最短路

南开中学信息学竞赛教练组

最短路问题

- 是个问题

导航

问题：

- 找出南开本部到两江分校的最优路线

什么是最优路线？

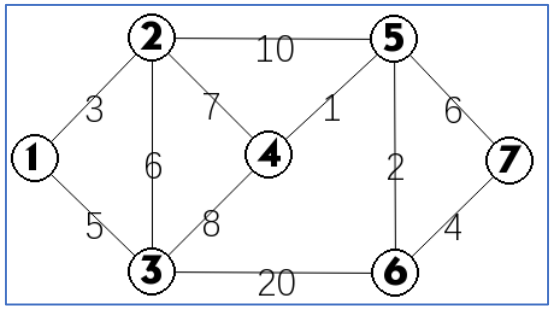
- 策略1：总路程最短
  - 显然不是最优的
  - 要避开限速、修路、拥堵路段
- 策略2：总时间最短
  - 也不一定最优
  - 可以考虑避开收费路段、事故多发路段
- 策略3：给出几条推荐路线，用户自己选择

最短路问题

问题：

- 将导航转化为图论问题：
  - 路口→节点
  - 双向道路→无向边
  - 单行道→有向边
  - 道路长度（或预期的通行时间）→边的长度
  - 最优路径→总长度最小的路径

- 右图中，从节点1到节点7的最短路长度是多少？怎么走？



PART

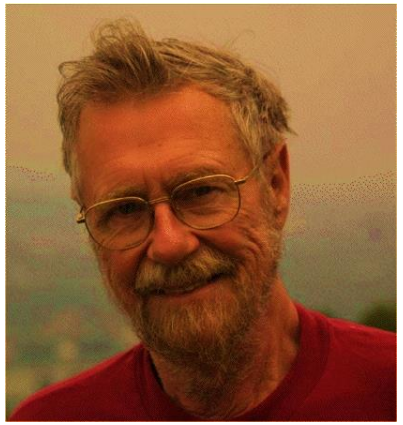
02

Dijkstra算法

- 读作“迪杰斯特拉”
- 最重要的最短路算法，

  没有之一

Dijkstra算法



Dijkstra:

- 全名Edsger Wybe Dijkstra
- 荷兰人，计算机科学家。
- 早年钻研物理及数学，而后转为计算学。
- 曾在1972年获得过素有计算机科学界的诺贝尔奖之称的图灵奖。
- 与D. E. Knuth并称为我们这个时代最伟大的计算机科学家的人。

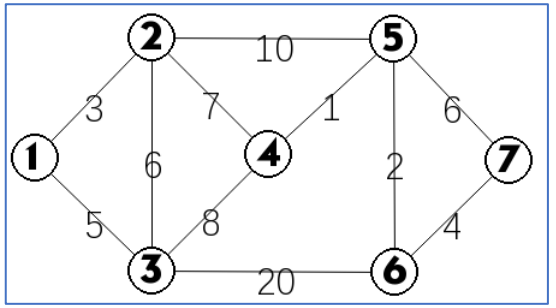
Dijkstra算法:

- 解决“单源最短路”问题
  - 计算从某一个起点到图上所有节点的最短路
- 广泛应用于导航软件、自动驾驶、网络通讯等各种领域。
- 也是信息学竞赛中最重要的算法之一

Dijkstra算法

Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。



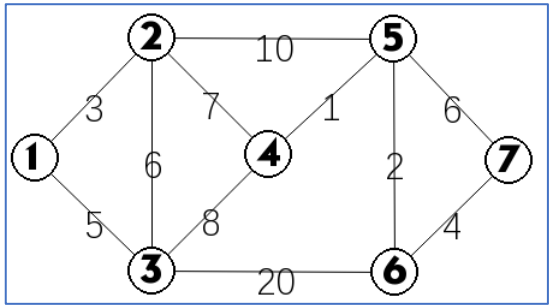
Dijkstra算法

Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	0	999	999	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值



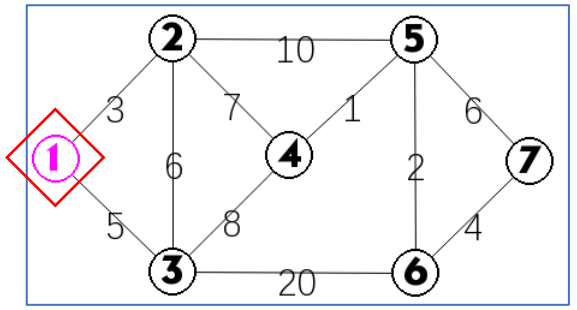
Dijkstra算法

Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	999	999	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；



