

04-630

# Data Structures and Algorithms for Engineers

David Vernon  
Carnegie Mellon University Africa

[vernon@cmu.edu](mailto:vernon@cmu.edu)  
[www.vernon.eu](http://www.vernon.eu)

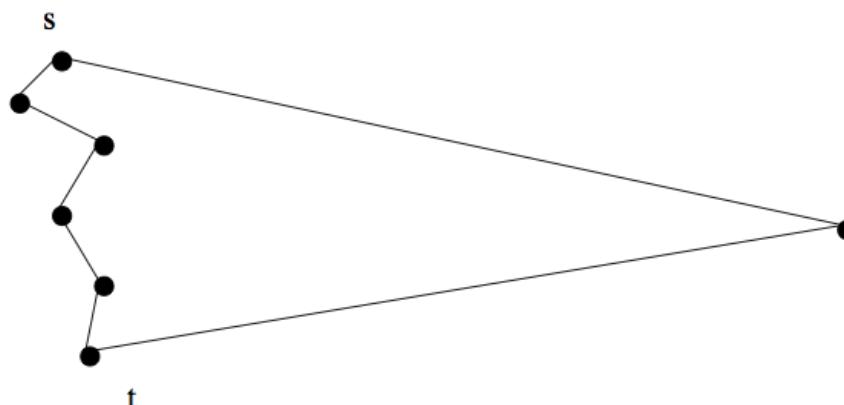
# Lecture 22

## Graphs

- Types of graph
- Adjacency matrix representation
- Adjacency list representation
- Breadth-First Search traversal
- Depth-First Search traversal
- Topological Sorting
- Minimum Spanning Tree
  - Prim's Algorithm
  - Kruskal's algorithm
- Shortest Path Algorithms
  - Dijkstra's algorithm
  - Floyd's algorithm

# Shortest Paths

- A *path* is a sequence of edges connecting two vertices
- The shortest path in an unweighted graph can be constructed using Breadth-First Search
- Also works if the weights are all the same in a weighted graph
- Does not work for weighted graphs in general:  
the shortest route (e.g. in time) may pass through many intermediate vertices



# Shortest Paths

- Two main algorithms for finding shortest paths
  - Dijkstra's Algorithm
    - Finds shortest path between **start and destination** vertices
      - Some implementations find the shortest path between a start vertex and all other vertices, i.e. **shortest path spanning tree rooted in the start vertex**
    - $O(n^2)$  for with simple data structures
  - Floyd's Algorithm
    - Finds shortest path between **all pairs** of vertices in a graphs
    - $O(n^3)$

# Shortest Paths

- Dijkstra's Algorithm
  - Greedy algorithm
  - Repeatedly select the smallest weight edge that will extend the path
    - Start from given vertex,
    - Extend the path, one edge at a time,
    - Until all vertices are included
  - Very similar to Prim's algorithm, except ...
    - Instead of just considering the weight of the next potential edge
    - **We also have to consider the distance from the start to the vertex from which that edge emanates**

# Shortest Paths

ShortestPath-Dijkstra( $G, s, t$ )

```
path= {s}
for i = 1 to n, dist[i] = ∞
for each edge (s, v), dist[v] = w(s, v) // initial distances are just the weights
last = s
while (last != t)
    select  $v_{next}$ , the unknown vertex minimizing dist[v]
    for each edge ( $v_{next}, x$ ), dist[x] = min[dist[x], dist[vnext] + w( $v_{next}, x$ )]
    last =  $v_{next}$ 
path = pathU { $v_{next}$ }
```

# Shortest Paths

```
/* Dijkstra's algorithm: implementation based of Prim's algorithm */

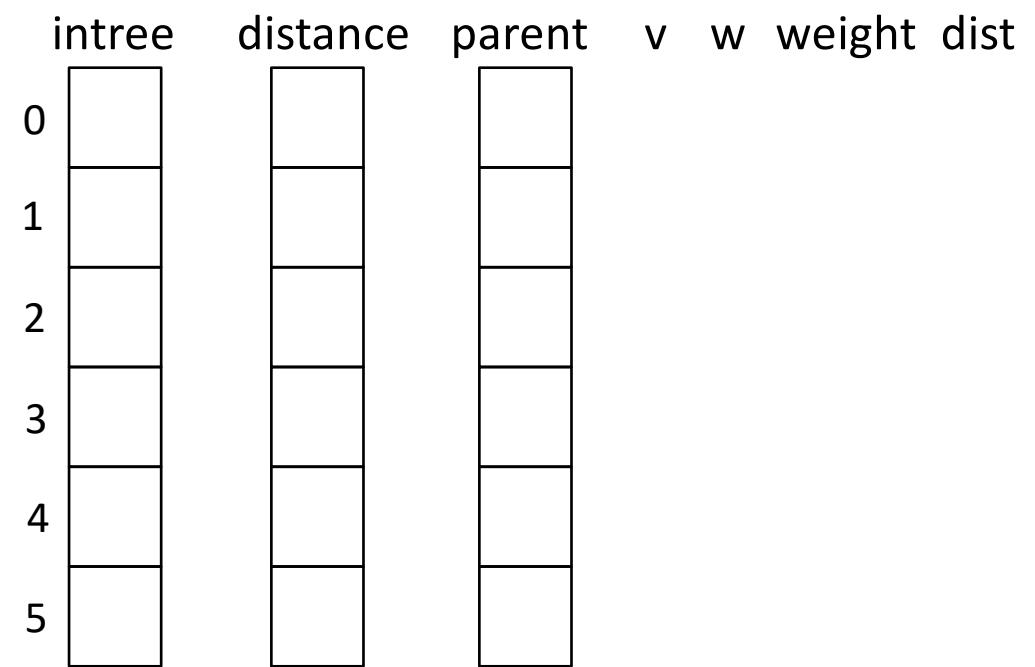
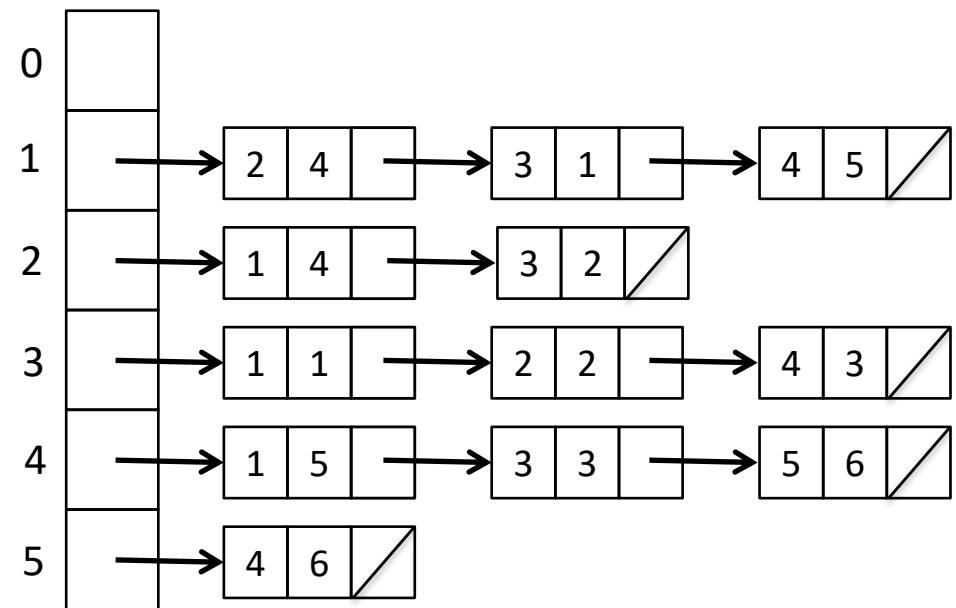
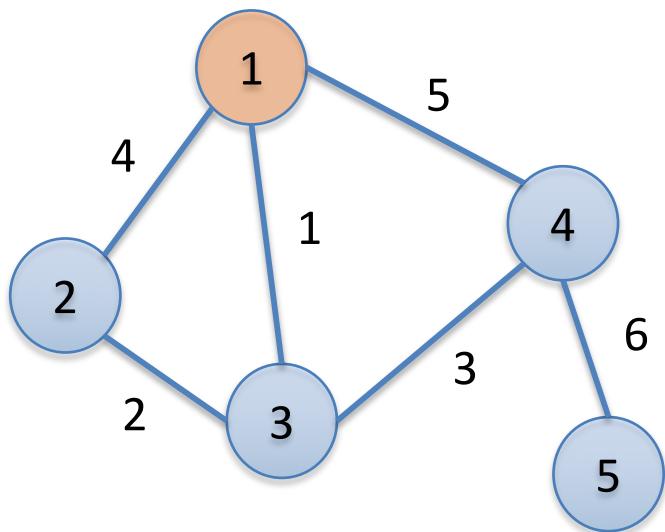
dijkstra(graph *g, int start) {
    int i;                      /* counter */ *
    edgenode *p;                /* temporary pointer */ *
    bool intree[MAXV+1];        /* is the vertex in the tree yet? */ *
    int distance[MAXV+1];       /* cost of adding to tree */ *
    int parent[MAXV+1];         /* parent vertex */ *
    int v;                      /* current vertex to process */ *
    int w;                      /* candidate next vertex */ *
    int weight;                 /* edge weight */ *
    int dist;                   /* best current distance from start */ *

    for (i=1; i<=g->nvertices; i++) {
        intree[i] = FALSE;
        distance[i] = MAXINT;
        parent[i] = -1;
    }

    distance[start] = 0;
    v = start;
```

