数 据 结 构

第八章 图 [3] Dijkstra算法

任课老师：郭艳

数据结构课程组

计算机学院

中国地质大学（武汉）2020年秋

第十八次课

阅读：

殷人昆，第371-375页

习题：

作业18

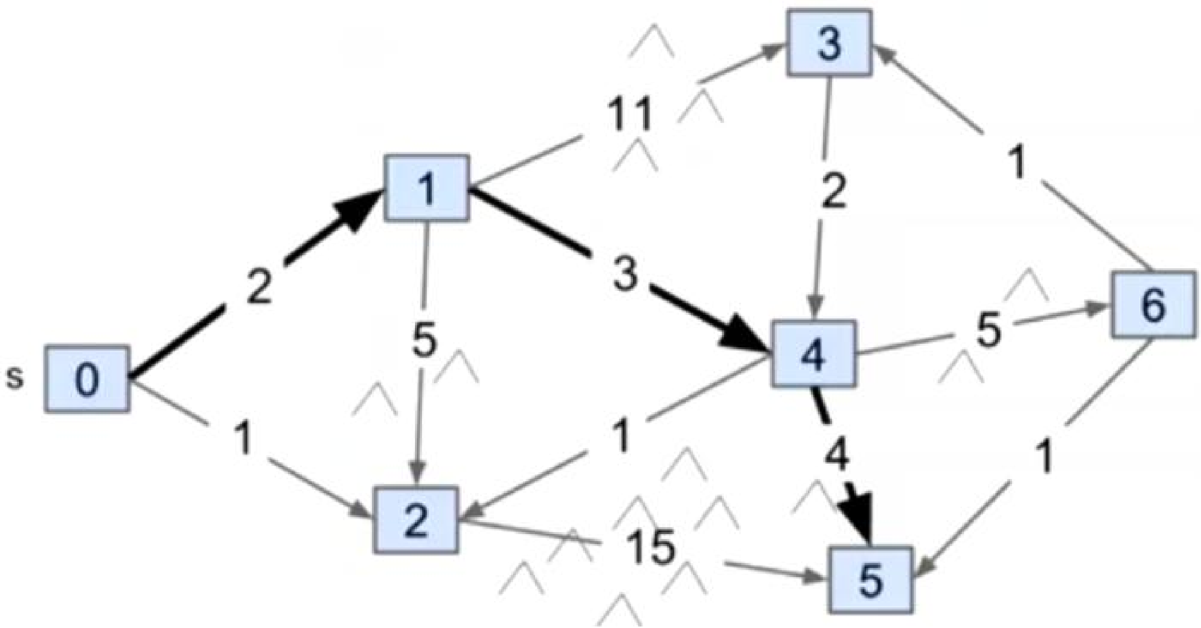
8.5 最短路径

- **什么是最短路径?**

例：假如用顶点表示城市，用边表示城市间的公路，则由这些顶点和边组成的图可以表示沟通各城市的公路网。若把两个城市之间的距离或该段公路的养路费等作为权值，赋给图中的边，就构成一个带权的图。

