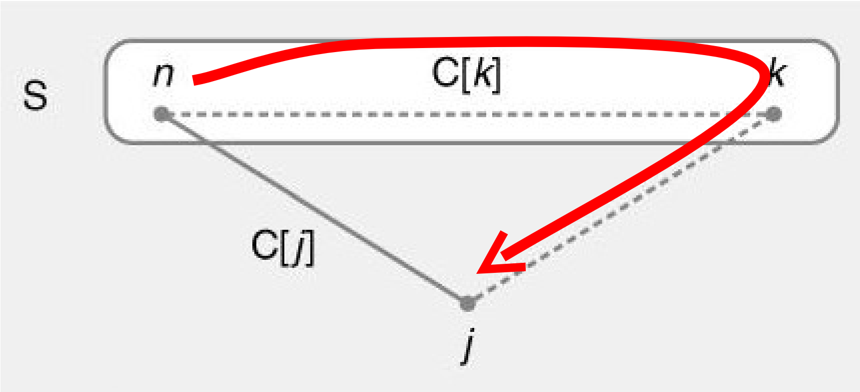
Shortest Path – Intuition for Dijkstra's

- At each step in the algorithm
  - Shortest paths are known for nodes in S
  - Store in C[K] length of shortest path to node K (for all paths through nodes in { S })
  - Add to { S } next closest node



Shortest Path – Intuition for Djikstra's

- Update distance to J after adding node K
  - Previous shortest path to K already in C[K]
  - Possibly shorter path to J by going through node K
  - Compare C[J] with C[K] + weight of (K,J), update C[J] if needed



Shortest Path – Dijkstra's Algorithm

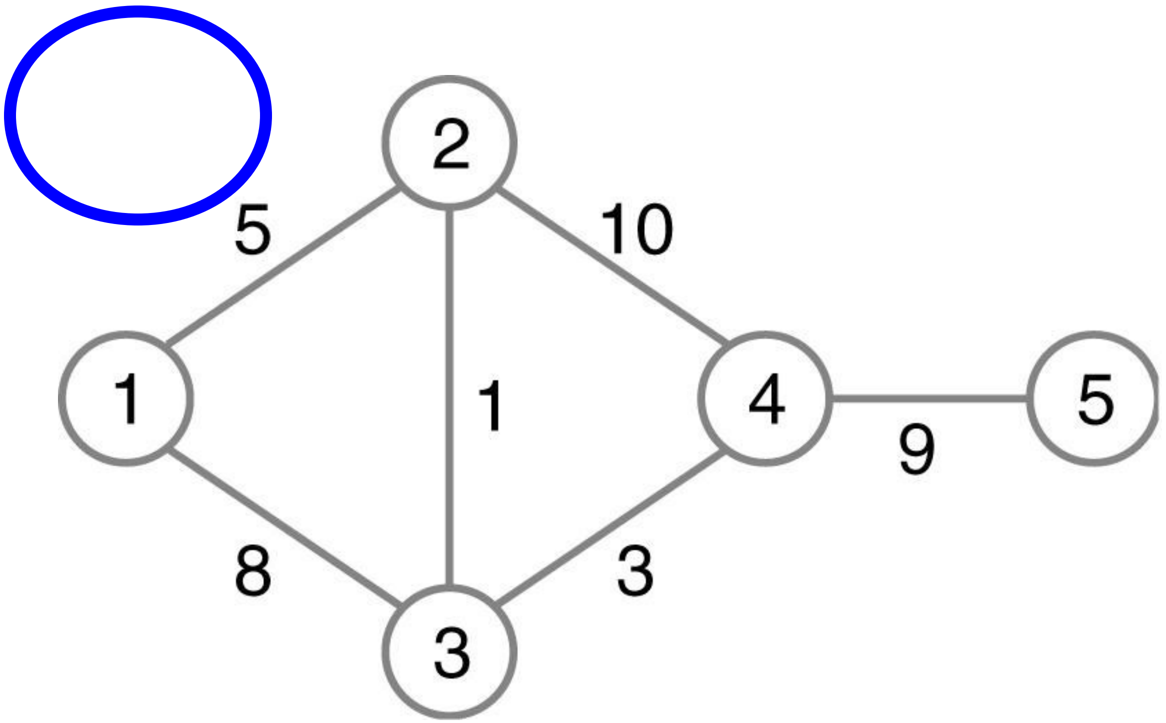
```
S = ≅
P[ ] = none for all nodes
C[start] = 0, C[ ] = ≅   for all other nodes
while ( not all nodes in S )
    find node K not in S with smallest C[K]
    add K to S
    for each node J not in S  adjacent to K
        if ( C[K] + cost of (K,J) < C[J] )
            C[J] = C[K] + cost of (K,J)
            P[J] = K
```