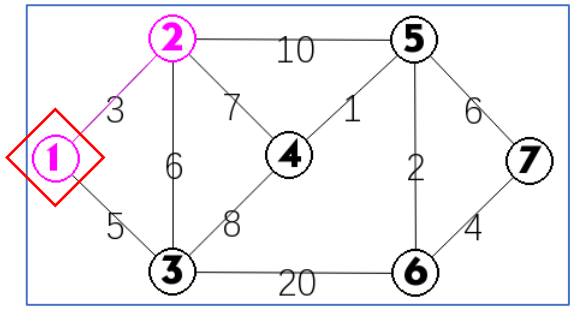
Dijkstra算法

Dijkstra算法：

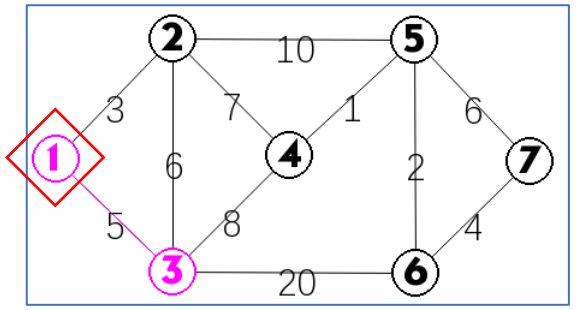
- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	3	999	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3



Dijkstra算法



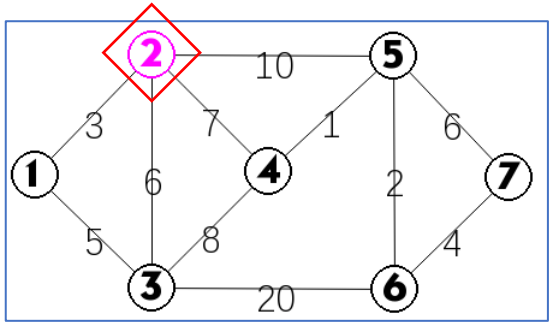
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	3	5	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5

Dijkstra算法



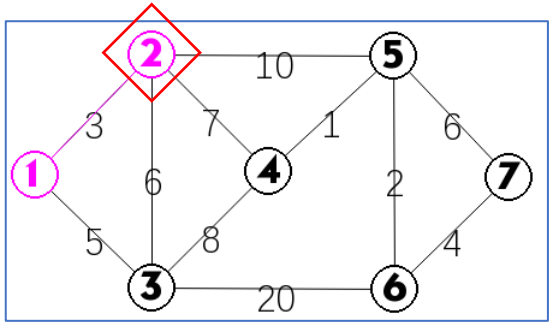
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	5	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；

Dijkstra算法



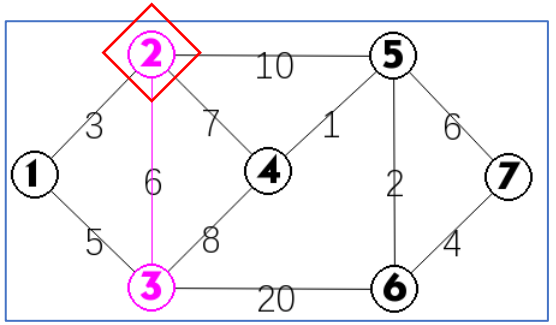
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	5	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新

Dijkstra算法



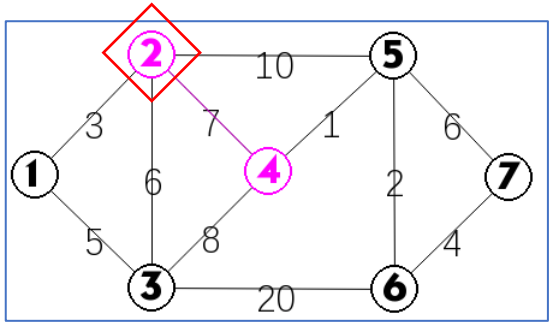
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	5	999	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5