定义：从源点到终点所经过的边上的权值之和（简称距离）为最小的路径。

问题：一个源点到一个终点的最短路径



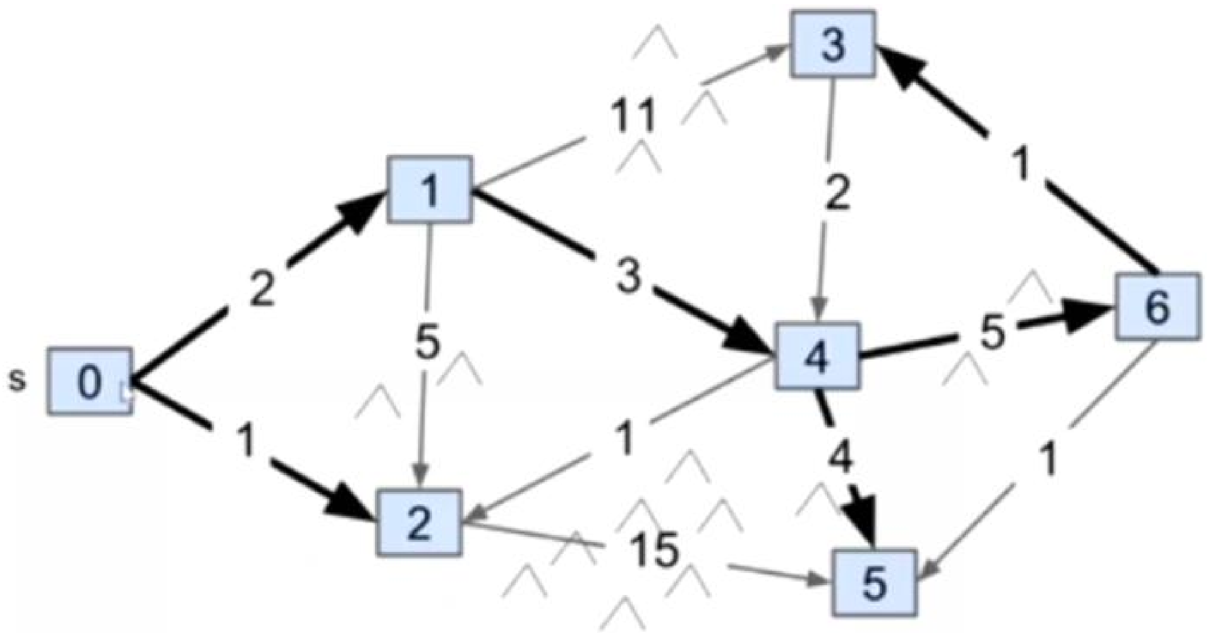
结果是一个无环的路径

从顶点0到顶点5的
最短路径

v	dist[]	path[]
0	0	-1
1	2	0
2	-	-
3	-	-
4	5	1
5	9	4
6	-	-

注：
1、dist[]存放最短路径距离
2、path[]存放最短路径上顶点的直接前驱下标
3、“0到5的最短路径”上有“0到4的最短路径”和“0到1的最短路径”

问题：单源最短路径



从顶点0到每个结点的最短路径

v	dist[]	path[]
0	0	-1
1	2	0
2	1	0
3	11	6
4	5	1
5	9	4
6	10	4

结果是一棵树（最短路径树）

可以认为是源点到所有顶点的最短路径的集合

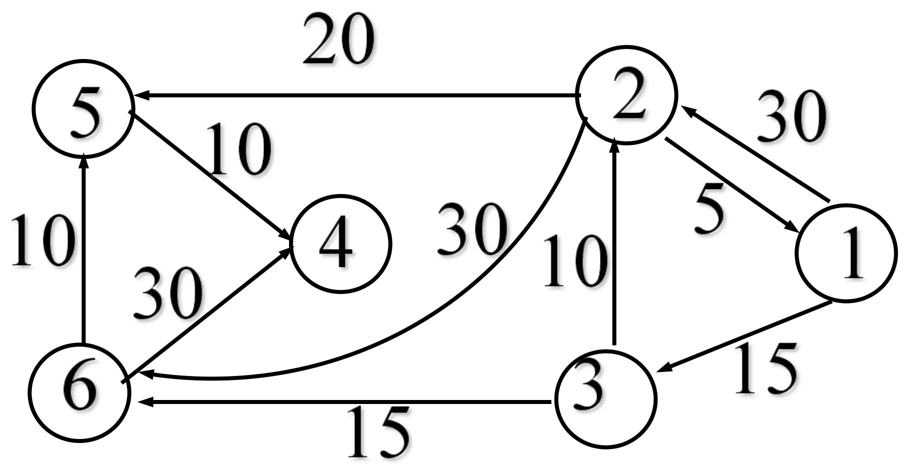
注：最短路径树与最小生成树不同，最短路径树上只有一个源点。最小生成树不一定。

最短路径问题

- 8.5.1 非负权值单源最短路径（single-source shortest paths）问题
  - 指的是对已知图G=（V，E），给定源顶点v0∈V，找出v0到图中其它各顶点的最短路径
- 8.5.3 每对顶点间的最短路径（all-pairs shortest paths）问题
  - 指的是对已知图G=（V，E），任意的顶点V_i，V_j∈V，找出从V_i到V_j的最短路径

■ 8.5.1 单源最短路径问题

例：



$V_1 \rightarrow V_2$： $<V_1, V_2>$ 30

$<V_1, V_3, V_2>$ 25 （最短路径）

$V_1 \rightarrow V_3$： $<V_1, V_3>$ 15 （最短路径）

……

8.5.1 Dijkstra算法

- **Dijkstra(迪杰斯特拉)算法基本思想**
  - **把图中所有顶点分成两组**
    - 第一组S包括已确定最短路径的顶点
    - 第二组包括尚未确定最短路径的顶点
  - **按路径长度递增的顺序逐个把第二组的顶点加到第一组中去**
    - 直至从v_0出发可以到达的所有顶点都包括进第一组中