# Shortest Paths

```
while (intree[v] == FALSE) {  
    intree[v] = TRUE;  
    p = g->edges[v];  
    while (p != NULL) {  
        w = p->y;  
        weight = p->weight;  
        if ((distance[v]+weight < distance[w])) { //changes from Prim  
            distance[w] = distance[v]+weight ;  
            parent[w] = v;  
        }  
        p = p->next;  
    }  
    v = 1;  
    dist = MAXINT;  
    for (i=1; i<=g->nvertices; i++)  
        if ((intree[i] == FALSE) && (distance[i] < dist)) {  
            dist = distance[i];  
            v = i;  
        }  
    }  
}
```



```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

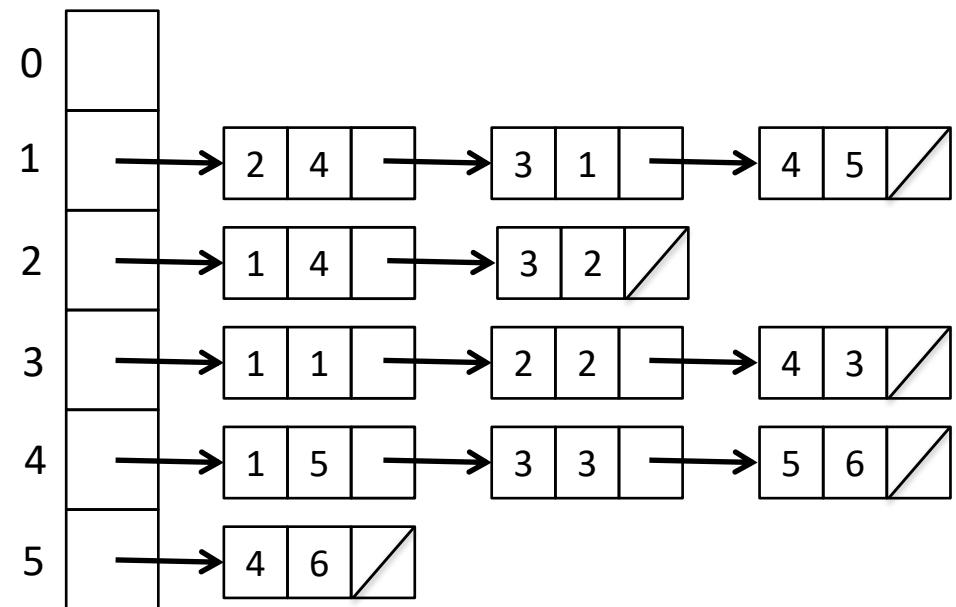
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							
1							
2							
3							
4							
5							

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

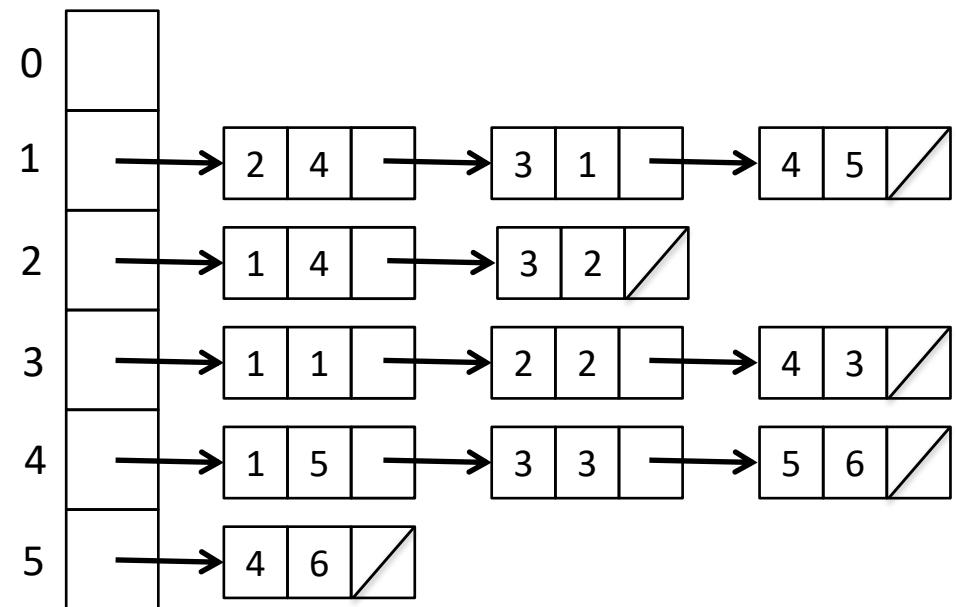
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							
1	F	$\infty$	-1				
2	F	$\infty$	-1				
3	F	$\infty$	-1				
4	F	$\infty$	-1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

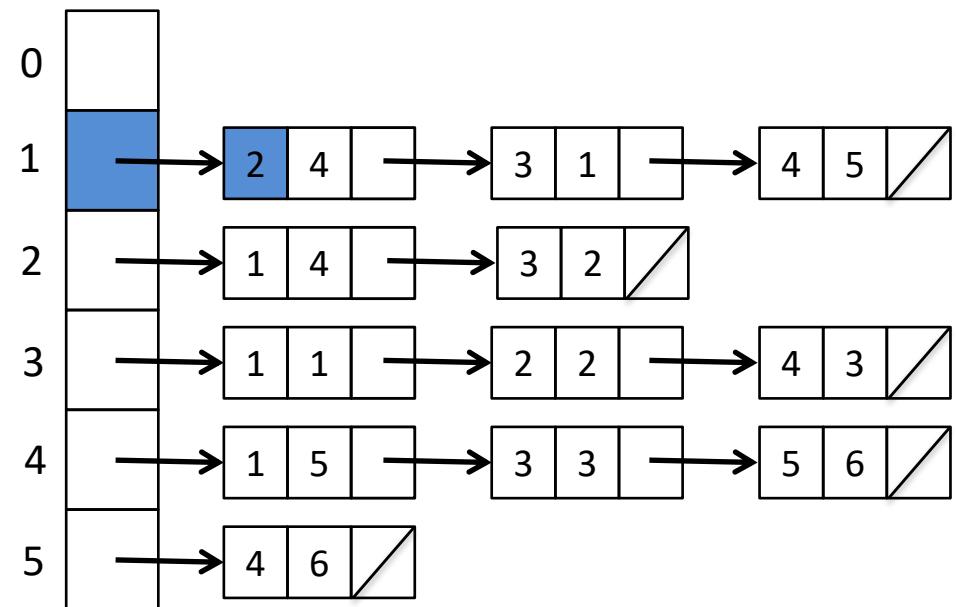
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							
1	T	0	-1	1	2	4	