Optimal solution computed with greedy algorithm

Dijkstra's Shortest Path Example

- **Initial state**
- **S = ≅**

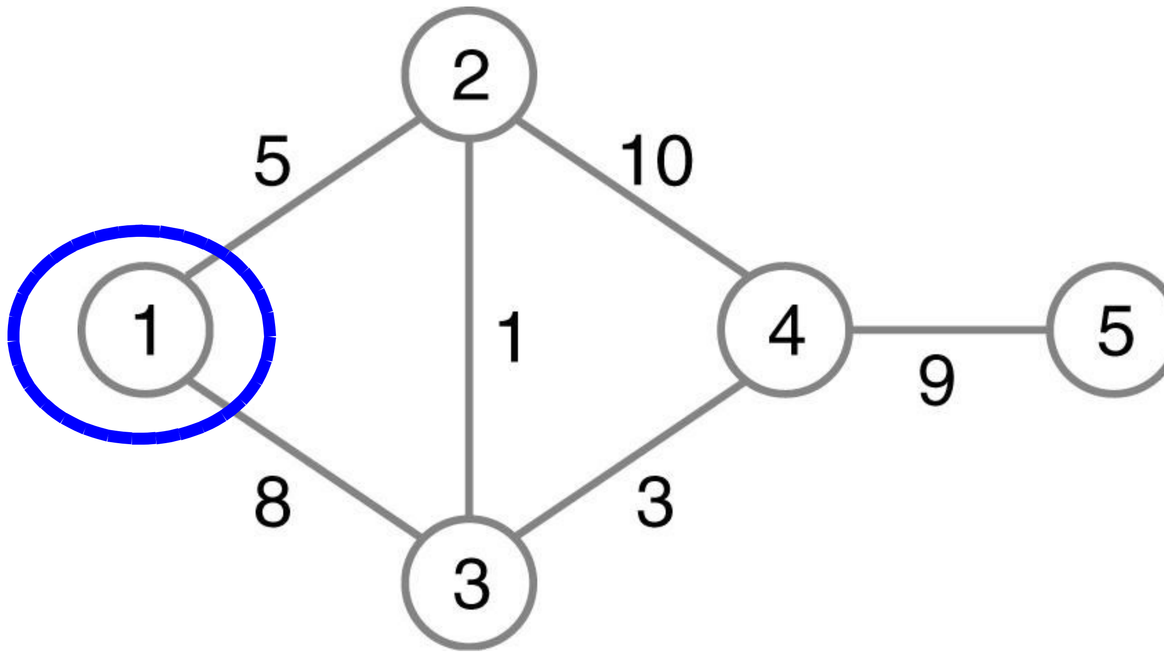
	C	P
1	0	none
2	≅	none
3	≅	none
4	≅	none
5	≅	none



Dijkstra’s Shortest Path Example

- **Find shortest paths starting from node 1**
- **S = 1**

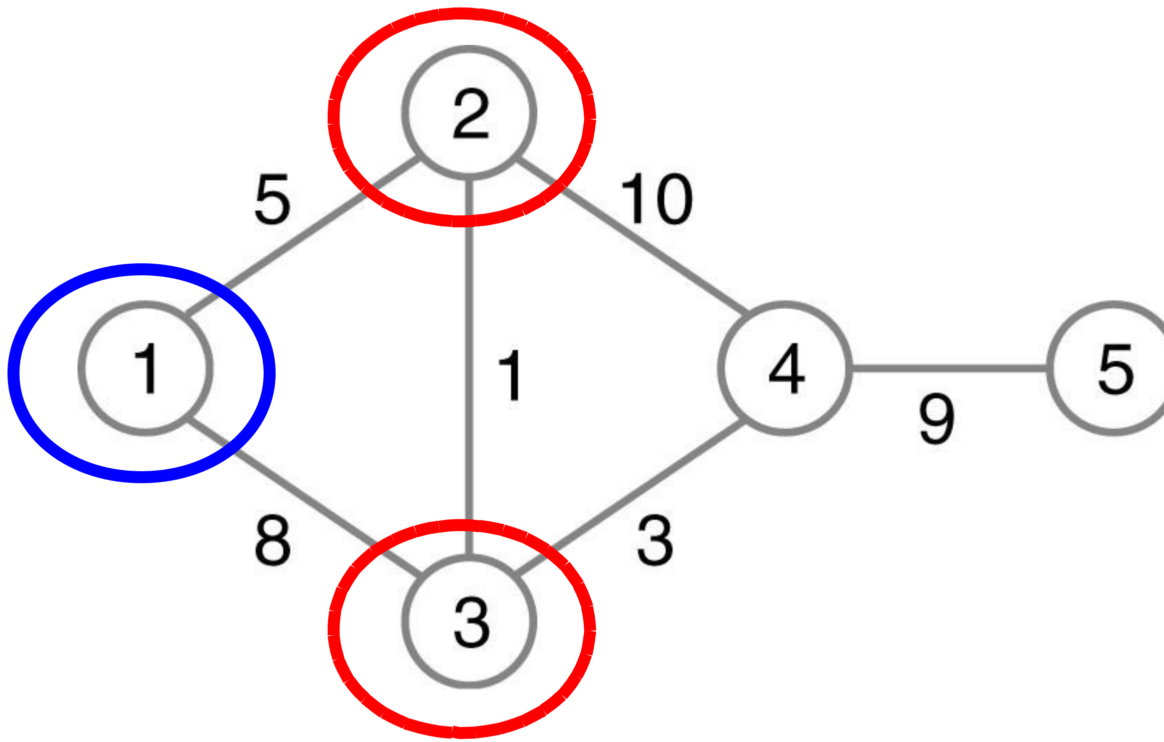
	C	P
1	0	none
2	≅	none
3	≅	none
4	≅	none
5	≅	none