Dijkstra算法



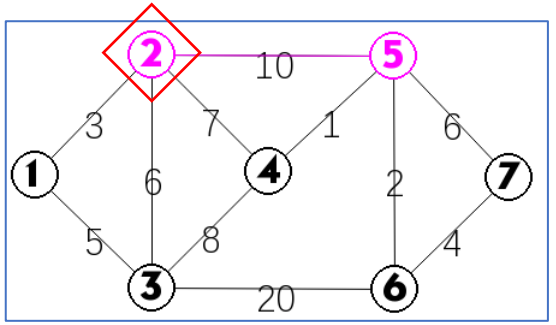
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	5	10	999	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5
  - dis[4] = min(dis[4], dis[2] + 7) = 10

Dijkstra算法



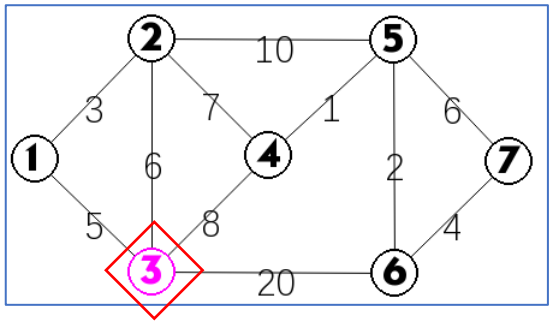
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	5	10	13	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5
  - dis[4] = min(dis[4], dis[2] + 7) = 10
  - dis[5] = min(dis[5], dis[2] + 10) = 13

Dijkstra算法



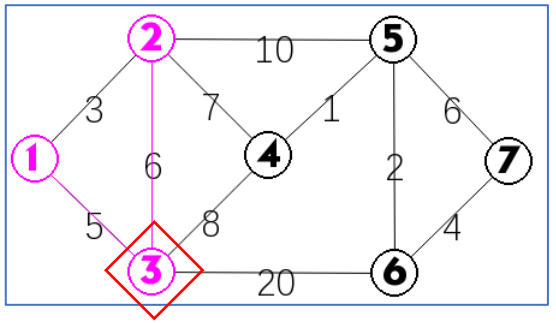
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	10	13	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5
  - dis[4] = min(dis[4], dis[2] + 7) = 10
  - dis[5] = min(dis[5], dis[2] + 10) = 13
- 找出最小的dis[3]=5，打个标记；

Dijkstra算法



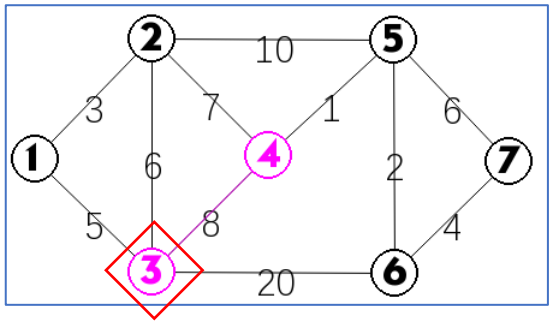
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	10	13	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5
  - dis[4] = min(dis[4], dis[2] + 7) = 10
  - dis[5] = min(dis[5], dis[2] + 10) = 13
- 找出最小的dis[3]=5，打个标记；然后更新与3相邻节点的dis
  - dis[1]和dis[2]已被标记，不更新

Dijkstra算法



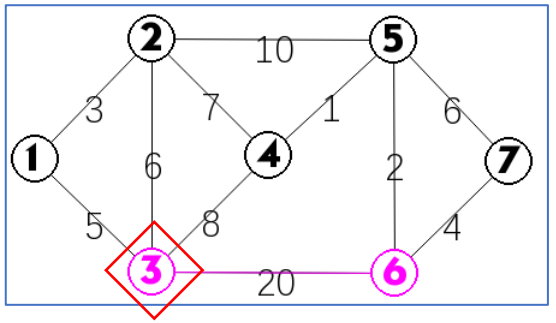
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	10	13	999	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5
  - dis[4] = min(dis[4], dis[2] + 7) = 10
  - dis[5] = min(dis[5], dis[2] + 10) = 13
- 找出最小的dis[3]=5，打个标记；然后更新与3相邻节点的dis
  - dis[1]和dis[2]已被标记，不更新
  - dis[4] = min(dis[4], dis[3] + 8) = 10