2	F	$\infty$	-1				
3	F	$\infty$	-1				
4	F	$\infty$	-1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

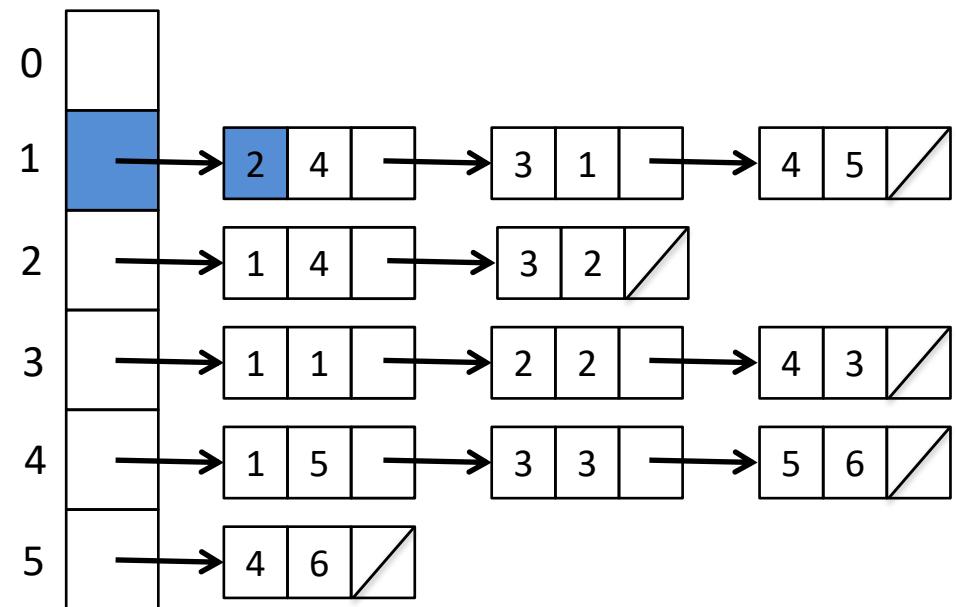
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							
1	T	0	-1	1	2	4	
2	F	4	1				
3	F	$\infty$	-1				
4	F	$\infty$	-1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

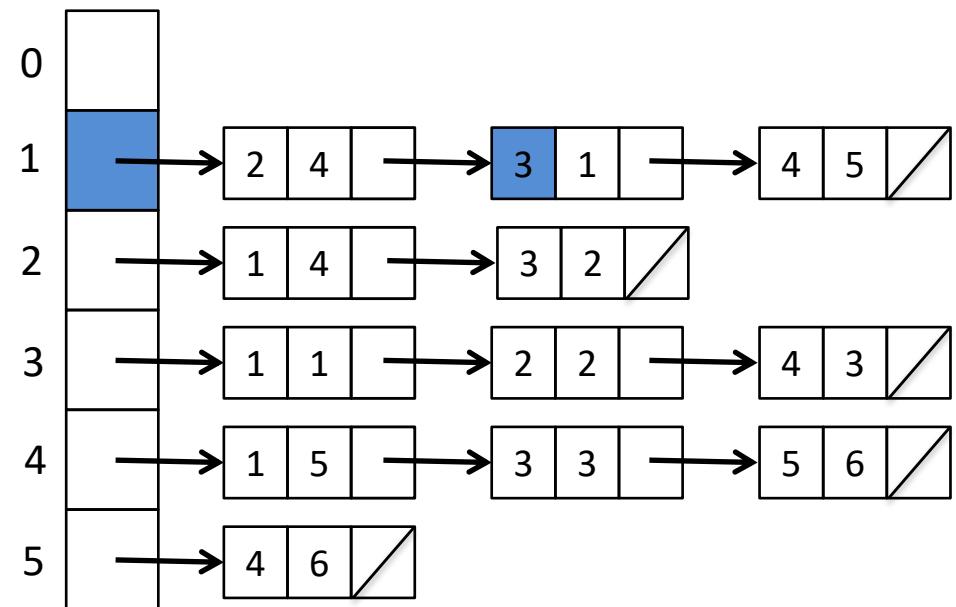
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							
1	T	0	-1	1	2	4	
2	F	4	1				1
3	F	1	1				
4	F	$\infty$	-1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

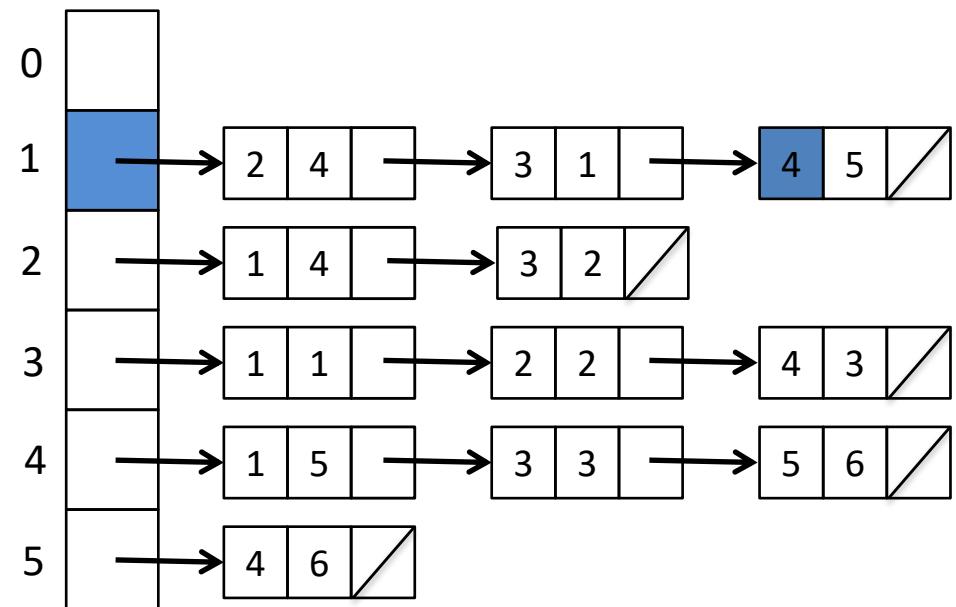
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0		0	-1	1	2	4	0
1	T	0	-1	1	3	1	1
2	F	4	1	1	1	1	1
3	F	1	1	1	1	1	1
4	F	5	1	1	1	1	1
5	F	∞	-1	1	1	1	1

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

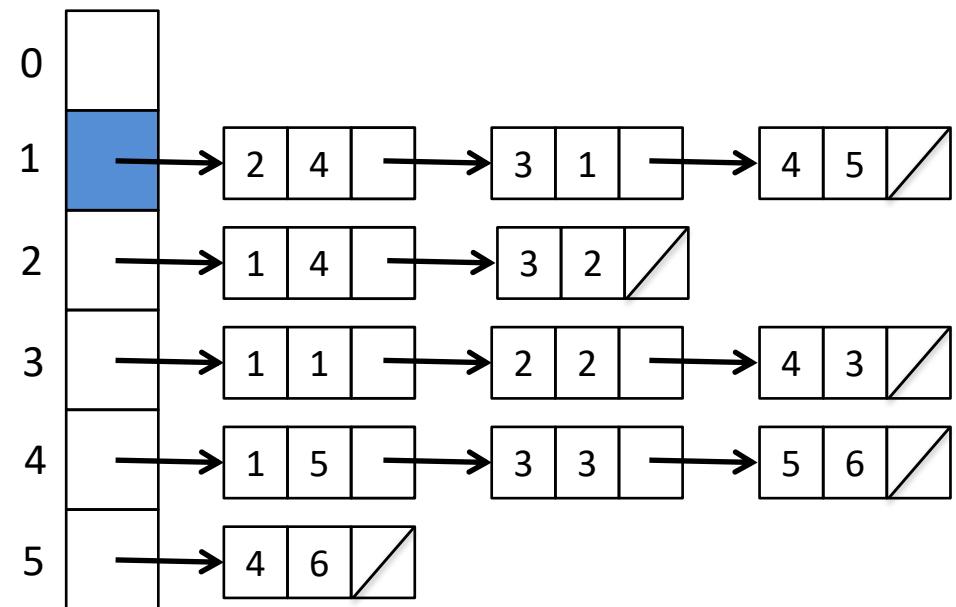
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	F	4	1	2	4	5	1
3	F	1	1				3
4	F	5	1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

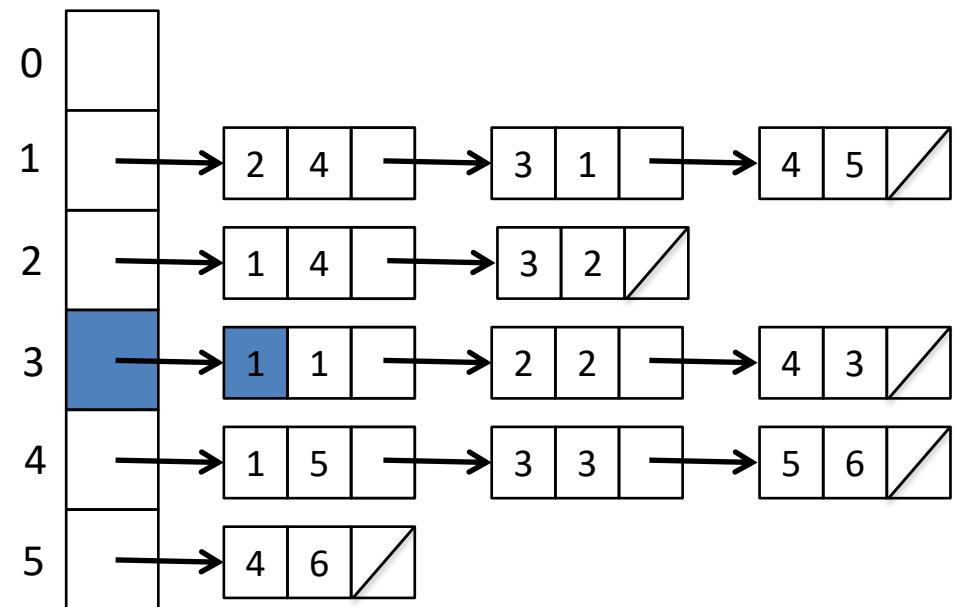
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	v	w	weight	dist
0	1	2	4	$\infty$
1	1	3	1	4
2	2	4	5	1
3	3	1	1	1
4				
5				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

