Introduction to Cognitive Robotics

Module 3: Mobile Robots

Lecture 9: Dijkstra's shortest path algorithm

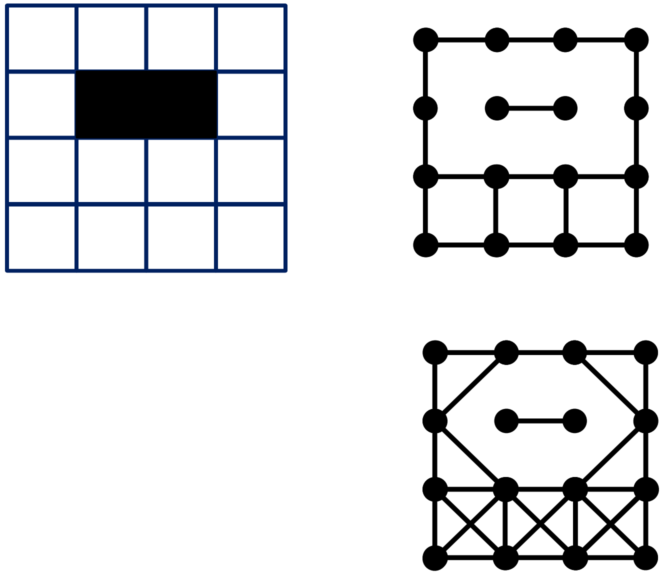
David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Finding the Shortest Path in a Map

Environment map

- If we represent the map as a graph
  - Free cells are vertices in one or more connected components
  - Obstacle cells are vertices in one or more connected components
    - Not strictly necessary because the robot path is confined to the free space connected component(s)
- We can use graph traversal algorithms to find the shortest path connecting a start position and a goal position



Finding the Shortest Path in a Map

Environment map

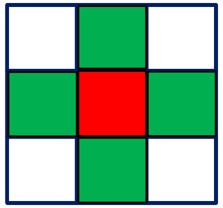
Vertices represent free space, i.e., navigable space

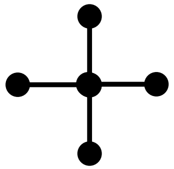
What about the edges? There are two possibilities

1. A vertex can be connected to four horizontal neighbour vertices: 4-connectivity

   - All edges represent the same distance, e.g. 1

   Use an unweighted graph



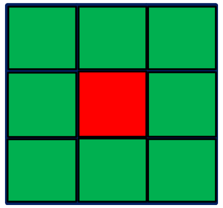


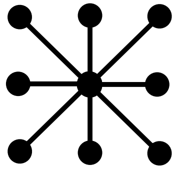
Finding the Shortest Path in a Map

Environment map

2. A vertex can be connected to all eight neighbour vertices: 8-connectivity

- Horizontal edges represent distance of 1
- Diagonal edges represent a distance of $\sqrt{2}$





Need to use a weighted graph:

- weight of 1 for horizontal and vertical edges
- weight $\sqrt{2}$ of for diagonal edges

Finding the Shortest Path in a Map

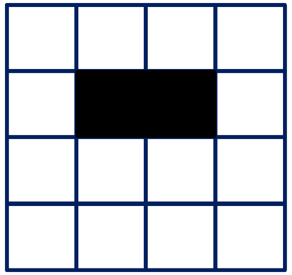
Environment map

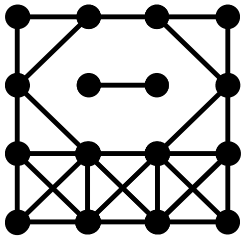
2. A vertex can be connected to all eight neighbour vertices: 8-connectivity

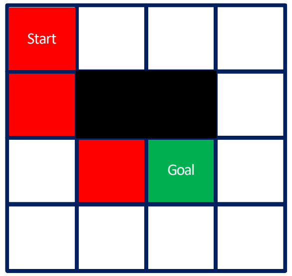
- Horizontal edges represent distance of 1
- Diagonal edges represent a distance of $\sqrt{2}$

Need to use a weighted graph:

- weight of 1 for horizontal and vertical edges
- weight $\sqrt{2}$ of for diagonal edges







Dijkstra's Shortest Path 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

– $O(n^2)$ with simple data structures

Dijkstra's Shortest Path Algorithm

– Greedy algorithm

– Repeatedly select the smallest weight edge that will extend the path

- Begin with some start vertex,
- Extend the path, one edge at a time
- Until all vertices are included

– Thus, incrementally construct the shortest path to all vertices

Dijkstra's Shortest Path Algorithm

The principle behind Dijkstra's algorithm is that

if $(s, \ldots, x, \ldots, t)$ is the shortest path from s to t,
then $(s, \ldots, x)$ had better be the shortest path from s to x.

This suggests a dynamic programming-like strategy:

We store the distance from s to all nearby vertices,
and use them to find the shortest path to more distant vertices.

Dijkstra's Shortest Path Algorithm

$known = \{s\}$
for $i = 1$ to n, $dist[i] = \infty$
for each edge (s, v), $dist[v] = d(s, v)$
last=s
while $(last \neq t)$
    select v such that $dist(v) = \min_{unknown} dist(i)$
    for each (v, x), $dist[x] = \min(dist[x], dist[v] + w(v, x))$
    last=v
    $known = known \cup \{v\}$

Select from the unknown vertices

S. Skiena, The Algorithm Design Manual, Springer 2010

Dijkstra's Shortest Path Algorithm

```
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)
        if dist[x] > dist[v_next] + w(v_next, x)
            dist[x] = dist[v_next] + w(v_next, x)
            parent[x] = v_next
    last = v_next
    path = path ∪ {v_next}
```

Dijkstra's Shortest Path Algorithm

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

The weight of edge (s, v) from vertex s to vertex v

last = s

while (last != t)

  select v_{next}, the unknown vertex minimizing dist[v]

  for each edge (v_{next}, x)

    if dist[x] > dist[v_{next}] + w(v_{next}, x)

      dist[x] = dist[v_{next}] + w(v_{next}, x)

      parent[x] = v_{next}

  last = v_{next}

  path = path ∪ {v_{next}}

Dijkstra's Shortest Path Algorithm

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)
        if dist[x] > dist[v_next] + w(v_next, x)
            dist[x] = dist[v_next] + w(v_next, x)
            parent[x] = v_next
    last = v_next
    path = path ∪ {v_next}
```

Extend the path from the vertex with the shortest distance so far

Dijkstra's Shortest Path Algorithm