Dijkstra算法



Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	10	13	25	999

- dis[i]表示从起点到节点i的当前已经找到的最短路的长度，先赋初值
- 找到最小的dis[1]=0，打个标记；然后更新与1相邻节点的dis
  - dis[2] = min(dis[2], dis[1] + 3) = 3
  - dis[3] = min(dis[3], dis[1] + 5) = 5
- 找到最小的dis[2]=3，打个标记；然后更新与2相邻节点的dis
  - dis[1]已被标记，不更新
  - dis[3] = min(dis[3], dis[2] + 6) = 5
  - dis[4] = min(dis[4], dis[2] + 7) = 10
  - dis[5] = min(dis[5], dis[2] + 10) = 13
- 找出最小的dis[3]=5，打个标记；然后更新与3相邻节点的dis
  - dis[1]和dis[2]已被标记，不更新
  - dis[4] = min(dis[4], dis[3] + 8) = 10
  - dis[6] = min(dis[6], dis[3] + 20) = 25

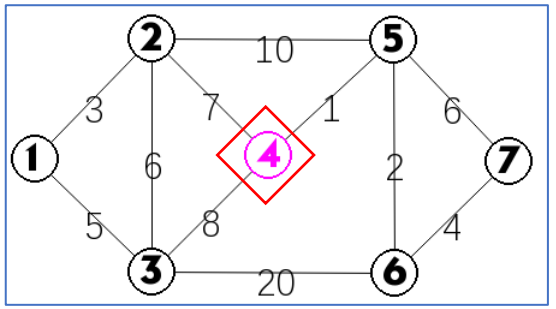
Dijkstra算法

Dijkstra算法：

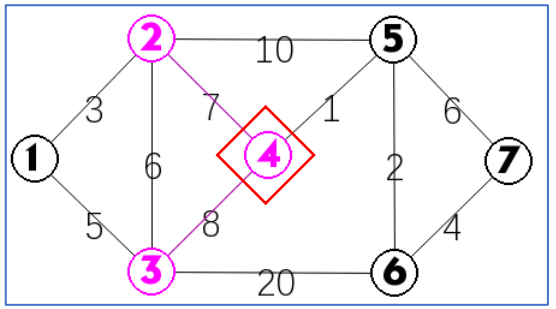
- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	13	25	999

- ……
- 找到最小的dis[4]=10，打个标记；



Dijkstra算法



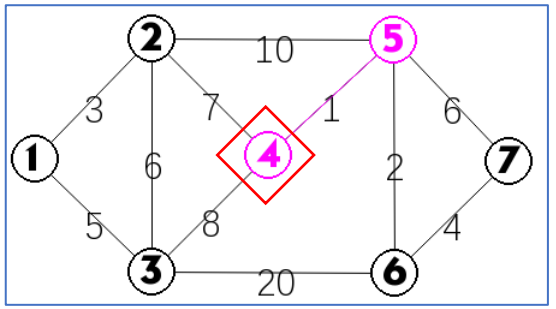
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	13	25	999

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新

Dijkstra算法



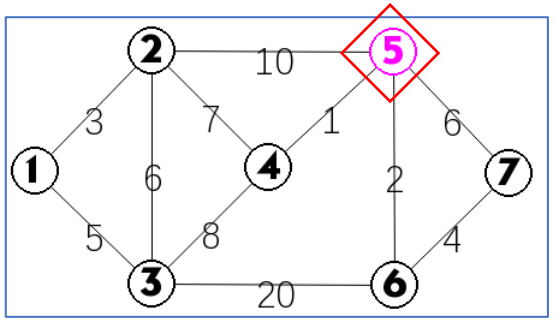
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	11	25	999

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11

Dijkstra算法



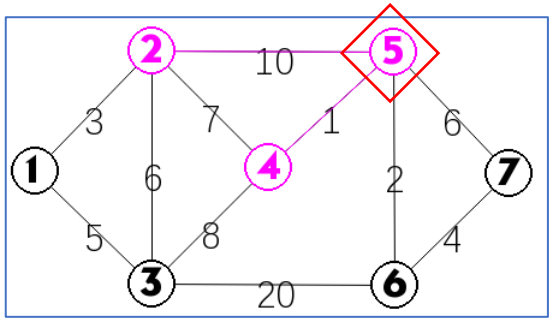
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	25	999