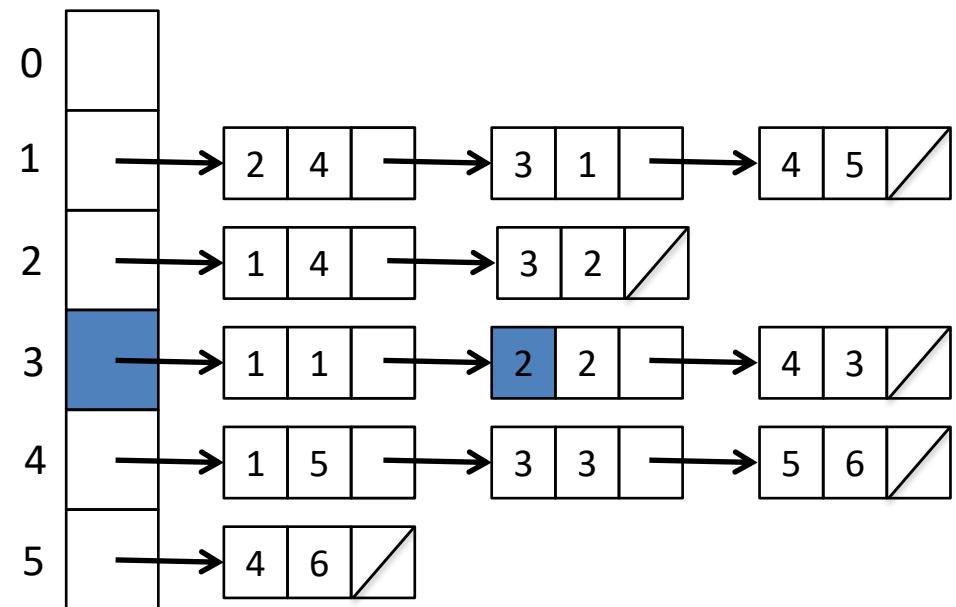
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	F	4	1	2	4	5	1
3	T	1	1	3	1	1	
4	F	5	1		2	2	
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

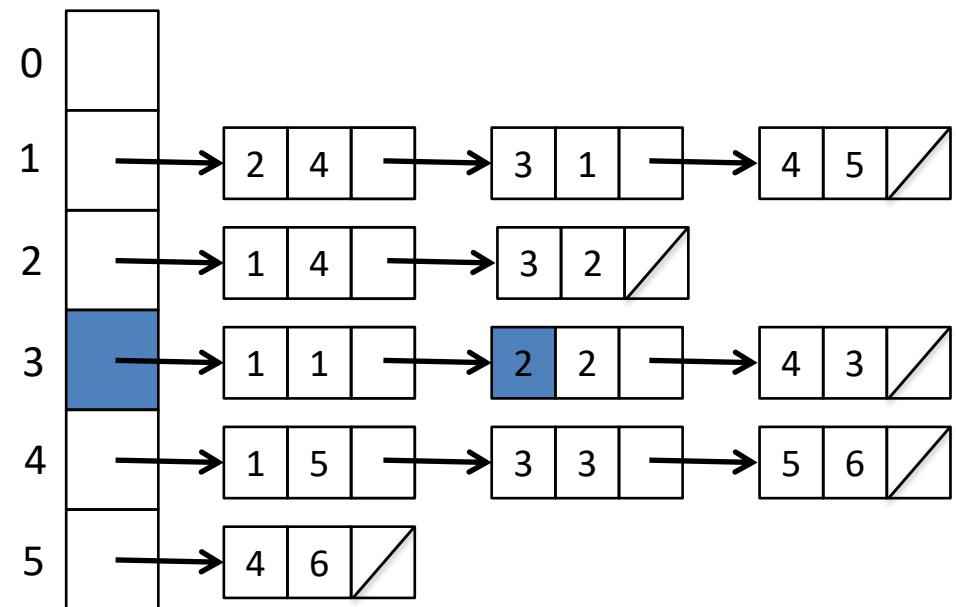
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	F	3	3	2	4	5	1
3	T	1	1	3	1	1	
4	F	5	1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

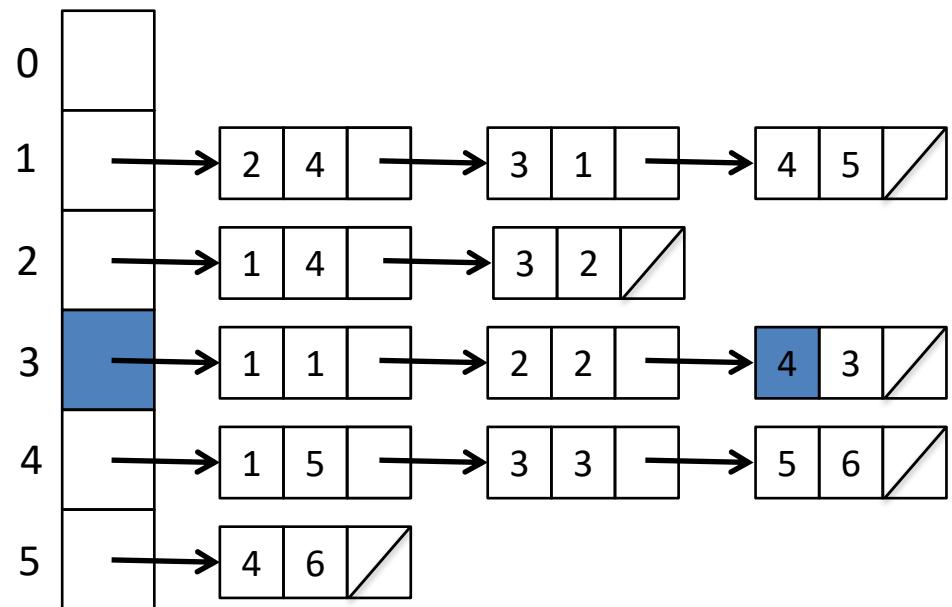
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])){
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							$\infty$
1	T	0	-1	1	2	4	4
2	F	3	3	1	3	1	1
3	T	1	1	2	4	5	2
4	F	5	1				
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

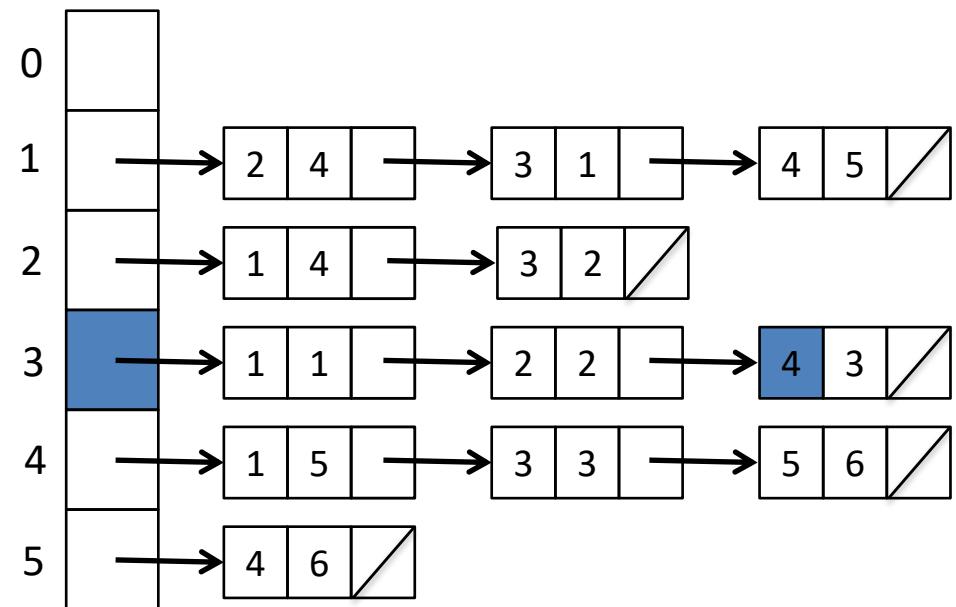
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							$\infty$
1	T	0	-1	1	2	4	4
2	F	3	3	1	3	1	1
3	T	1	1	2	4	5	2
4	F	5	1	3	1	1	3
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

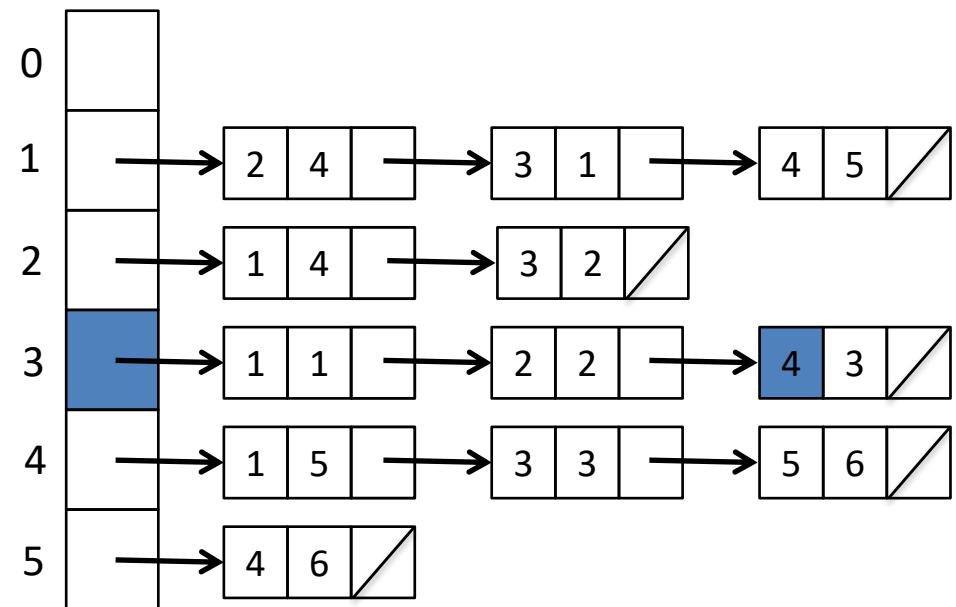
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	F	3	3	2	4	5	1
3	T	1	1	3	1	1	2
4	F	4	3	4		3	
5	F	$\infty$	-1				

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

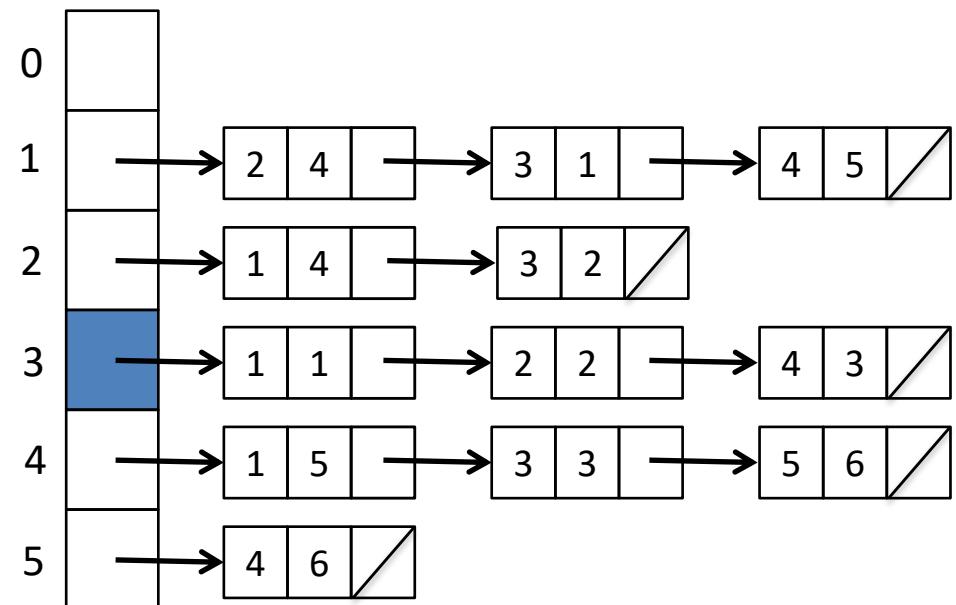
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	F	3	3	2	4	5	1