Djikstra's Shortest Path Example

- **Update C[K] for all neighbors of 1 not in { S }**
- **S = { 1 }**

	C	P
1	0	none
2	5	1
3	8	1
4	≅	none
5	≅	none

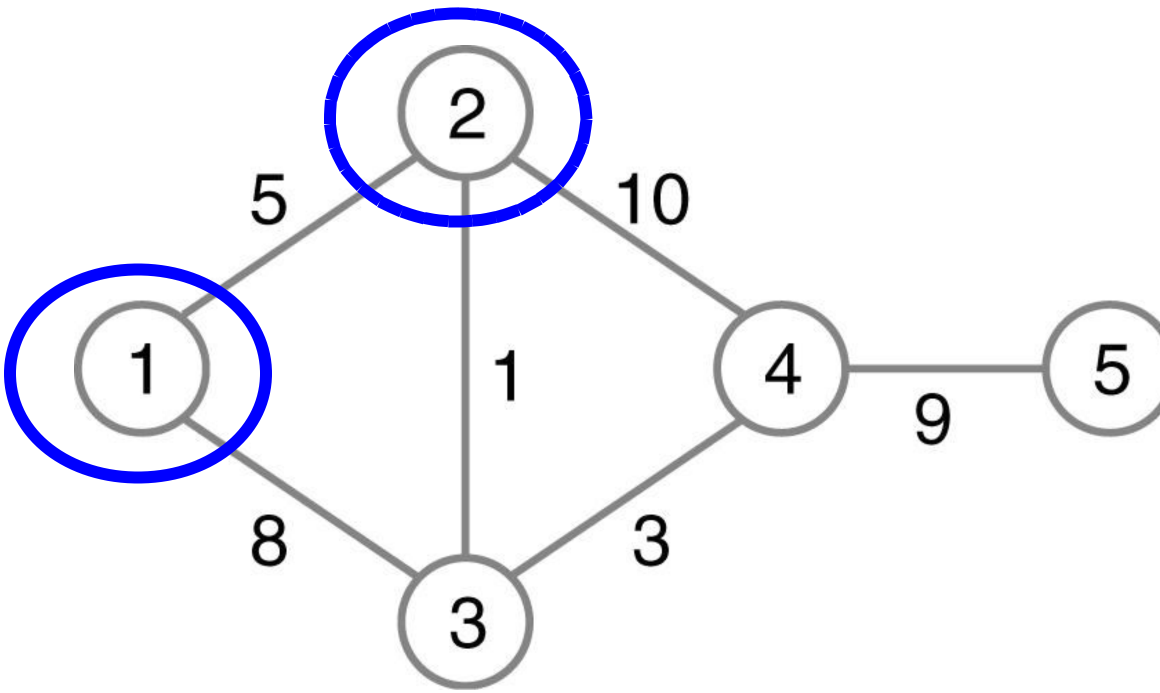


C[2] = min (≅ , C[1] + (1,2)) = min (≅ , 0 + 5) = 5
C[3] = min (≅ , C[1] + (1,3)) = min (≅ , 0 + 8) = 8