```
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)
        if dist[x] > dist[v_next] + w(v_next, x)
            dist[x] = dist[v_next] + w(v_next, x)
            parent[x] = v_next
    last = v_next
    path = path ∪ {v_next}
```

This can be implemented efficiently using a priority queue (implemented as a binary heap)

Dijkstra's Shortest Path Algorithm

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)            <- We now have a new way of reaching x ...
        if dist[x] > dist[v_next] + w(v_next, x)   <- if the total distance to x is less than the current distance
            dist[x] = dist[v_next] + w(v_next, x)  <- update the total distance to x
            parent[x] = v_next                     <- Record the parent of x
    last = v_next
    path = path ∪ {v_next}
```

Dijkstra's Shortest Path Algorithm

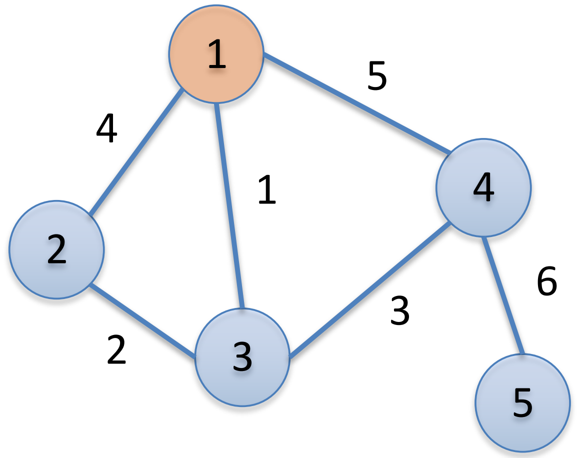
```
/* Dijkstra's algorithm */

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;
```

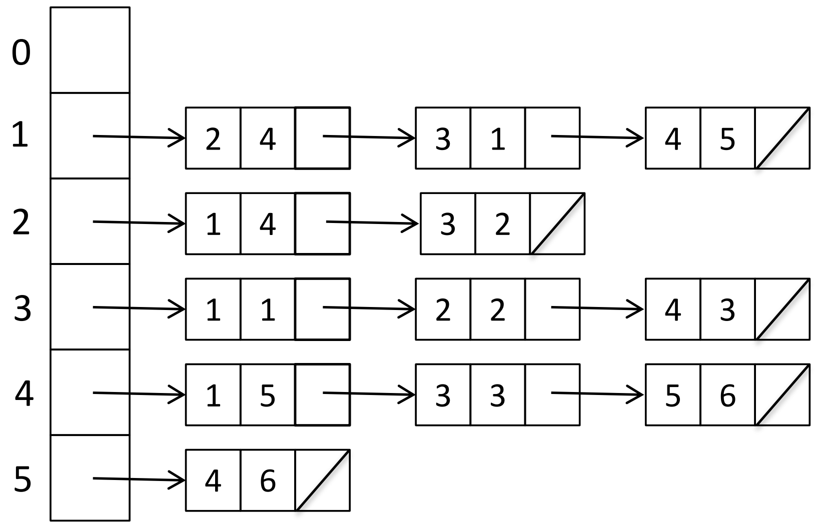
S. Skiena, The Algorithm Design Manual, Springer 2010

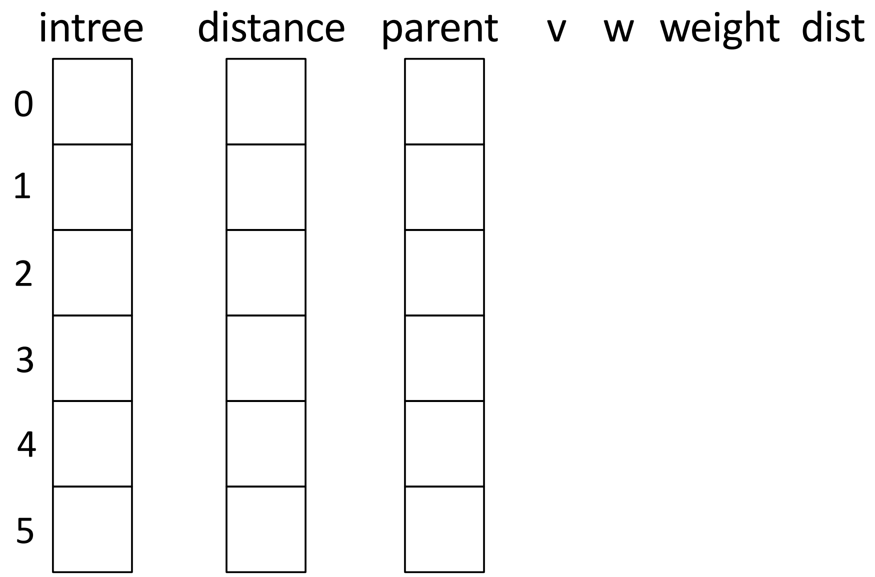
Dijkstra's Shortest Path Algorithm

```
    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])) { // can we improve
                distance[w] = distance[v] + weight;     // on the distance to w?
                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;
            }
    }
}
```

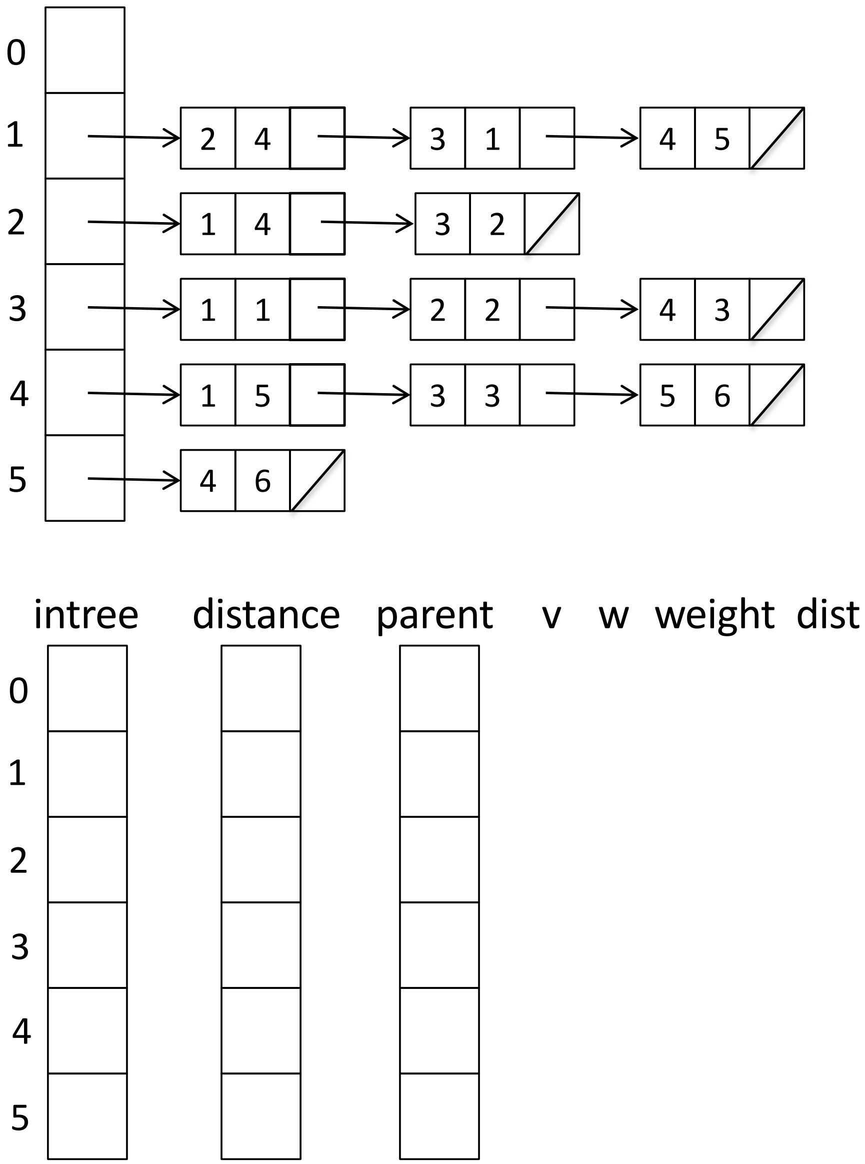
S. Skiena, The Algorithm Design Manual, Springer 2010