3	T	1	1	3	1	1	$\infty$
4	F	4	3	1	2	2	2
5	F	$\infty$	-1	2	4	3	

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

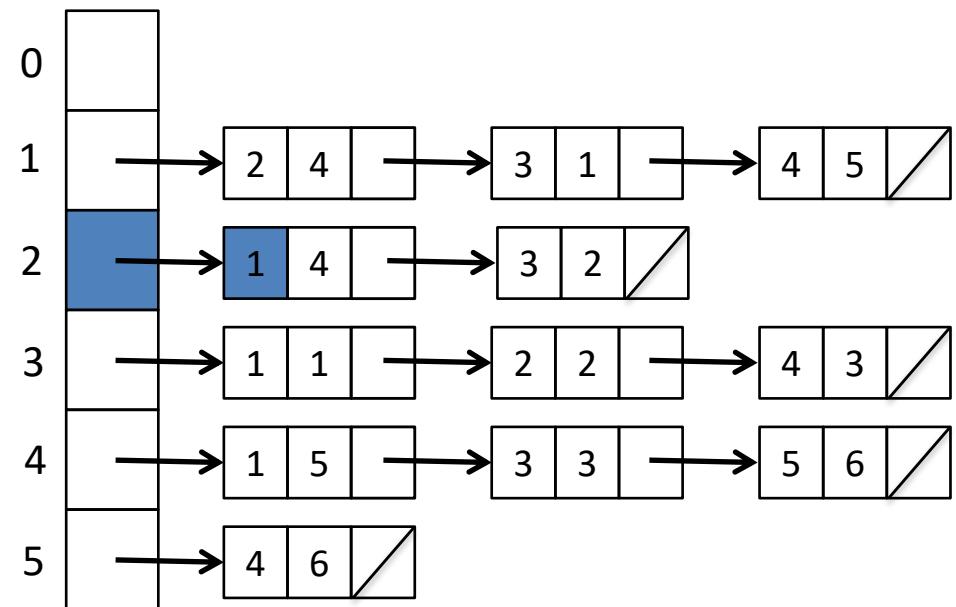
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							$\infty$
1	T	0	-1	1	2	4	4
2	T	3	3	1	3	1	1
3	T	1	1	2	4	3	$\infty$
4	F	4	3	1			2
5	F	$\infty$	-1				4

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

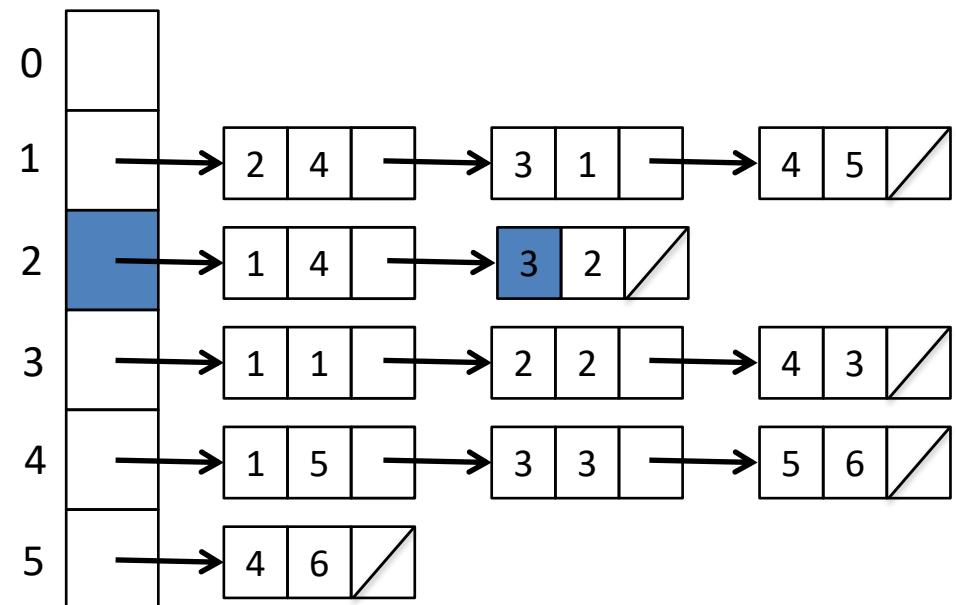
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	T	3	3	2	4	5	1
3	T	1	1	3	1	1	$\infty$
4	F	4	3	1	2	2	2
5	F	$\infty$	-1	2	4	3	3

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

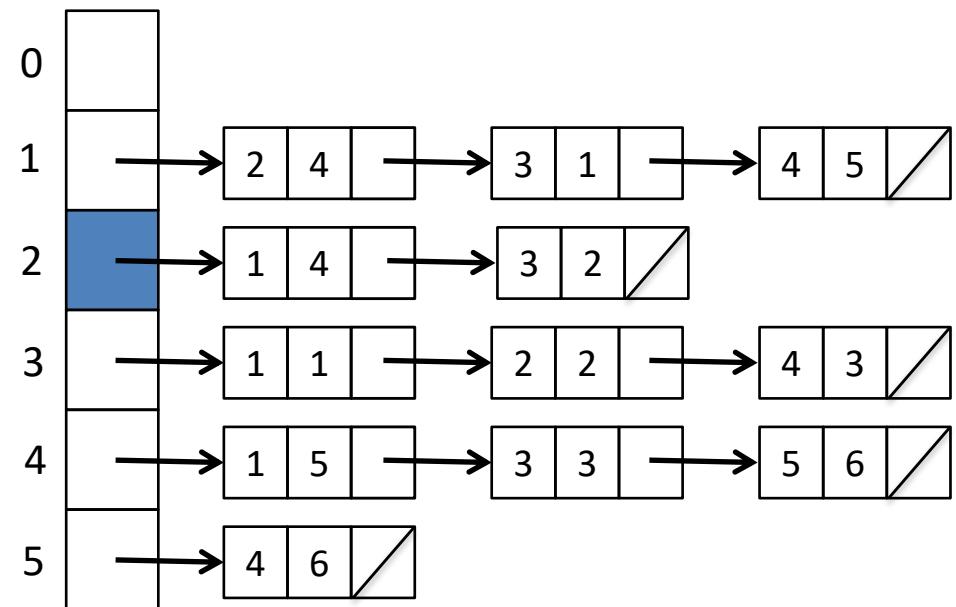
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	v	w	weight	dist
0	1	2	4	∞
1	1	3	1	4
2	2	4	5	1
3	3	1	1	∞
4	1	2	2	2
5	2	4	3	∞
6	1	1	4	3
7	4	3	2	2
8	4	6	1	-

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

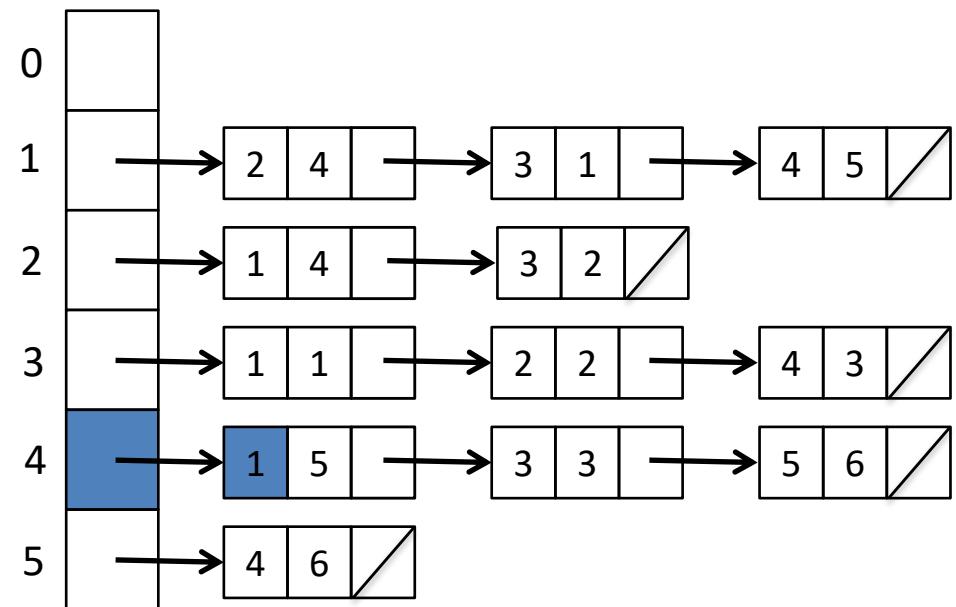
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	T	3	3	2	4	5	1
3	T	1	1	3	1	1	$\infty$
4	T	4	3	1	2	2	2
5	F	$\infty$	-1	2	4	3	$\infty$

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

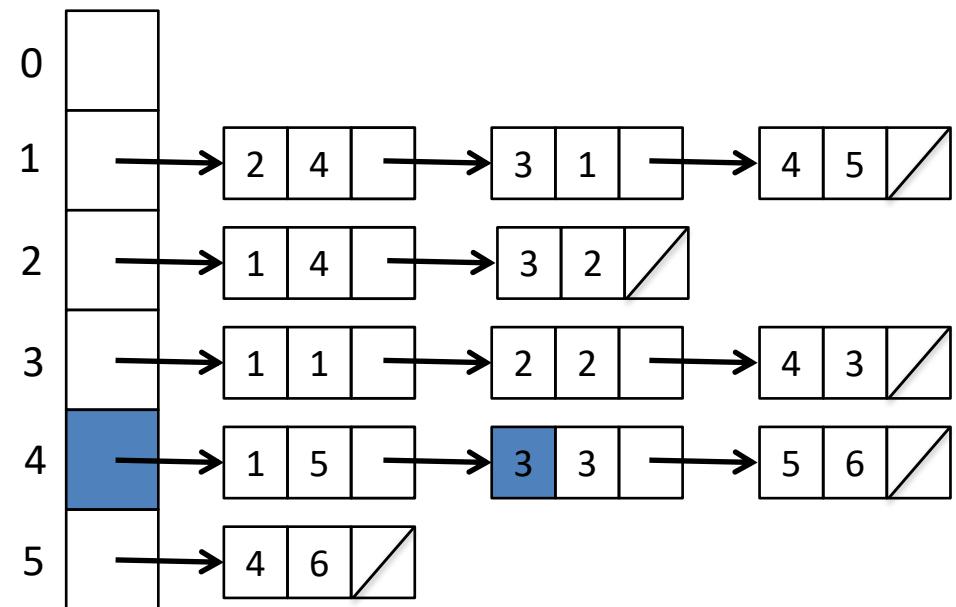
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0							$\infty$