8.5.1 Dijkstra算法（续）

- **在这个过程中，总保持从v_0到第一组S各顶点的最短距离都不大于从v_0到第二组的任何顶点的最短距离，而且每个顶点都对应一个距离值：**
  - 第一组S的顶点对应的距离值就是从v_0到该顶点的最短距离
  - 第二组的顶点对应的距离值是从v_0到该顶点的只包括第一组的顶点为中间顶点的最短距离

8.5.1 Dijkstra算法（续）

- **Dijkstra算法的具体做法：**
  - 一开始第一组S只包括顶点v_0，第二组包括其它所有顶点；
  - v_0对应的距离值为0，而第二组的顶点对应的距离值这样确定：
    - 若图中有边<v_0，v_i>或者（v_0，v_i），则v_i的距离值为此边所带的权值，否则v_i的距离值为∞。
  - 然后，每次从第二组的顶点中选一个其距离值为最小的顶点v_u加入到第一组中；

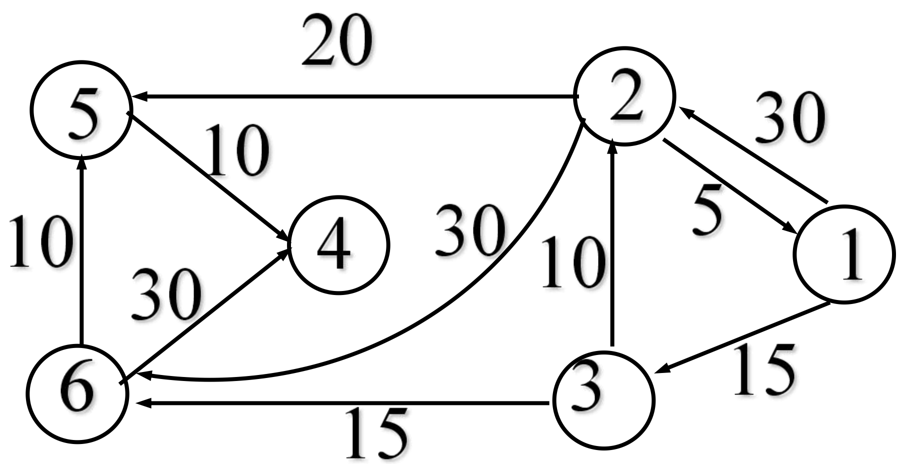
8.5.1 Dijkstra算法（续）

- 每往第一组加入一个顶点v_u，就要对第二组的各顶点的距离值进行一次修正：
  - 若加进v_u做中间顶点，使从v_0到v_i的最短路径比不加v_u的为短，则需要修改v_i的距离值。
- 修正后再选距离值最小的顶点加入到第一组中。
- 如此进行下去，直到图的所有顶点都包括在第一组中或者再也没有可加入到第一组的顶点存在。

Dijkstra算法的设计

——按**路径长度递增**的次序产生最短路径

有向网G采用邻接矩阵存储（为方便讲解）



$$Edge=\begin{pmatrix} 0 & 30 & 15 & \infty & \infty & \infty \\ 5 & 0 & \infty & \infty & 20 & 30 \\ \infty & 10 & 0 & \infty & \infty & 15 \\ \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 10 & 0 & \infty \\ \infty & \infty & \infty & 30 & 10 & 0 \end{pmatrix}$$

Dijkstra算法的设计

(1) 引入一维数组**S[i]**($0\le i<n$)存放顶点v_i是否已求得最短路径，S[i]=false表示v_i未求得最短路径；S[i]=true表示v_i已求得最短路径。

(2) 引入一维数组dist[j]($0 \le j<n$) 存放当前求出的从源点v到终点v_j的最短路径距离。初始，dist[i]=G.GetWeight(v,i)。

(3)引入一维数组path[i]($0 \le i<n$)存放从源点v到各终点v_i的最短路径上v_i的直接前驱结点下标。若从v到v_i无路径，则path[i]=-1。