1
2
3
4
5
4
5
1
2
3
6

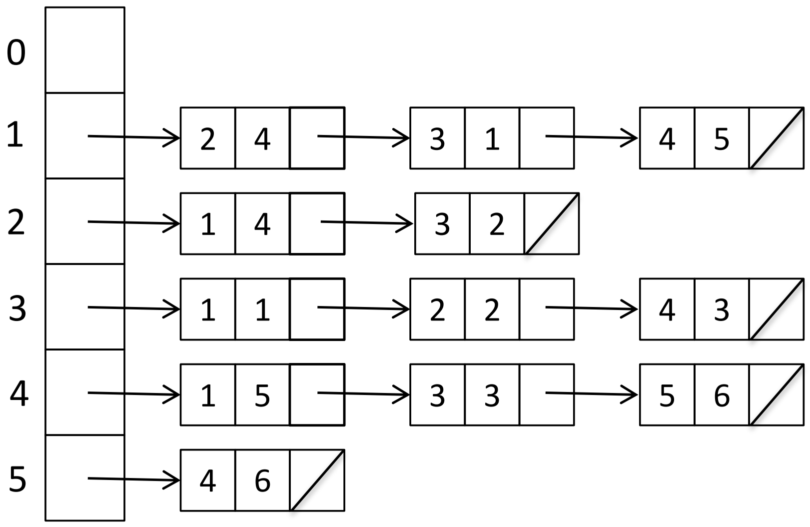

0
1 → 2 4 → 3 1 → 4 5
2 → 1 4 → 3 2
3 → 1 1 → 2 2 → 4 3
4 → 1 5 → 3 3 → 5 6
5 → 4 6


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;
      }
}
```

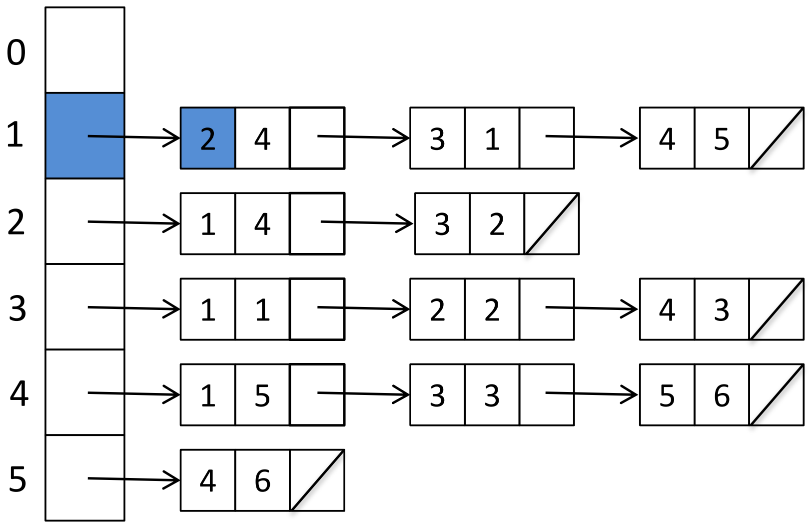


```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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	∞	-1				
2	F	∞	-1				
3	F	∞	-1				
4	F	∞	-1				
5	F	∞	-1				

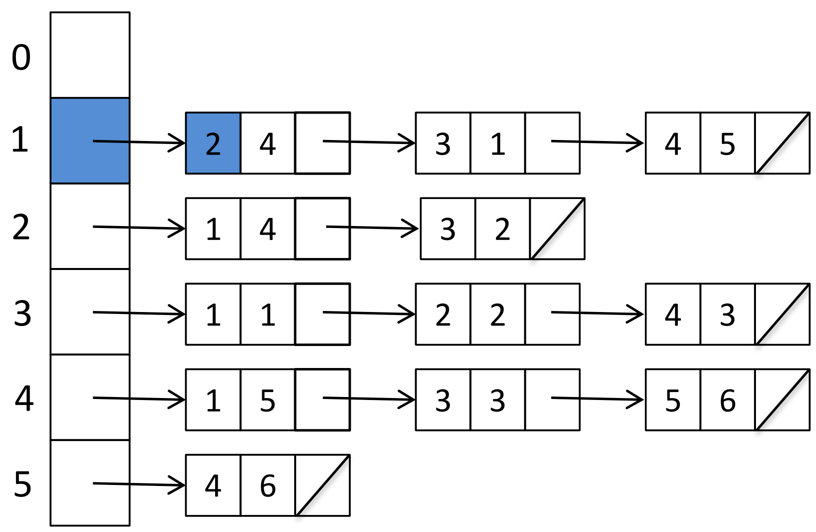
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	∞	-1
3	F	∞	-1
4	F	∞	-1
5	F	∞	-1