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；

Dijkstra算法



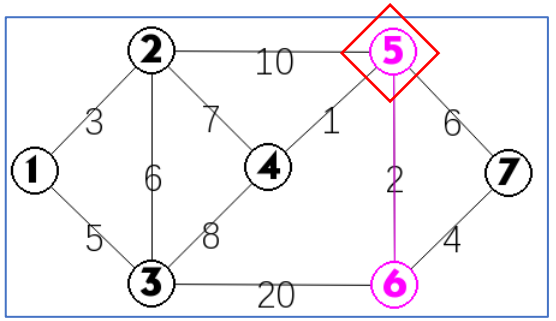
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	13	999

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新

Dijkstra算法



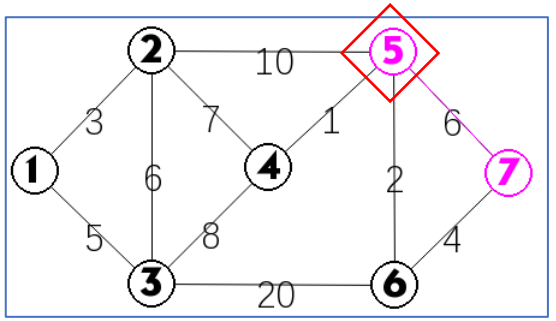
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	13	999

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13

Dijkstra算法



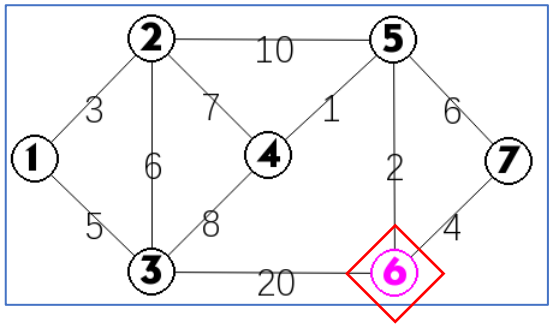
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	13	17

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17

Dijkstra算法



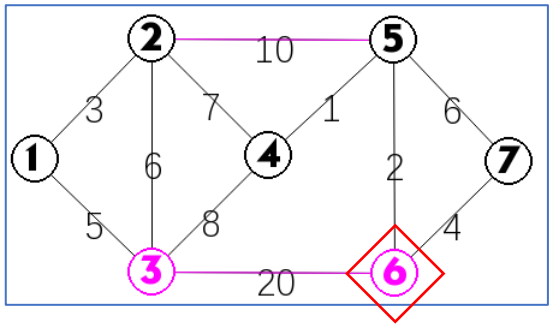
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	**13**	17

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17
- 找到最小的dis[6]=13，打个标记；

Dijkstra算法



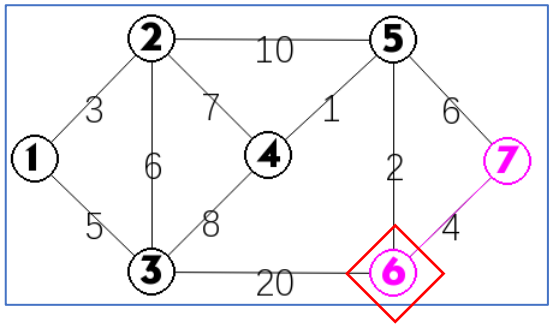
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	**13**	17

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17
- 找到最小的dis[6]=13，打个标记；然后更新与4相邻节点的dis
  - dis[3]已被标记，不更新

Dijkstra算法



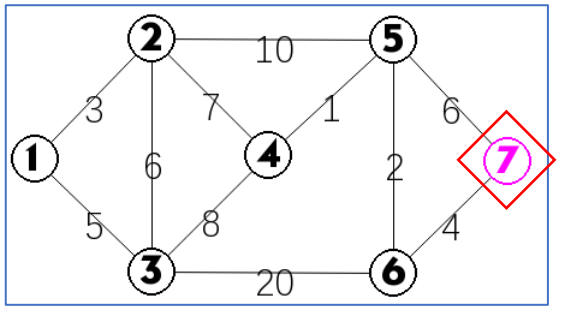
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	**13**	17

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17
- 找到最小的dis[6]=13，打个标记；然后更新与4相邻节点的dis
  - dis[3]已被标记，不更新
  - dis[7] = min(dis[7], dis[6] + 4) = 17