Dijkstra算法描述

设源点序号v，顶点数为n。图G是Graph类型

1.初始S[i]=false, dist[i]=G.GetWeight(v,i), path[i]=v或-1（i: 0→n−1）。

2. S[v]=true

3. 重复执行下列步骤n−1次，直至所有顶点都包含在S中：

①在所有不在S集合的顶点中，选取dist[i]为最小的一个顶点，设为第u个顶点；

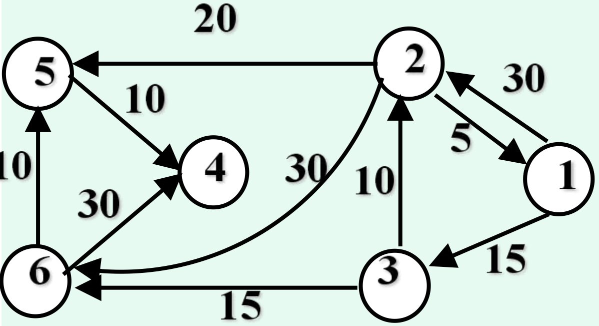
②将选出的第u个顶点并入S集合中，即S[u]=true。

③以u作为新考虑的中间点，修改不在S集合的各顶点dist[k]值：

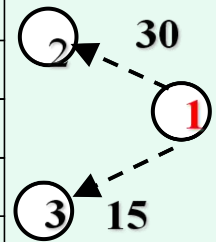
若dist[u]+G.GetWeight(u,k)<dist[k]

则令dist[k]=dist[u]+G.GetWeight(u,k)，同时记下此路径，path[k]=u。

- 算法时间复杂度：$O(n^2)$

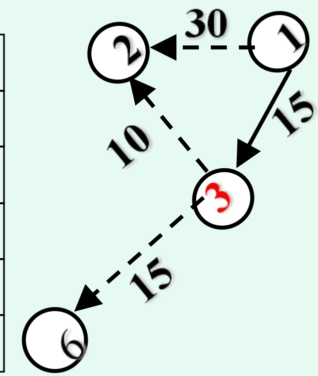


	distance	path	S
0	0	−1	1
1	30	0	0
2	15	0	0
3	∞	−1	0
4	∞	−1	0
5	∞	−1	0



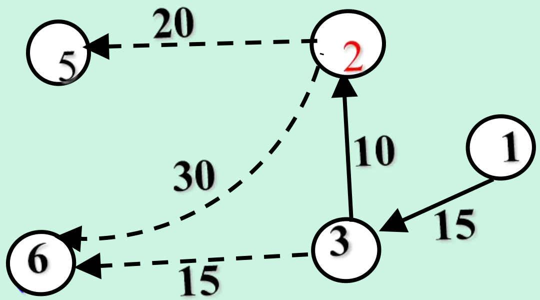
(a) v1→v2(30),v1→v3(15)

	distance	path	S
0	0	−1	1
1	25	2	0
2	15	0	1
3	∞	−1	0
4	∞	−1	0
5	30	2	0



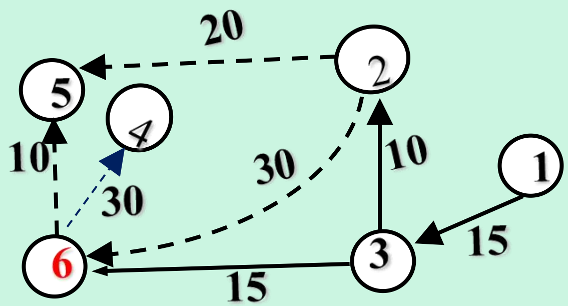
(b) v1→v3(15),v1→v3→v2(25), v1→v3→v6(30)

	distance	path	S
0	0	−1	1
1	25	2	1
2	15	0	1
3	∞	−1	0
4	45	1	0
5	30	2	0



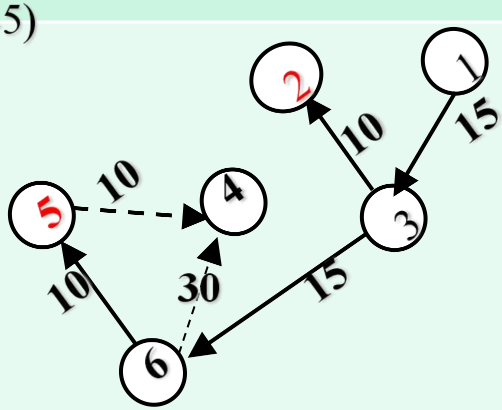
(c) v1→v3(15),v1→v3→v2(25),v1→v3→v6(30),v1→v3→v2→v5(45)