v	w	weight	dist
1	2	4	

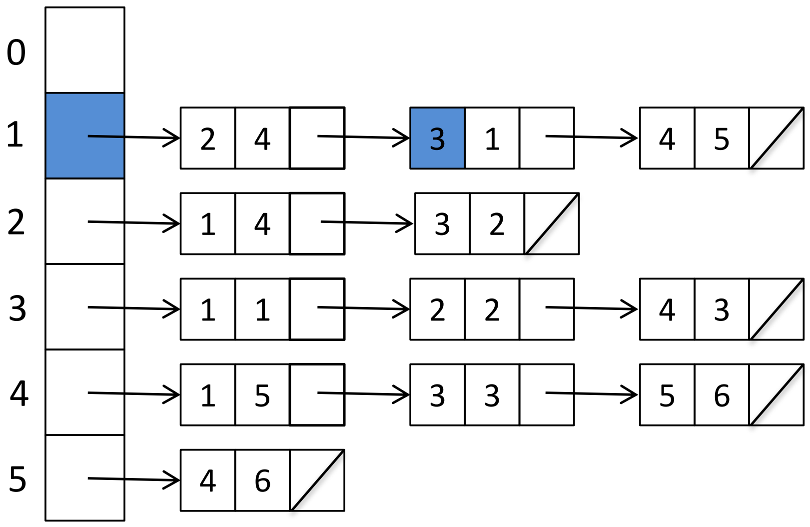
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	4	1
3	F	∞	-1
4	F	∞	-1
5	F	∞	-1

v	w	weight	dist
1	2	4	

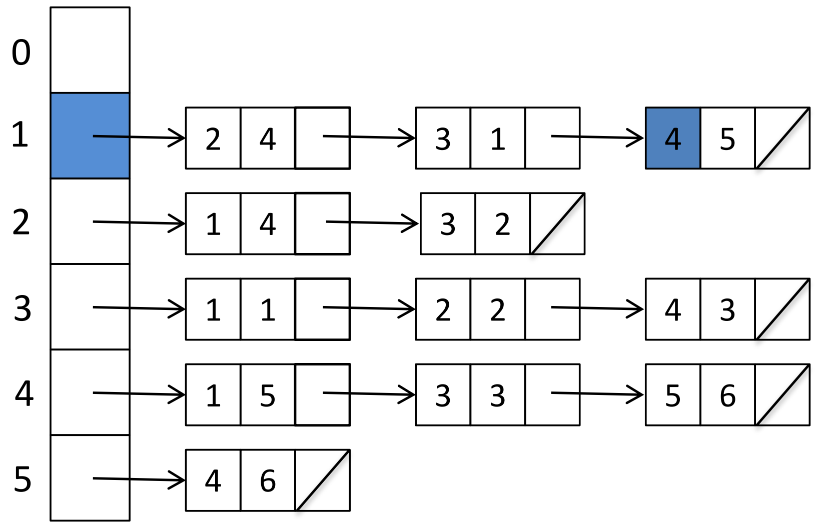
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	4	1
3	F	1	1
4	F	∞	-1
5	F	∞	-1