Djikstra's Shortest Path Example

- **Find node K with smallest C[K] and add to S**
- **S = { 1, 2 }**

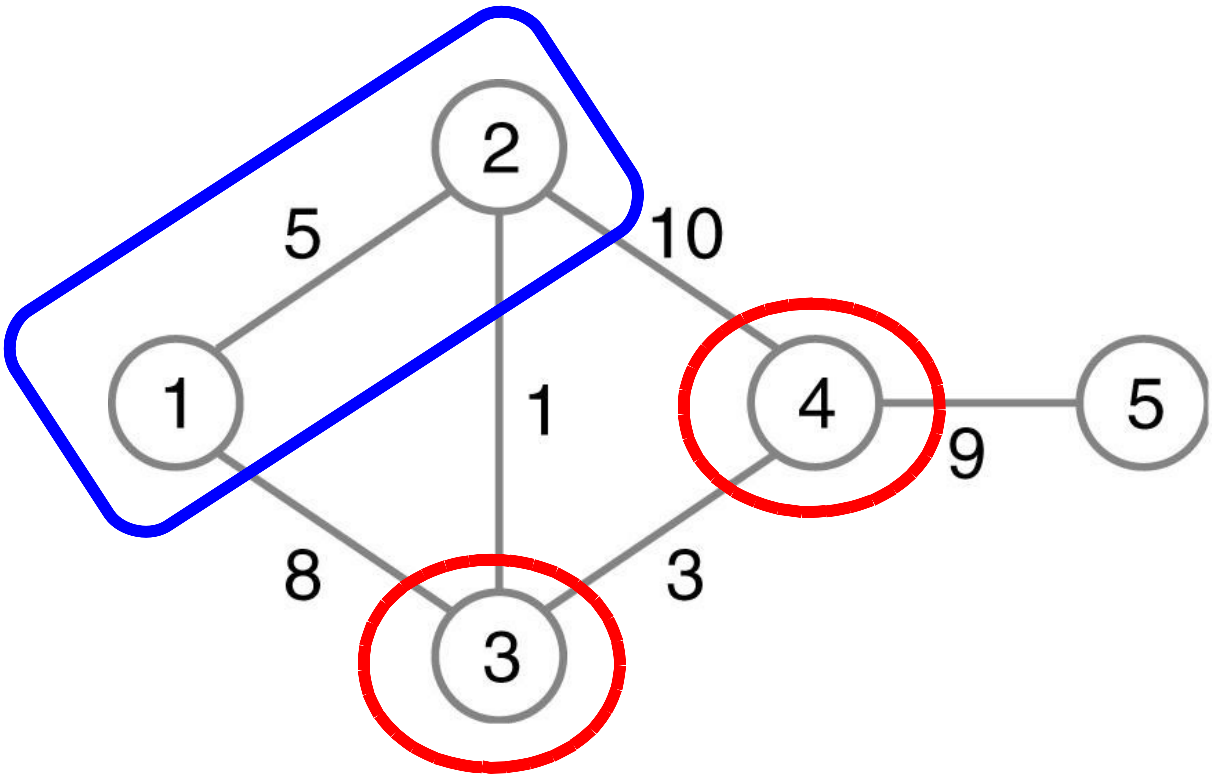
	C	P
1	0	none
2	5	1
3	8	1
4	≅	none
5	≅	none