Dijkstra算法



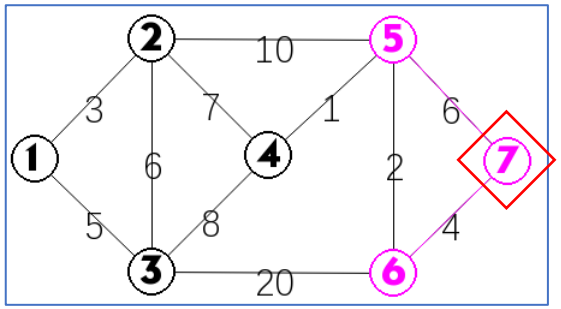
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	**13**	**17**

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17
- 找到最小的dis[6]=13，打个标记；然后更新与4相邻节点的dis
  - dis[3]已被标记，不更新
  - dis[7] = min(dis[7], dis[6] + 4) = 17
- 找到最小的dis[7]=17，打个标记；

Dijkstra算法



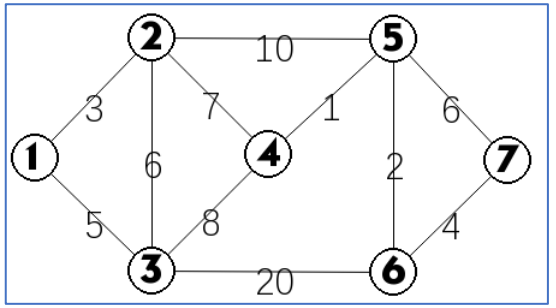
Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	**13**	**17**

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17
- 找到最小的dis[6]=13，打个标记；然后更新与4相邻节点的dis
  - dis[3]已被标记，不更新
  - dis[7] = min(dis[7], dis[6] + 4) = 17
- 找到最小的dis[7]=17，打个标记；然后更新与4相邻节点的dis
  - dis[5]和dis[6]已被标记，不更新

Dijkstra算法



Dijkstra算法：

- 计算从起点（节点1）到其他所有点的最短路。

	1	2	3	4	5	6	7
dis[]	**0**	**3**	**5**	**10**	**11**	**13**	**17**

- ……
- 找到最小的dis[4]=10，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[3]已被标记，不更新
  - dis[5] = min(dis[5], dis[4] + 1) = 11
- 找到最小的dis[5]=11，打个标记；然后更新与4相邻节点的dis
  - dis[2]和dis[4]已被标记，不更新
  - dis[6] = min(dis[6], dis[5] + 2) = 13
  - dis[7] = min(dis[7], dis[5] + 6) = 17
- 找到最小的dis[6]=13，打个标记；然后更新与4相邻节点的dis
  - dis[3]已被标记，不更新
  - dis[7] = min(dis[7], dis[6] + 4) = 17
- 找到最小的dis[7]=17，打个标记；然后更新与4相邻节点的dis
  - dis[5]和dis[6]已被标记，不更新
- 算法结束

Dijkstra算法

算法流程：

- 初始化：dis[起点]=0，dis[其他]=∞
- 找出与起点最近的未标记点u，将u标记
- 枚举与u相邻的点v，如果从起点经过u到v路径更短，则更新
- 重复n次，直到所有点都被标记

Dijkstra算法

算法流程：

- 初始化：dis[起点]=0，dis[其他]=∞
- 找出与起点最近的未标记点u，将u标记
- 枚举与u相邻的点v，如果从起点经过u到v路径更短，则更新
- 重复n次，直到所有点都被标记

```
//Dijkstra - NKOJ1120 - 最短路默写
const int INF = 999999999;
const int MAX_N = 100 + 5;
int N, M, A, B;
int edge[MAX_N][MAX_N];
int dis[MAX_N];
bool mark[MAX_N];
int main() {
    scanf("%d%d", &N, &M);
    for (int i = 1; i <= N; i ++) {
        for (int j = 1; j <= M; j ++) {
            edge[i][j] = INF;
        }
    }
    for (int i = 1; i <= M; i ++) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        if (edge[x][y] > z) {
            edge[x][y] = z;
            edge[y][x] = z;
        }
    }
    scanf("%d%d", &A, &B);
    for (int i = 1; i <= N; i ++) {
        dis[i] = INF;
        mark[i] = false;
    }
    dis[A] = 0;
    for (int i = 1; i <= N; i ++) {
        int k = -1;
        for (int j = 1; j <= N; j ++) {
            if (!mark[j] && (k == -1 || dis[j] < dis[k])) {
                k = j;
            }
        }
        mark[k] = true;
        if (k == B) {
            break;
        }
        for (int j = 1; j <= N; j ++) {
            dis[j] = min(dis[j], dis[k] + edge[k][j]);
        }
    }
    printf("%d\n", dis[B]);
    return 0;
}
```