v	w	weight	dist
1	2	4	
	3	1	

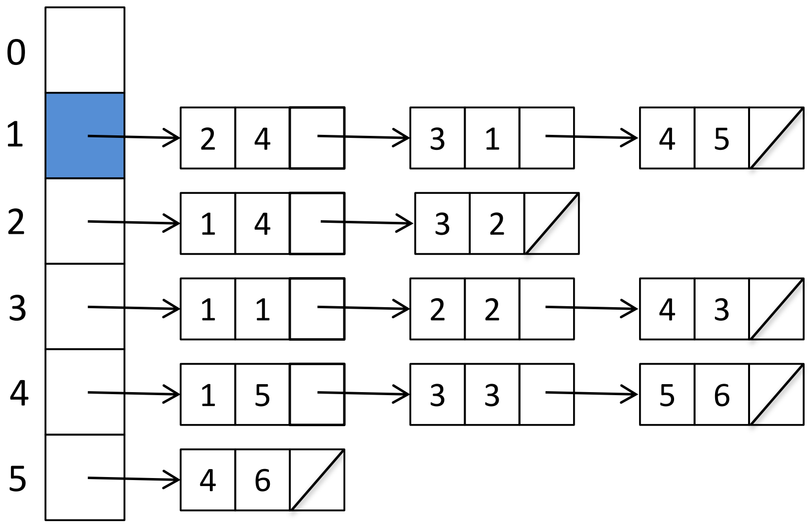
```
for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	4	1
3	F	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	
	3	1	
	4	5	

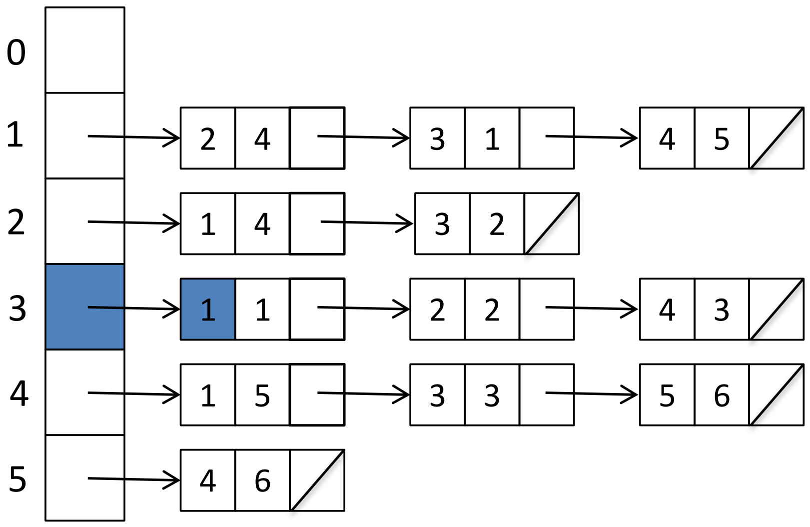
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	4	1
3	F	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3			

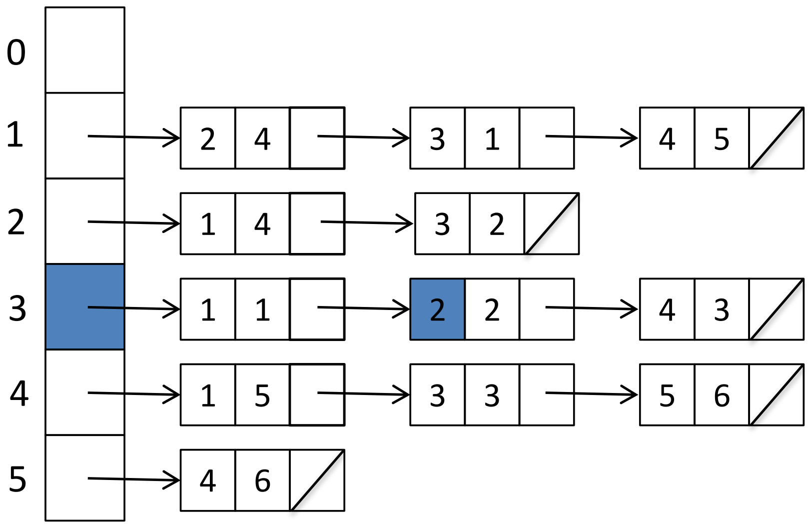
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	4	1
3	T	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	

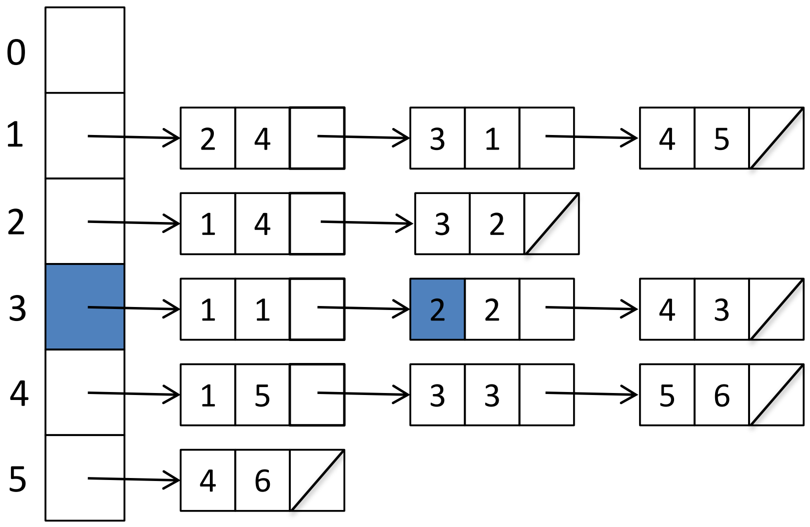
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	4	1
3	T	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	
	2	2	

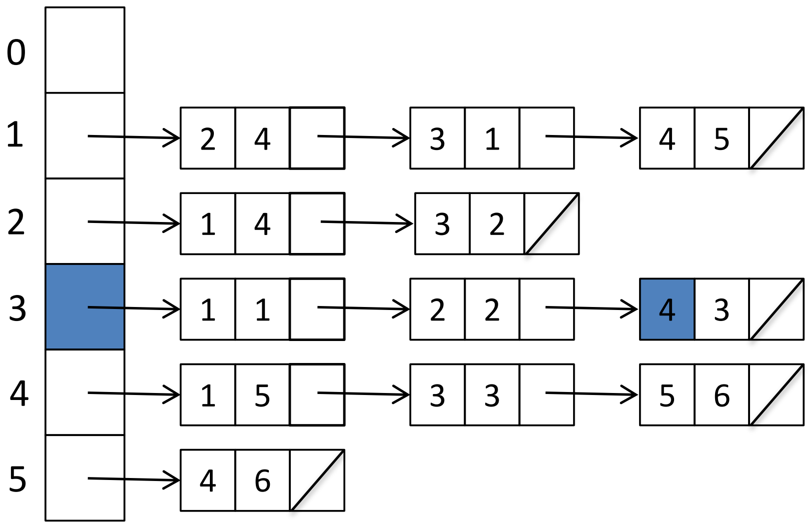
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	3	3
3	T	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	
	2	2	

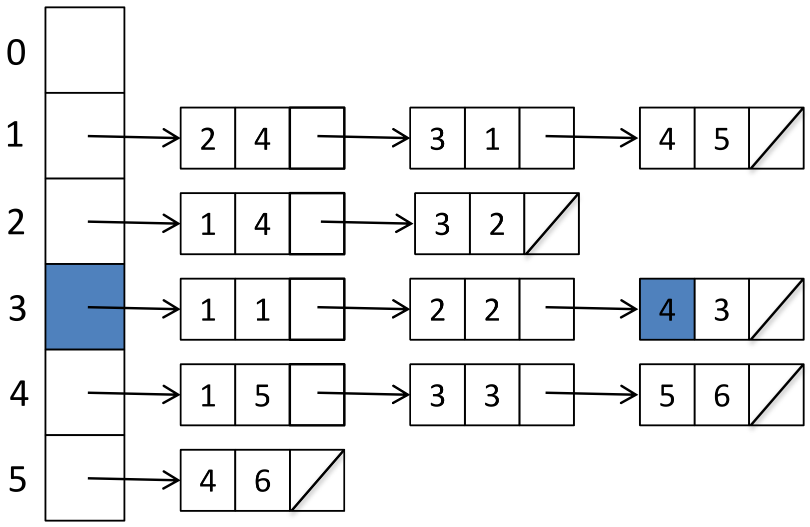
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	3	3
3	T	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	
	2	2	

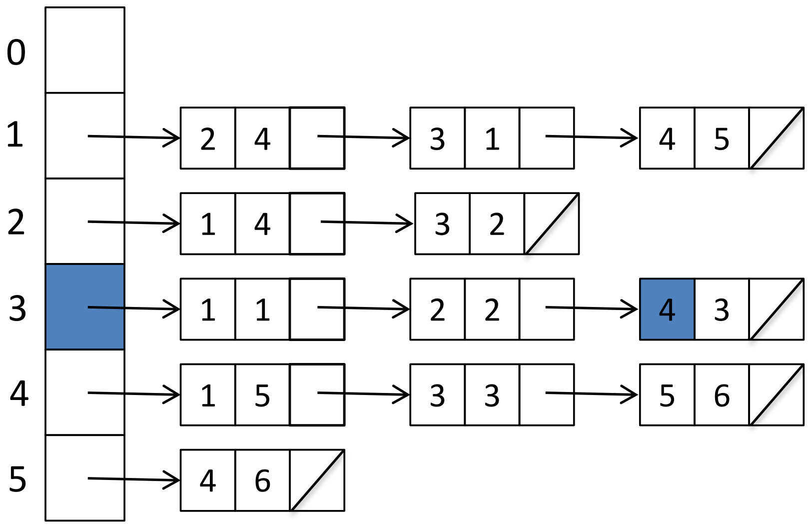
```
for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	3	3
3	T	1	1
4	F	5	1
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	
	2	2	
	4	3	