1	T	0	-1	1	2	4	4
2	T	3	3	1	3	1	1
3	T	1	1	2	4	3	$\infty$
4	T	4	3	1	1	4	3
5	F	$\infty$	-1	4	3	2	

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

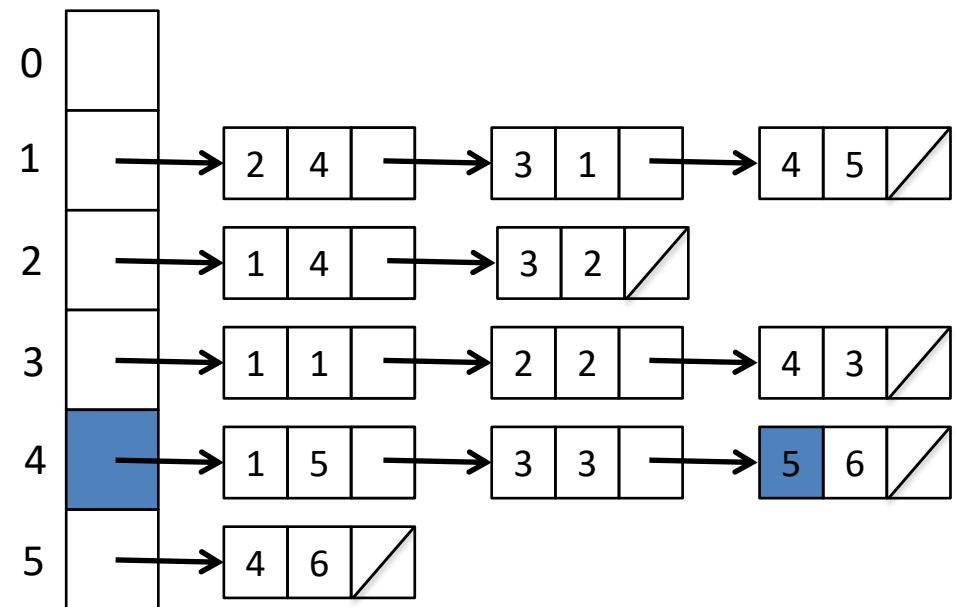
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	T	3	3	2	4	5	1
3	T	1	1	3	1	1	$\infty$
4	T	4	3	1	2	2	2
5	F	$\infty$	-1	2	4	3	3

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

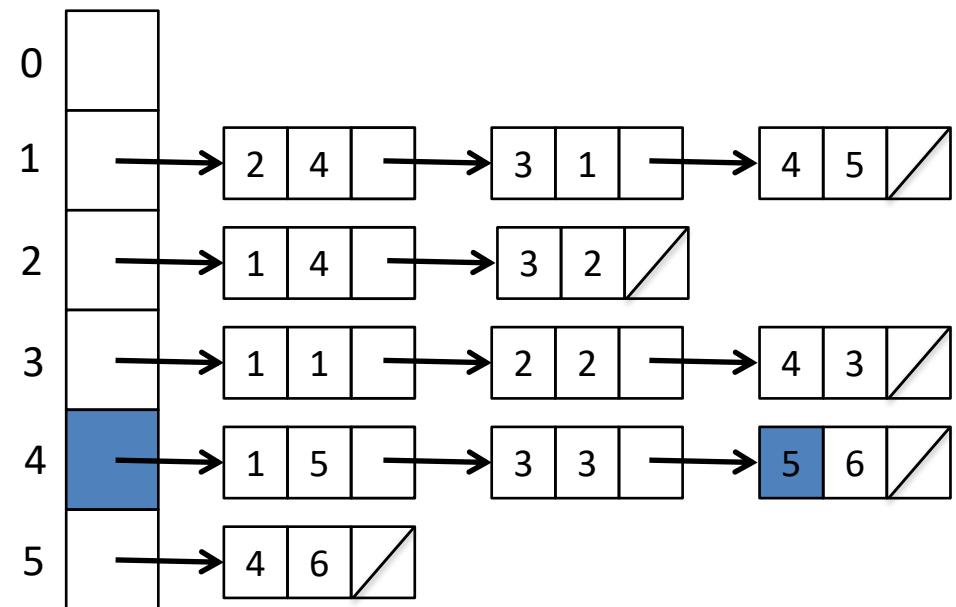
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	2	4	$\infty$
1	T	0	-1	1	3	1	4
2	T	3	3	2	4	5	1
3	T	1	1	3	1	1	$\infty$
4	T	4	3	1	2	2	2
5	F	10	4	2	4	3	3

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

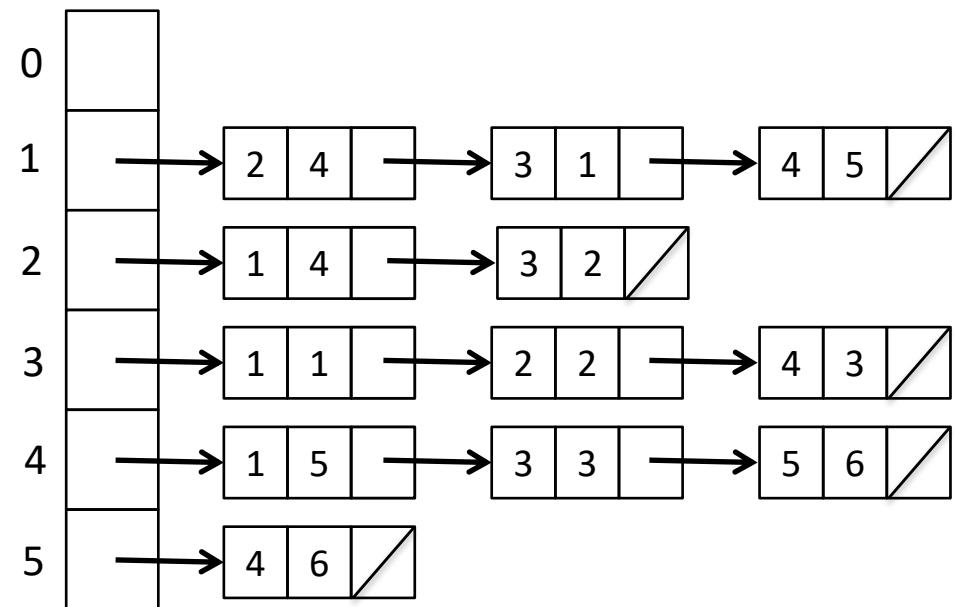
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	v	w	weight	dist
0	1	2	4	$\infty$
1	1	3	1	4
2	2	4	5	1
3	3	1	1	$\infty$
4	1	2	2	2
5	2	4	3	$\infty$
6	1	1	4	3
7	4	3	2	$\infty$
8	1	1	5	6
9	5	3	3	3

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

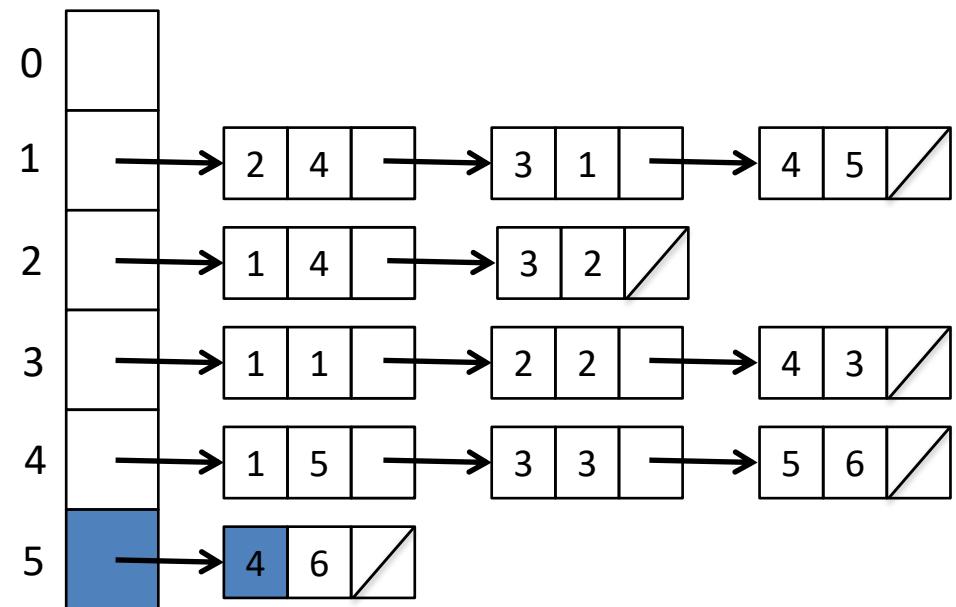
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```



	intree	distance	parent	v	w	weight	dist
0				1	...	...	$\infty$
1	T	0	-1	1	4	6	4
2	T	3	3	2			1
3	T	1	1	2			$\infty$
4	T	4	3	1			3
5	T	10	4	1			6