Dijkstra算法

算法流程：

- 初始化：dis[起点]=0，dis[其他]=∞
- 找出与起点最近的未标记点u，将u标记
- 枚举与u相邻的点v，如果从起点经过u到v路径更短，则更新
- 重复n次，直到所有点都被标记

效率？

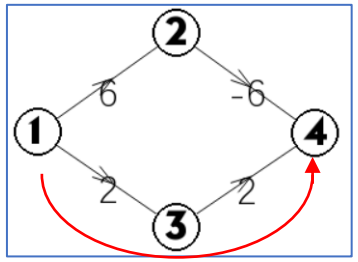
- 每次找到未标记的最小\dis[i]
  - 共找n次，时间复杂度$O(n^2)$
- 每条有向边会被枚举1次，无向边枚举2次
  - 时间复杂度$O(m) \leq O(n^2)$
- 总时间复杂度$O(n^2)$

```
//Dijkstra - NKOJ1120 - 最短路默写
const int INF = 999999999;
const int MAX_N = 100 + 5;
int N, M, A, B;
int edge[MAX_N][MAX_N];
int dis[MAX_N];
bool mark[MAX_N];
int main() {
    scanf("%d%d", &N, &M);
    for (int i = 1; i <= N; i ++) {
        for (int j = 1; j <= M; j ++) {
            edge[i][j] = INF;
        }
    }
    for (int i = 1; i <= M; i ++) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        if (edge[x][y] > z) {
            edge[x][y] = z;
            edge[y][x] = z;
        }
    }
    scanf("%d%d", &A, &B);
    for (int i = 1; i <= N; i ++) {
        dis[i] = INF;
        mark[i] = false;
    }
    dis[A] = 0;
    for (int i = 1; i <= N; i ++) {
        int k = -1;
        for (int j = 1; j <= N; j ++) {
            if (!mark[j] && (k == -1 || dis[j] < dis[k])) {
                k = j;
            }
        }
        mark[k] = true;
        if (k == B) {
            break;
        }
        for (int j = 1; j <= N; j ++) {
            dis[j] = min(dis[j], dis[k] + edge[k][j]);
        }
    }
    printf("%d\n", dis[B]);
    return 0;
}
```

Dijkstra算法

思考：

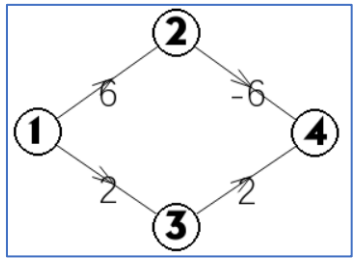
- 前提条件？
  - 边权非负
  - 如果边权有负数，Dijkstra可能算出错误的答案
  - 例如右图，起点为1，算出1→3→4的路径后就给dis[4]打标记
  - 这种情况下，用其他的最短路算法



Dijkstra算法

思考：

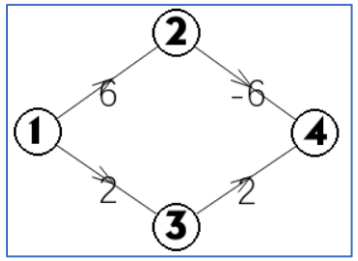
- 前提条件？
  - 边权非负
  - 如果边权有负数，Dijkstra可能算出错误的答案
  - 例如右图，起点为1，算出1→3→4的路径后就给dis[4]打标记
  - 这种情况下，用其他的最短路算法
- 打标记的意义？
  - 标记dis[i]表示它的数值已确定
  - 事实上，即使不打标记，dis[i]也不会再被更新
  - 标记的作用是每次找dis最小的节点时避免重复。



Dijkstra算法

思考：

- 前提条件？
  - 边权非负
  - 如果边权有负数，Dijkstra可能算出错误的答案
  - 例如右图，起点为1，算出1→3→4的路径后就给dis[4]打标记
  - 这种情况下，用其他的最短路算法
- 打标记的意义？
  - 标记dis[i]表示它的数值已确定
  - 事实上，即使不打标记，dis[i]也不会再被更新
  - 标记的作用是每次找dis最小的节点时避免重复。
- 提前结束？
  - 如果需要计算的终点只有1个
  - 当dis[终点]被标记时，算法可以提前结束。