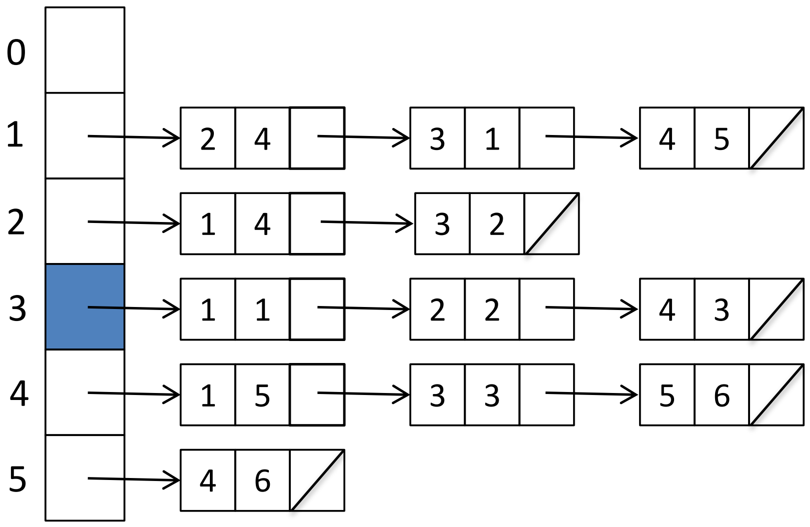
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	3	3
3	T	1	1
4	F	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	
	2	2	
	4	3	

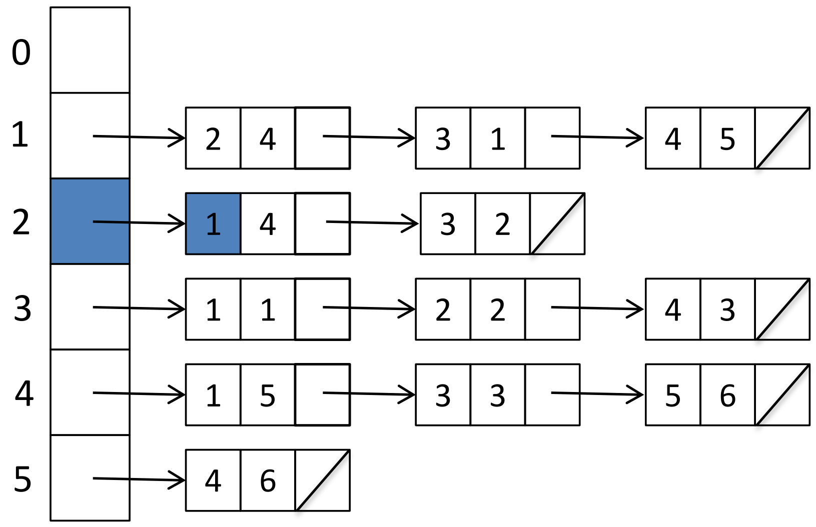
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	F	3	3
3	T	1	1
4	F	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	

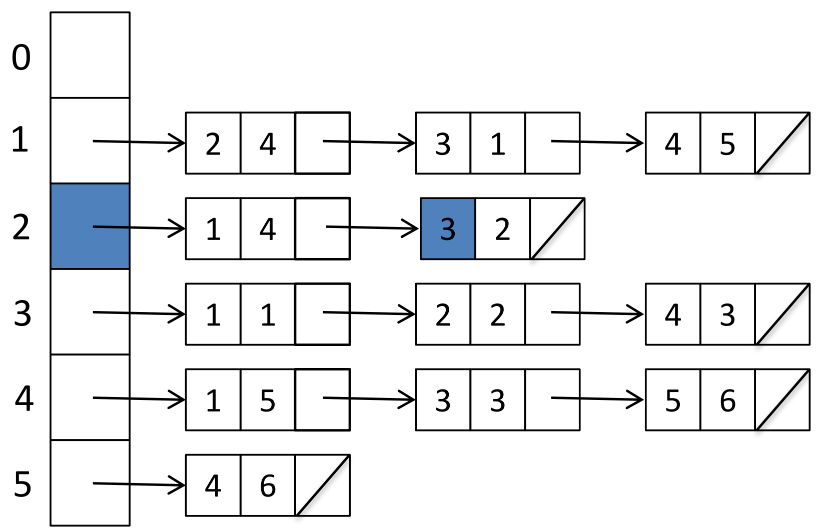
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	F	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	
	1	4	