```

for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

distance[start] = 0;
v = start;

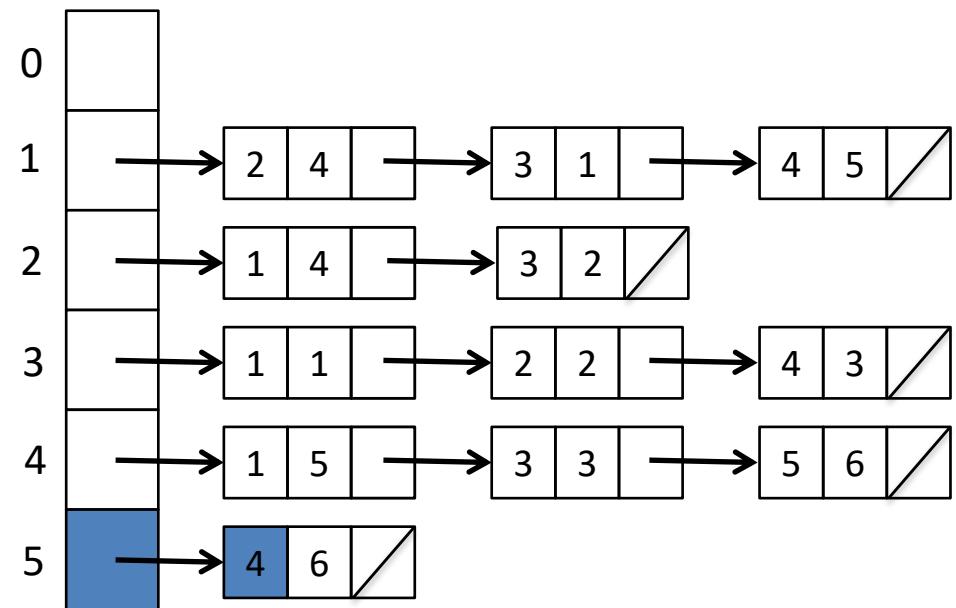
while (intree[v] == FALSE) {

    intree[v] = TRUE;
    p = g->edges[v];

    while (p != NULL) {
        w = p->y;
        weight = p->weight;
        if ((distance[v]+weight < distance[w])) {
            distance[w] = distance[v]+weight ;
            parent[w] = v;
        }
        p = p->next;
    }

    v = 1;
    dist = MAXINT;
    for (i=1; i<=g->nvertices; i++)
        if ((intree[i] == FALSE) &&
            (distance[i] < dist)) {
            dist = distance[i];
            v = i;
        }
}

```

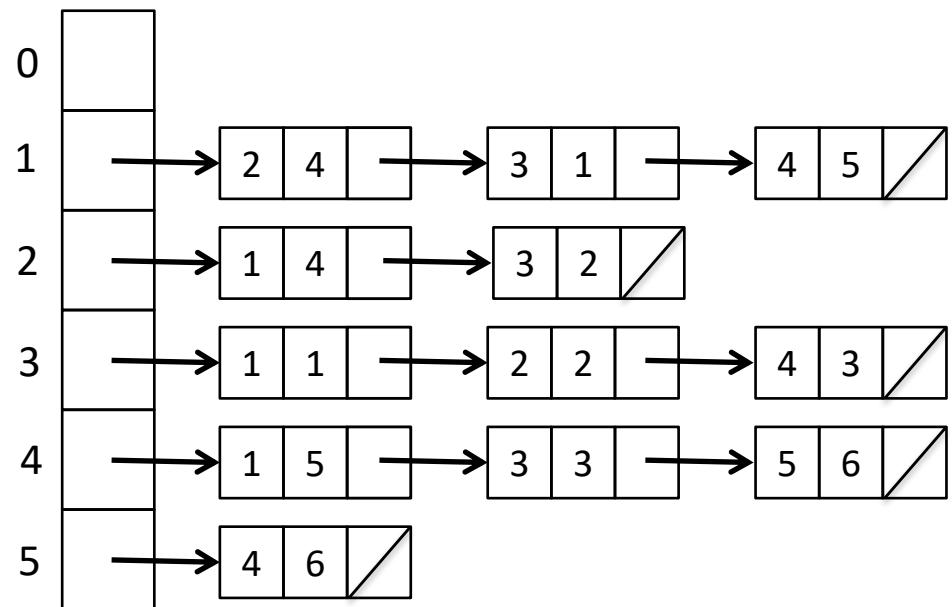
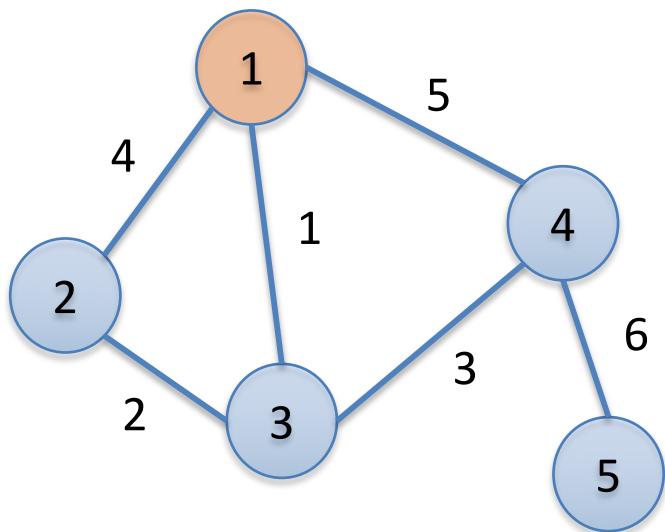


	intree	distance	parent	v	w	weight	dist
0				1	...	...	$\infty$
1	T	0	-1	1	4	6	4
2	T	3	3	2			1
3	T	1	1	2			$\infty$
4	T	4	3	1			3
5	T	10	4	1	5	6	$\infty$

# Shortest Paths

- Dijkstra's Algorithm
  - This implementation finds the shortest path spanning tree, i.e. shortest path between a `start` vertex and all other vertices
  - The length of the shortest path from `start` to a given vertex `t` is exactly the value of `distance[t]`
  - To find the actual path, follow the `parent` relations from `t` until we hit `start` (or -1 if no such path exists)
  - We did this in Breadth-First Search