Dijkstra算法

思考：

- 打印路径？
- 再开一个prev数组
  - prev[i]是从起点到i，距离为dis[i]的路径中，上一个节点的编号
  - 设prev[起点]=0
  - 每次更新dis[i]的同时也修改prev[i]的数值
- 倒序打印路径上每个节点的编号：
  - for (i = 终点; i ; i = prev[i]) cout << i << " ";
  - 需要顺序输出就再颠倒一下

Dijkstra算法优化

空间优化：

- 当点数n和边数m都比较大时，n×n的邻接矩阵不再适用
- 改用链表存边的方式：

```
struct Edge {
    int x, y, z, next;
} e[MAX_M];
int elast[MAX_N];
```

- 读入M条有向边：

```
for (int i = 1; i <= M; i ++) {
    scanf("%d%d%d", &e[i].x, &e[i].y, &e[i].z);
    e[i].next = elast[e[i].x];
    elast[e[i].x] = i;
}
```

- 无向边？

- 空间复杂度$O(n+m)$

Dijkstra算法优化

时间优化：

- Dijkstra时间复杂度$O(n^2+m)$
- 当n和m都比较大时，瓶颈在找未标记的最小\dis[i]
- 用堆进行优化。

Dijkstra+堆

怎么写？

- 对于Dijkstra来说
  - 在堆里，比较大小的关键字是dis[i]的值，小根堆
  - 从堆中取出最小元素后，需要节点编号i
  - 方法1：pair小根堆

```
priority_queue<pair<int,int>, vector<pair<int,int> >, greater<pair<int,int> > > pq;
```

  - 方法2：结构体小根堆

```
struct PQNode { int dis,i; }; //定义结构体
bool operator <(PQNode a, PQNode b){ return a.dis > b.dis; } //重载<运算符
priority_queue<PQNode> pq; //声明优先队列
```

  - 方法3：pair添负号（不推荐，容易写错）

```
priority_queue<pair<int,int> > pq;
插入改成：pq.push(make_pair(-dis[i],i))
```

- 时间复杂度
  - 每条有向边至多会调用pq.push一次（无向边两次）
  - 堆的操作次数为$O(m)$，所以总时间复杂度$O(m \log m)$

堆(Heap)

- Dijkstra+堆？

```
//Dijkstra - NKOJ3639 - 最短路默写2
const long long INF = 9999999999999999ll;
const int MAX_N = 400000 + 5;
const int MAX_M = 4000000 + 5;
int N, M, A, B;

struct Edge {
    int x, y, z, next;
} e[MAX_M];
int elast[MAX_N];

struct PQNode {
    int u;
    long long dis;
    PQNode(int u_, long long dis_) {
        u = u_;
        dis = dis_;
    }
};
bool operator < (PQNode a, PQNode b) {
    return a.dis > b.dis;
}

long long dis[MAX_N];
bool mark[MAX_N];
```

```
int main() {
    scanf("%d%d", &N, &M);
    for (int i = 1, j = 1; i <= M; i ++) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        e[j].x = x;
        e[j].y = y;
        e[j].z = z;
        e[j].next = elast[x];
        elast[x] = j ++;
        e[j].x = y;
        e[j].y = x;
        e[j].z = z;
        e[j].next = elast[y];
        elast[y] = j ++;
    }
    scanf("%d%d", &A, &B);
    for (int i = 1; i <= N; i ++) {
        dis[i] = INF;
        mark[i] = false;
    }
    dis[A] = 0;
    priority_queue<PQNode> pq;
    pq.push(PQNode(A, 0));
```

```
    while (!pq.empty()) {
        int u = pq.top().u;
        pq.pop();
        if (mark[u]) {
            continue;
        }
        if (u == B) {
            break;
        }
        mark[u] = true;
        for (int j = elast[u]; j; j = e[j].next) {
            int v = e[j].y;
            if (dis[v] > dis[u] + e[j].z) {
                dis[v] = dis[u] + e[j].z;
                pq.push(PQNode(v, dis[v]));
            }
        }
    }
    printf("%lld\n", dis[B]);
    return 0;
}
```