Dijkstra's Shortest Path Example

- Update C[K] for all neighbors of 2 not in S
- S = { 1, 2 }

	C	P
1	0	none
2	5	1
3	6	2
4	15	2
5	≅	none

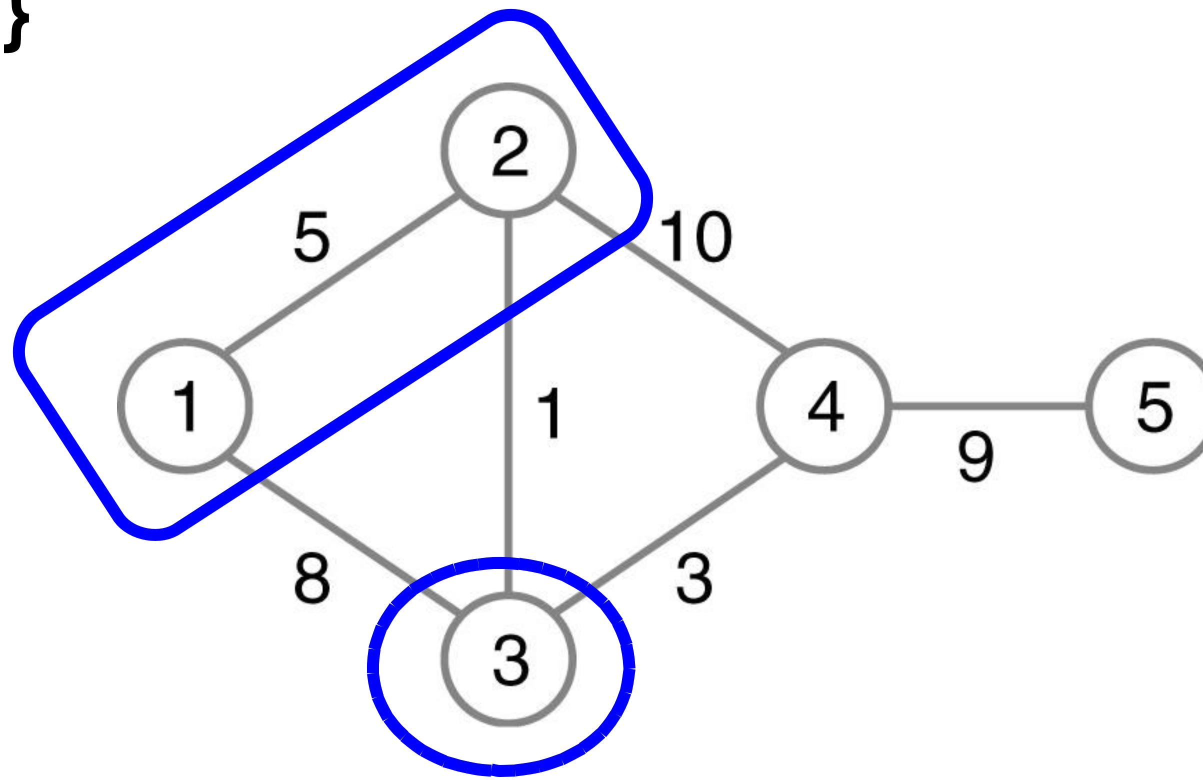


C[3] = min (8 , C[2] + (2,3)) = min (8 , 5 + 1) = 6
C[4] = min (≅ , C[2] + (2,4)) = min (≅ , 5 + 10) = 15

Dijkstra's Shortest Path Example

- Find node K with smallest C[K] and add to S
- S = { 1, 2, 3 }

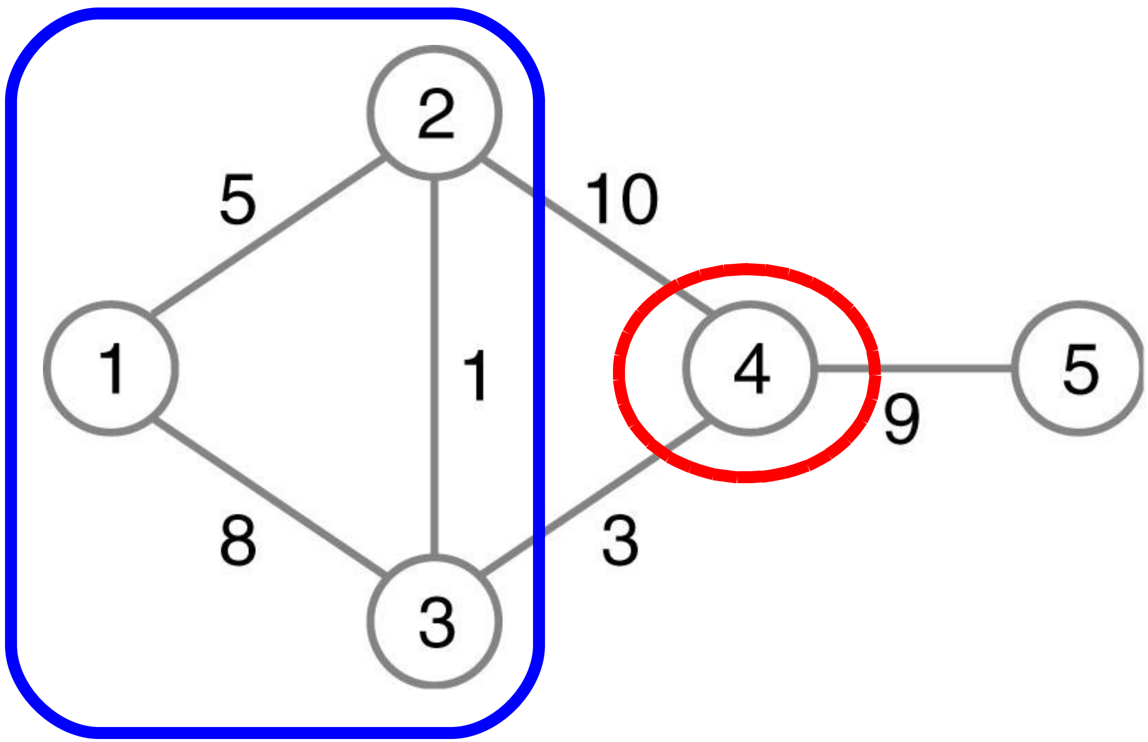
	C	P
1	0	none
2	5	1
3	6	2
4	15	2
5	≅	none