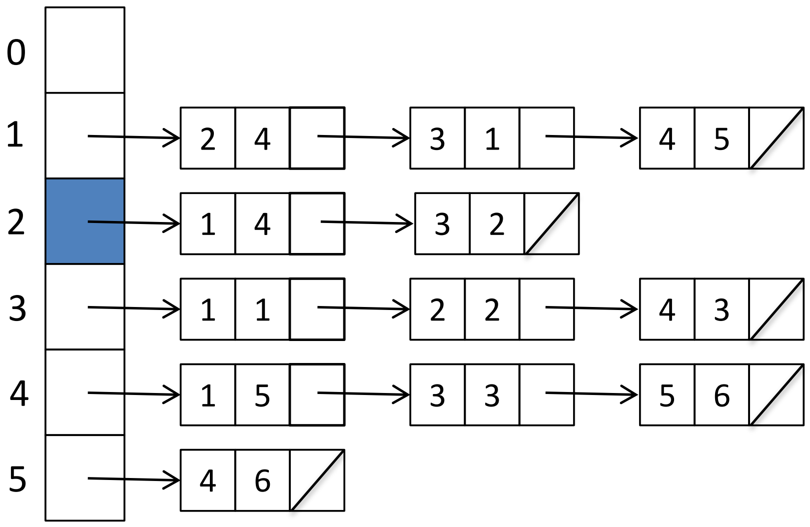
```
for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	F	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	
	1	4	
	3	2	

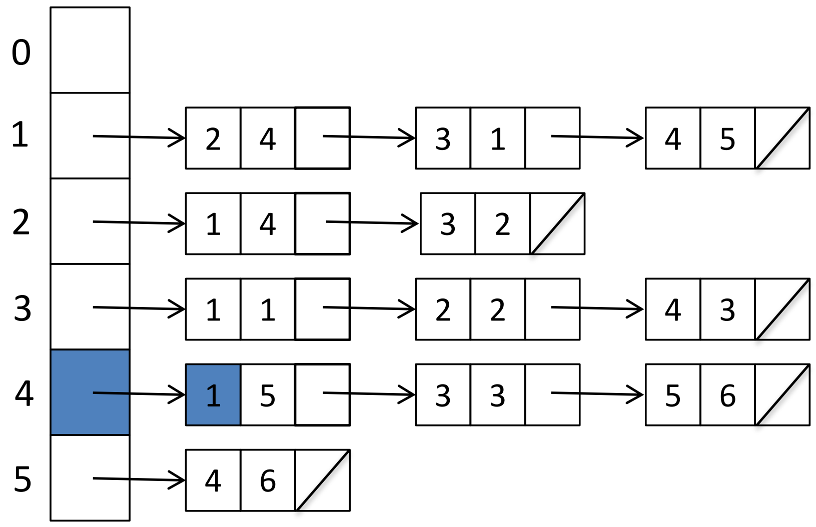
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	F	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	∞
1	1	4	3
4	3	2	

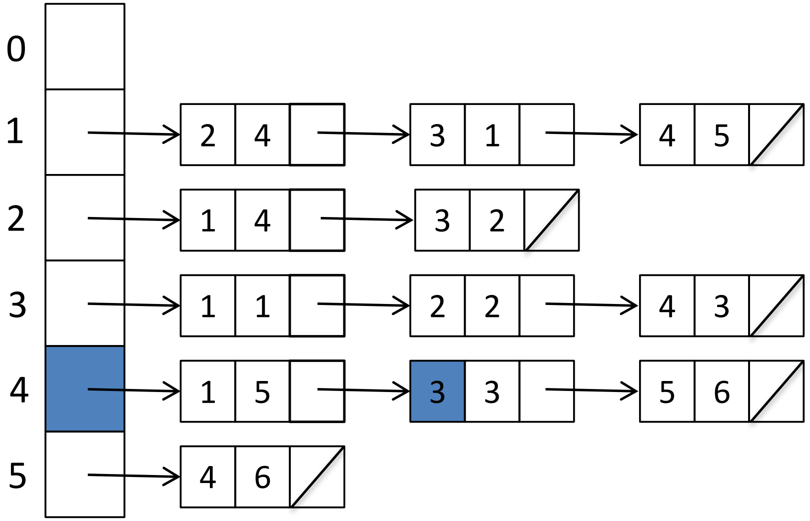
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	∞
1	1	4	3
4	3	2	
	1	5	

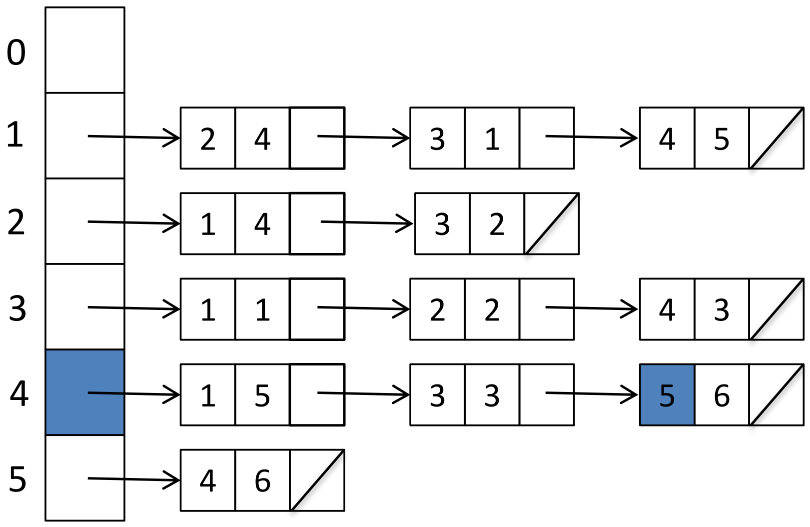
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	∞
1	1	4	3
4	3	2	
	1	5	
	3	3	

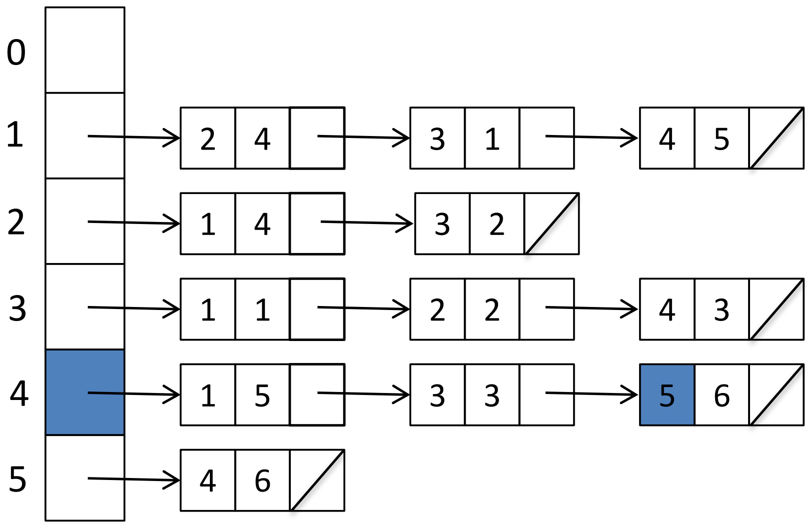
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	F	∞	-1

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	∞
1	1	4	3
4	3	2	
	1	5	
	3	3	
	5	6	

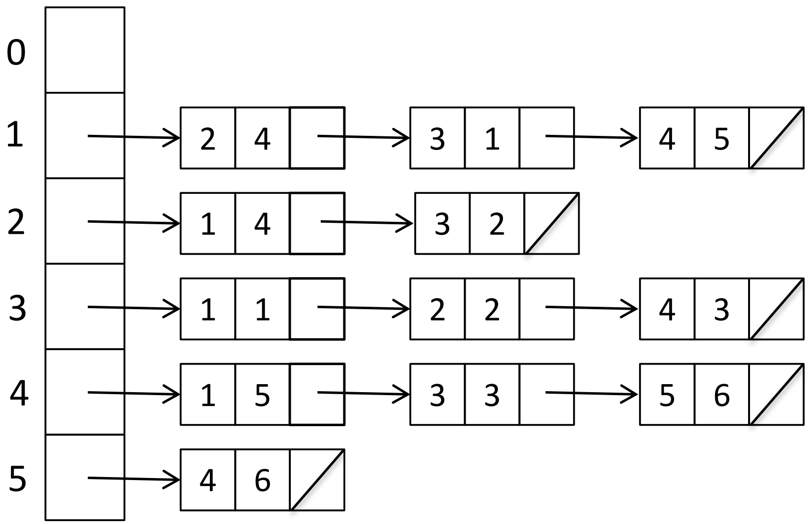
```
for (i=1; i<=g->nvertices; i++) {
    intree[i] = FALSE;
    distance[i] = MAXINT;
    parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	F	10	4

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	∞
1	1	4	3
4	3	2	
	1	5	
	3	3	
	5	6	