```
find_path(int start, int end, int parents[])
```



	parent	distance
0		
1	-1	0
2	3	3
3	1	1
4	3	4
5	4	10

# Shortest Paths

- All-Pairs Shortest Path – Floyd's Algorithm
  - Find the centre vertex in a graph
    - Minimize **longest or average distance to all other nodes**
    - e.g. good place to set up a pizza shop

# Shortest Paths

- All-Pairs Shortest Path – Floyd's Algorithm
  - Find the diameter of a graph
    - Minimize **longest shortest-path distance over all pairs of vertices**
    - e.g. longest possible time for a network packet to be delivered
  - Both examples require computation of the **shortest path between all pairs of vertices** in a given graph ( $n \times n$  distance matrix)

# Shortest Paths

- All-Pairs Shortest Path – Floyd's Algorithm
  - Simple solution: call Dijkstra's algorithm from **each of the  $n$  possible starting vertices** (hence  $n \times n$  distance matrix)
  - Floyd-Warshall Algorithm
    - Construct  $n \times n$  shortest-path distance matrix directly **from original  $n \times n$  weight matrix**
    - Use adjacency matrix instead of adjacency list data structure

# Shortest Paths

- All-Pairs Shortest Path – Floyd's Algorithm

Use adjacency matrix instead of adjacency list data structure

```
typedef struct {
    int weight[MAXV+1][MAXV+1]; /* adjacency/weight info */
    int nvertices;                /* number of vertices in graph */
} adjacency_matrix;
```

Initialize each non-edge to **MAXINT** (**INT\_MAX**) instead of **zero**

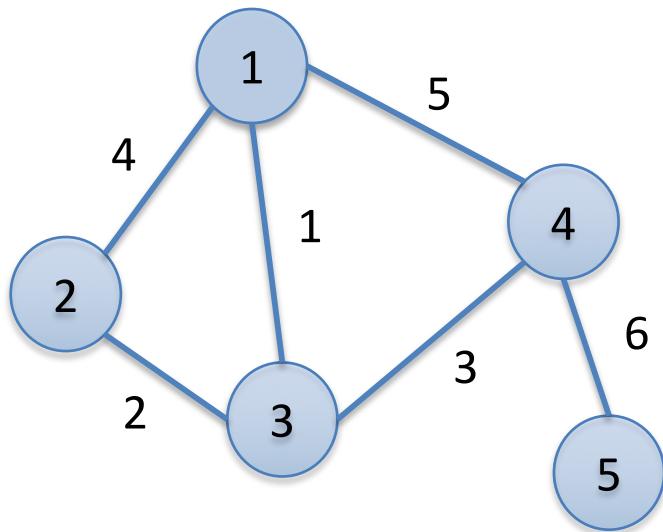
Will flag non-edge and still be ignored in shortest path algorithm

# Shortest Paths

- Floyd-Warshall Algorithm
  - Number all the vertices from 1 to  $n$ 
    - Use these numbers to order the vertices (not label them)
  - Define  $W[i, j]^k$  to be the **length** of the shortest path from  $i$  to  $j$  using vertices numbered from  $1, 2, \dots, k$  as possible intermediate vertices

# Shortest Paths

- Floyd-Warshall Algorithm



$\infty$	4	1	5	$\infty$
4	$\infty$	2	$\infty$	$\infty$
1	2	$\infty$	3	$\infty$
5	$\infty$	3	$\infty$	6
$\infty$	$\infty$	$\infty$	6	$\infty$

# Shortest Paths

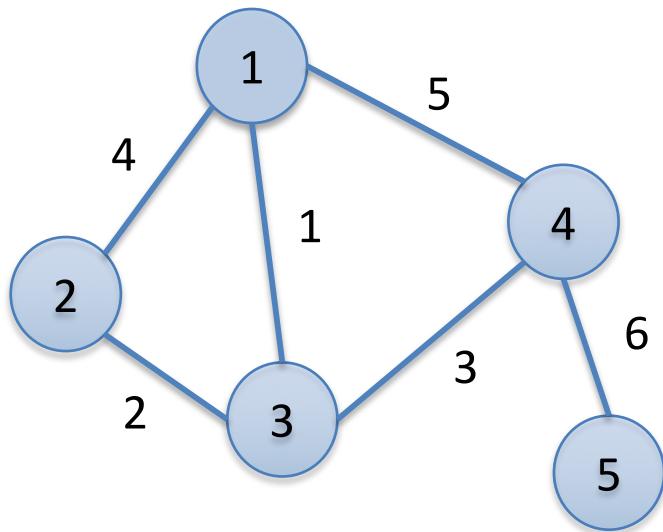
- Floyd-Warshall Algorithm
  - Perform  $k$  iterations
  - Iteration  $k$  allows only the first  $k$  vertices as possible intermediate steps on the path from between each pair of vertices  $x$  and  $y$
  - With each iteration we allow a richer set of possible shortest paths by adding a new vertex as a possible intermediary
  - Allowing the  $k$ th vertex as an intermediary stop helps only if there is a short path that goes through  $k$

$$W[i, j]^k = \min(W[i, j]^{k-1}, W[i, k]^{k-1} + W[k, j]^{k-1})$$

Think about this ...

# Shortest Paths

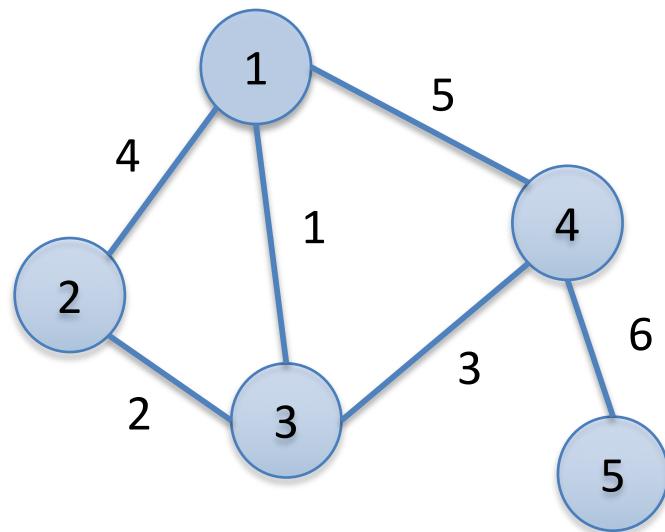
- Floyd-Warshall Algorithm



$\infty$	4	1	5	$\infty$
4	$\infty$	2	$\infty$	$\infty$
1	2	$\infty$	3	$\infty$
5	$\infty$	3	$\infty$	6
$\infty$	$\infty$	$\infty$	6	$\infty$

# Shortest Paths

- Floyd-Warshall Algorithm



$k=1$

$\infty$	4	1	5	$\infty$
4	<b>8</b>	2	<b>9</b>	$\infty$
1	2	<b>2</b>	3	$\infty$
5	<b>9</b>	3	<b>10</b>	6
$\infty$	$\infty$	$\infty$	6	$\infty$

$$W[i, j]^k = \min(W[i, j]^{k-1}, W[i, k]^{k-1} + W[k, j]^{k-1}) \quad k=1$$

# Shortest Paths

```
floyd(adjacency_matrix *g) {  
    int i,j;          /* dimension counters */  
    int k;            /* intermediate vertex counter */  
    int through_k;   /* distance through vertex k */  
  
    for (k=1; k<=g->nvertices; k++) {  
        for (i=1; i<=g->nvertices; i++) {  
            for (j=1; j<=g->nvertices; j++) {  
                through_k = g->weight[i][k]+g->weight[k][j];  
                if (through_k < g->weight[i][j]) {  
                    g->weight[i][j] = through_k;  
                }  
            }  
        }  
    }  
}
```

# Shortest Paths

- Floyd-Warshall Algorithm
  - $O(n^3)$
  - No better than  $n$  calls to Dijkstra's algorithm
  - Better in practice (tight loops)
  - One of the few algorithms that work better on adjacency matrices
  - Does not allow you to construct the actual shortest path between any given pair of vertices (not the point of the algorithm)