Dijkstra's Shortest Path Example

- **Update C[K] for all neighbors of 3 not in S**
- **{ S } = 1, 2, 3**

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	≅	none

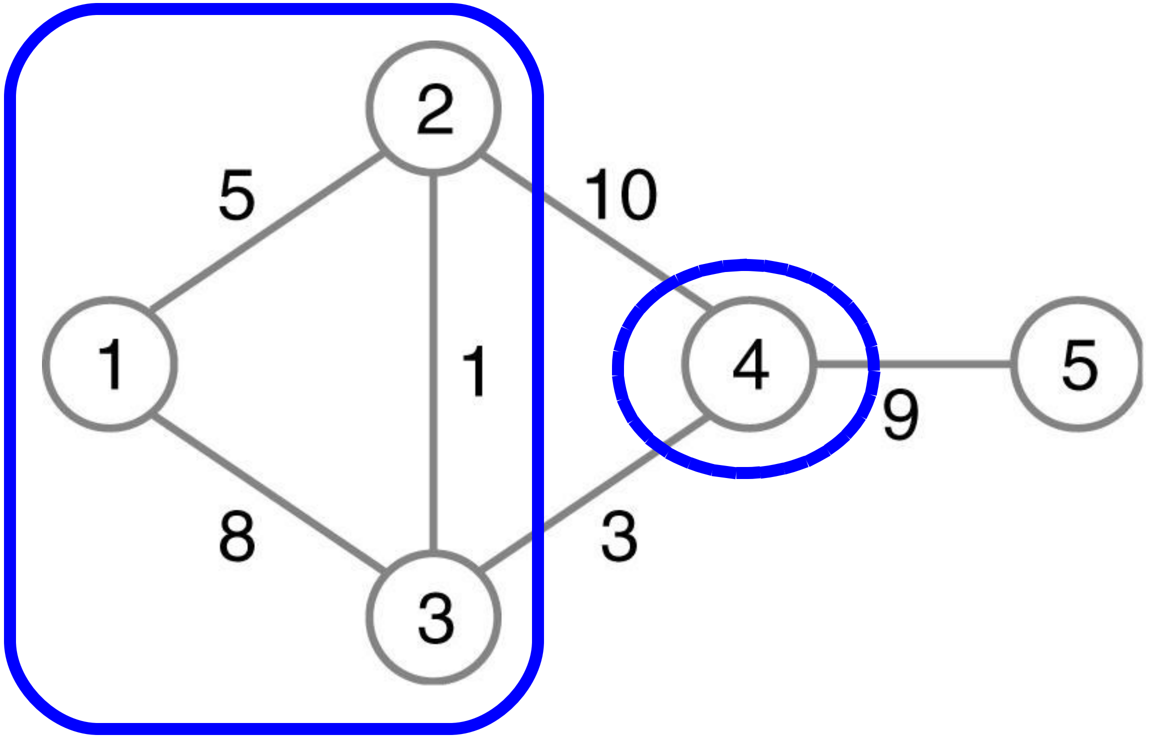


C[4] = min (15 , C[3] + (3,4)) = min (15 , 6 + 3) = 9

Dijkstra's Shortest Path Example

- **Find node K with smallest C[K] and add to S**
- **{ S } = 1, 2, 3, 4**

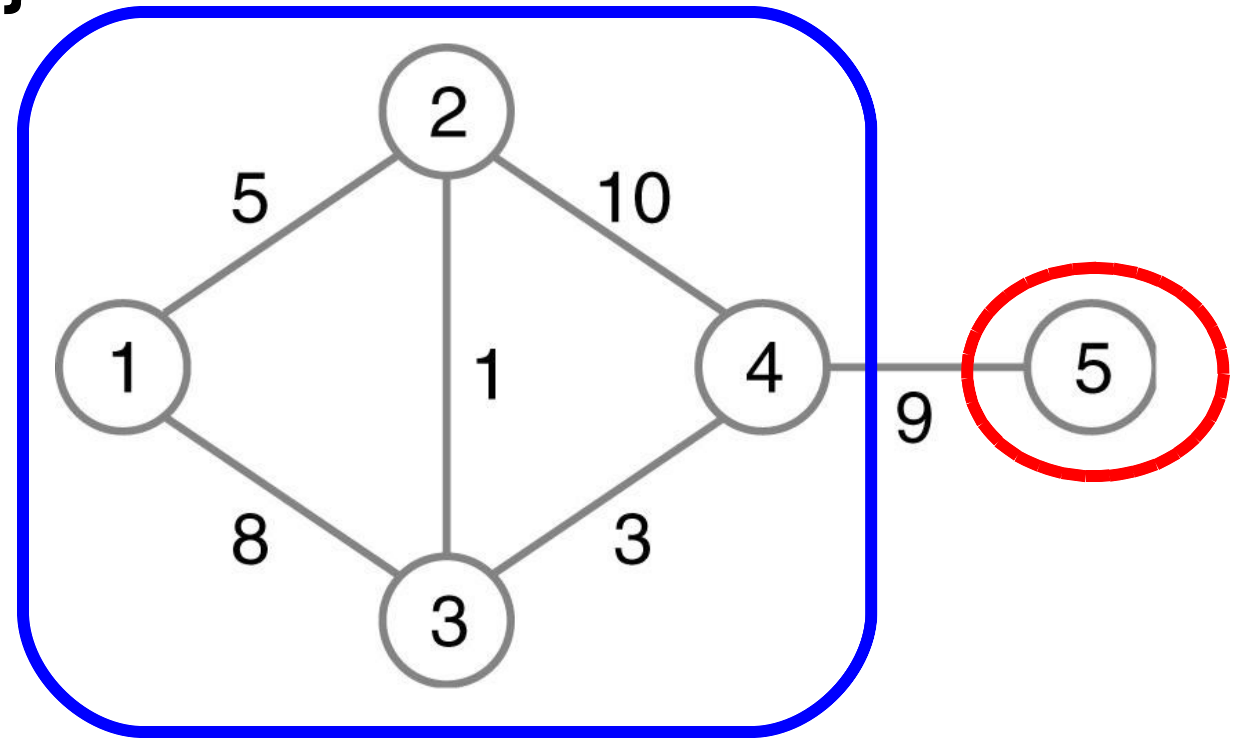
	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	≅	none