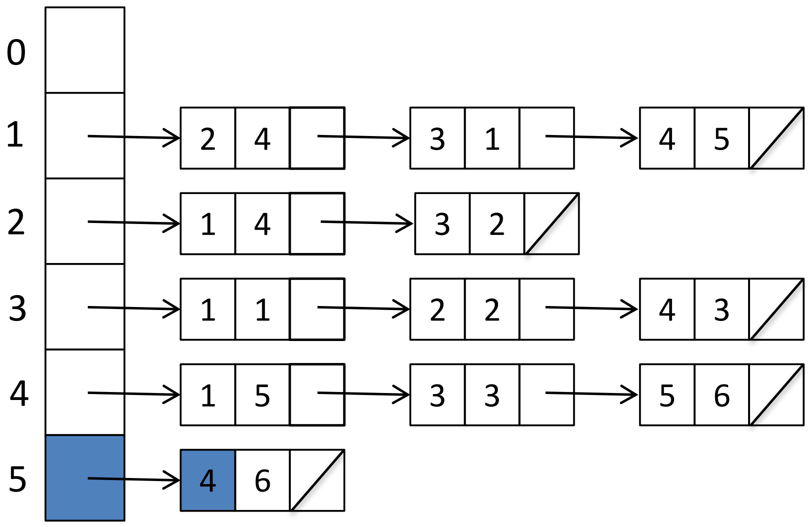
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	F	10	4

v	w	weight	dist
1	2	4	∞
1	3	1	4
2	4	5	1
3	1	1	∞
1	2	2	2
2	4	3	∞
1	1	4	3
4	3	2	∞
1	1	5	6
5	3	3	
	5	6	

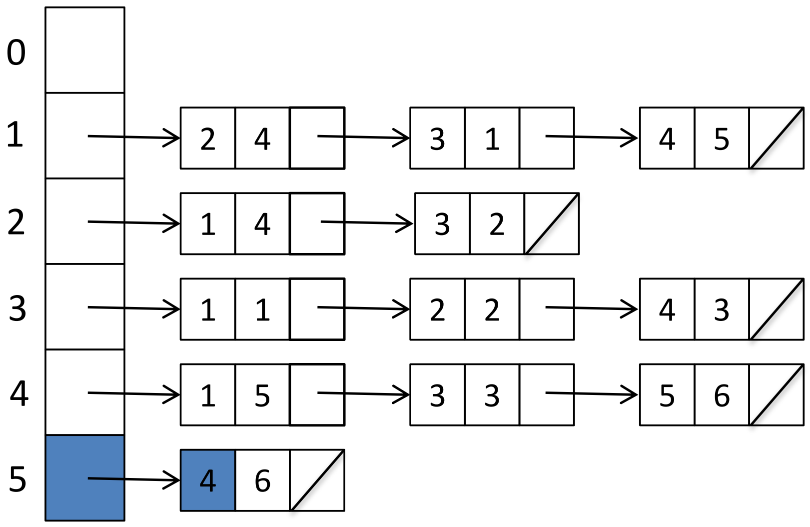
```
for (i=1; i<=g->nvertices; i++) {
   intree[i] = FALSE;
   distance[i] = MAXINT;
   parent[i] = -1;
}

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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	T	10	4

v	w	weight	dist
1	…	…	∞
1	4	6	4
2			1
3			∞
1			2
2			∞
1			3
4			∞
1			6
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
0			
1	T	0	-1
2	T	3	3
3	T	1	1
4	T	4	3
5	T	10	4

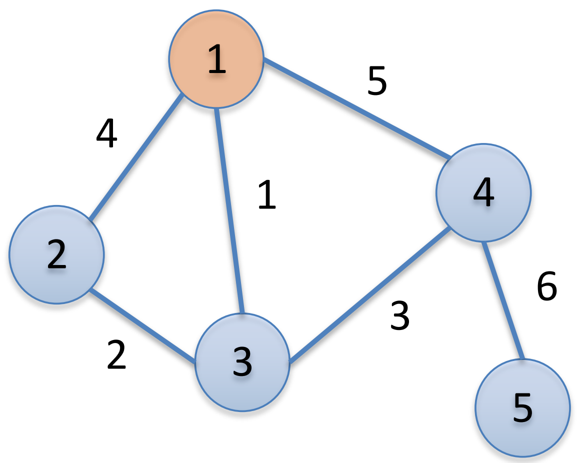
v	w	weight	dist
1	…	…	∞
1	4	6	4
2			1
3			∞
1			2
2			∞
1			3
4			∞
1			6
5			∞
1			

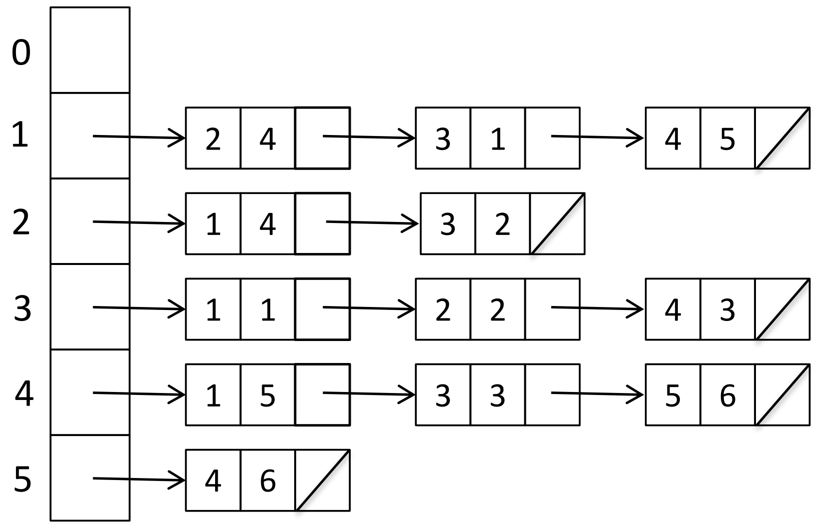
Shortest Paths

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

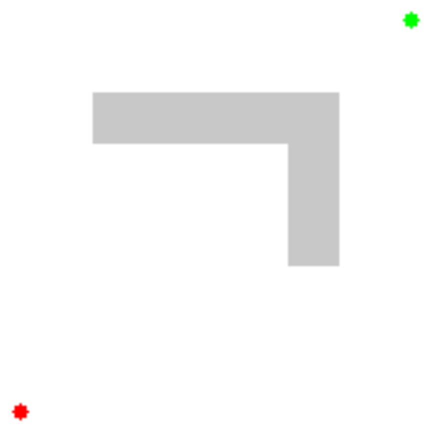
```
find_path(int start, int end, int parents[])
```





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

Dijkstra's Shortest Path Algorithm



"Illustration of Dijkstra's algorithm finding a path from a start node (lower left, red) to a goal node (upper right, green) in a robot motion planning problem. Open nodes represent the "tentative" set (aka set of "unvisited" nodes). Filled nodes are visited ones, with color representing the distance: the greener, the closer. Nodes in all the different directions are explored uniformly, appearing more-or-less as a circular wavefront as Dijkstra's algorithm uses a heuristic identically equal to 0."

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

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[])
```