Dijkstra's Shortest Path Example

- Update C[K] for all neighbors of 4 not in S
- S = { 1, 2, 3, 4 }

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4

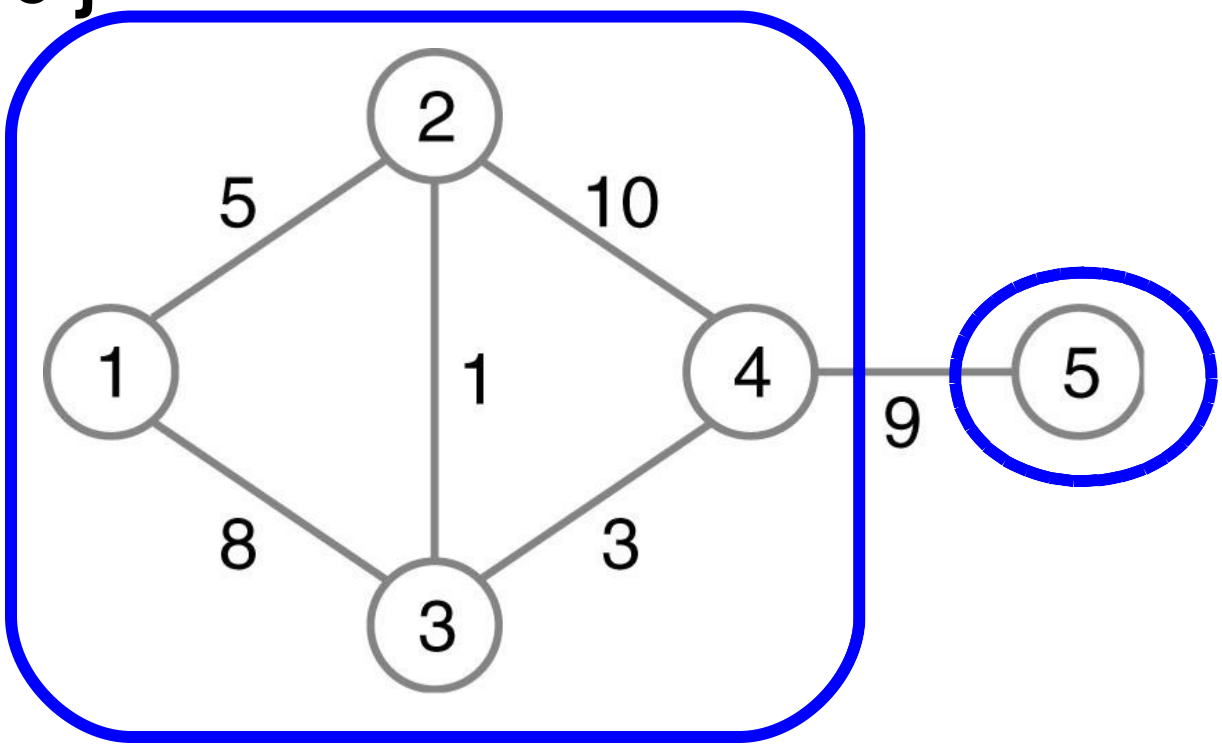


C[5] = min (≅ , C[4] + (4,5)) = min (≅ , 9 + 9) = 18

Dijkstra's Shortest Path Example

- Find node K with smallest C[K] and add to S
- S = { 1, 2, 3, 4, 5 }

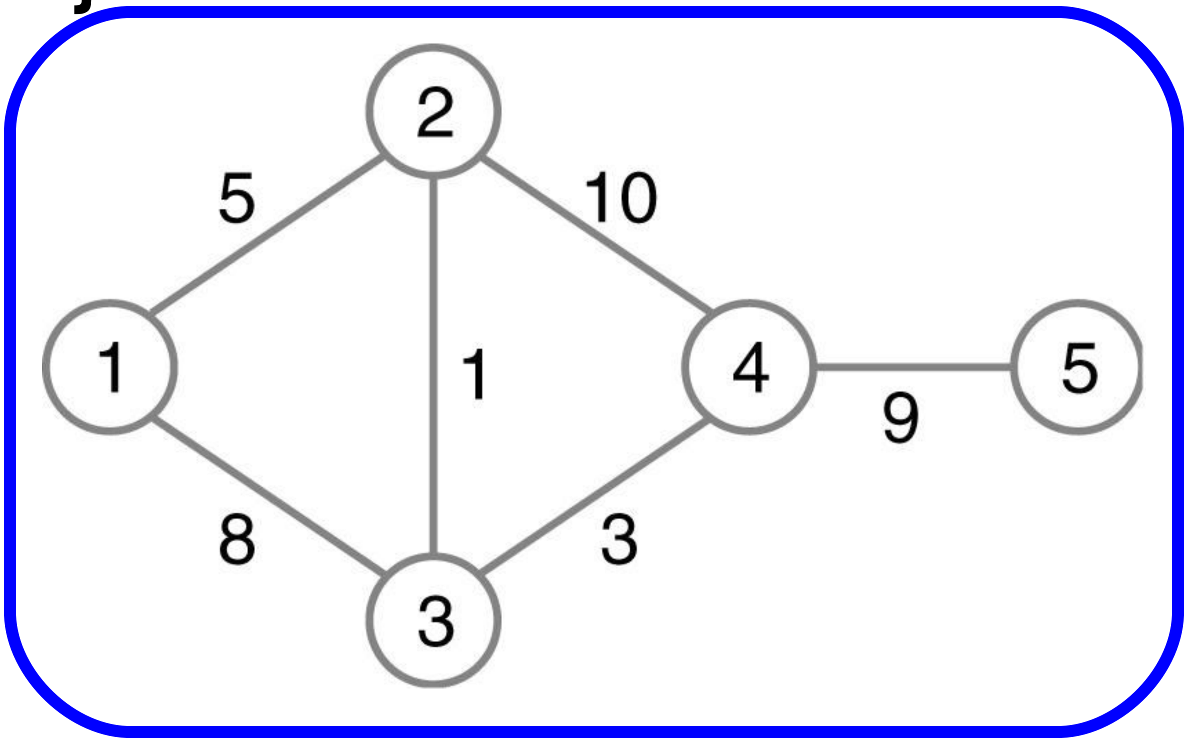
	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



Dijkstra's Shortest Path Example

- **All nodes in S, algorithm is finished**
- **S = { 1, 2, 3, 4, 5 }**

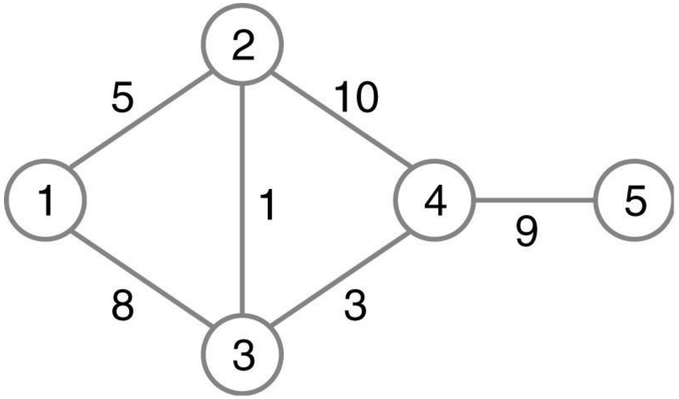
	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



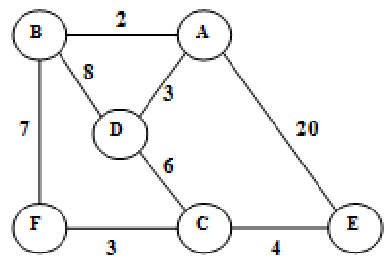
Dijkstra's Shortest Path Example

- **Find shortest path from start to K**
  - Start at K
  - Trace back predecessors in P[ ]
- **Example paths (in reverse)**
  - 2 ≅ 1
  - 3 ≅ 2 ≅ 1
  - 4 ≅ 3 ≅ 2 ≅ 1
  - 5 ≅ 4 ≅ 3 ≅ 2 ≅ 1

	C	P
1	0	none
2	5	1
3	6	2
4	9	3
5	18	4



Typical Problem for Exam/Quiz



Apply Dijkstra's algorithm using B as the starting (source) node. Indicate the cost and predecessor for each node in the graph after processing 1, 2 and 3 nodes (B and 2 other nodes) have been added to the set of processed nodes (Remember to update the appropriate table entries after processing the 3rd node added). An empty table entry implies an infinite cost or no predecessor. *Note: points will be deducted if you simply fill in the entire table instead showing the table at the first three steps.*

Answer:

After processing 1 node:

Node	A	B	C	D	E	F
Cost	2	0		8		7
Predecessor	B			B		B

After processing 2 nodes:

Node	A	B	C	D	E	F
Cost	2	0		5	22	7
Predecessor	B			A	A	B

After processing 3 nodes:

Node	A	B	C	D	E	F
Cost	2	0	11	5	22	7
Predecessor	B		D	A	A	B