	distance	path	S
0	0	−1	1
1	25	2	1
2	15	0	1
3	60	5	0
4	40	5	0
5	30	2	1



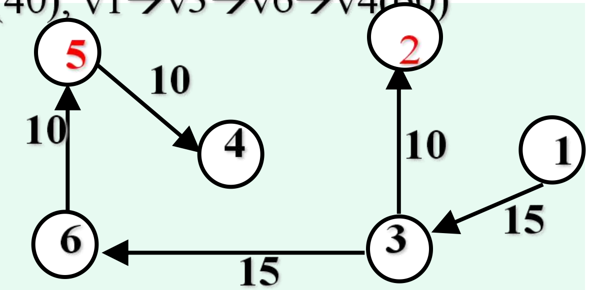
(d) v1→v3(15),v1→v3→v2(25),v1→v3→v6(30),v1→v3→v6→v5(40), v1→v3→v6→v4(60)

	distance	path	S
0	0	−1	1
1	25	2	1
2	15	0	1
3	50	4	0
4	40	5	1
5	30	2	1



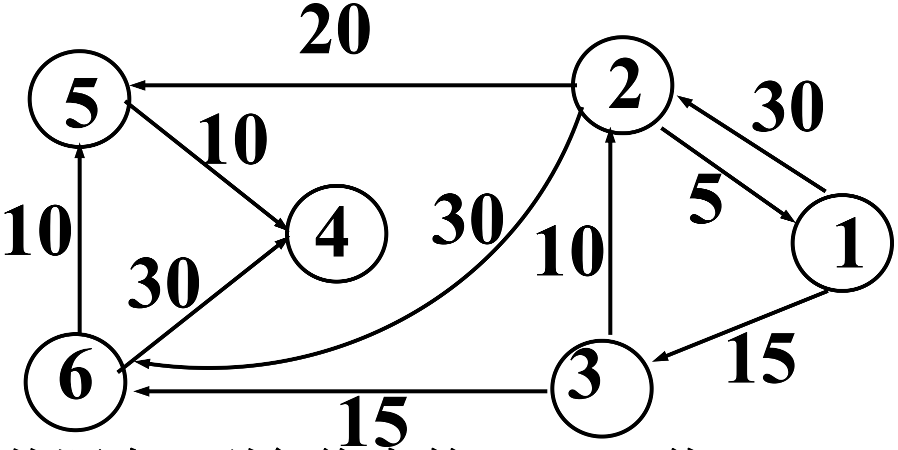
(e) v1→v3(15),v1→v3→v2(25),v1→v3→v6(30),v1→v3→v6→v5(40), v1→v3→v6→v5→v4(50)

	distance	path	S
0	0	−1	1
1	25	2	1
2	15	0	1
3	50	4	1
4	40	5	1
5	30	2	1



(f) v1→v3(15),v1→v3→v2(25),v1→v3→v6(30),v1→v3→v6→v5(40), v1→v3→v6→v5→v4(50)

完成！



终点	从源点v_1到各终点的distance值				
v_2	$30<_{1,2}>$ path[1]=0	$25<_{1,3,2}>$ path[1]=2			
v_3	$15<_{1,3}>$ path[2]=0				
v_4	∞ Path[3]=-1	∞ Path[3]=-1	∞ Path[3]=-1	$60<_{1,3,6,4}>$ path[3]=5	$50<_{1,3,6,5,4}>$ path[3]=4
v_5	∞ Path[4]=-1	∞ Path[4]=-1	$45<_{1,3,2,5}>$ path[4]=1	$40<_{1,3,6,5}>$ path[4]=5	
v_6	∞ Path[5]=-1	$30<_{1,3,6}>$ path[5]=2	$30<_{1,3,6}>$ path[5]=2		
v_u	v_3	v_2	v_6	v_5	v_4

【Dijkstra算法】

```
void ShortestPath (Graph<T, E>& G, int v, E dist[], int path[])
  {/*Graph是一个带权有向图。dist[j], 0≤j<n, 是当前求到的从顶点
  v到顶点j的最短路径长度, path[j],0≤j<n, 存放求到的最短路径。*/
  int n = G.NumberOfVertices();
  bool *S = new bool[n];          //最短路径顶点集
  int i, j, k;  E w, min;
  for (i = 0; i < n; i++) {
     dist[i] = G.getWeight(v, i);
     S[i] = false;
     if (i != v && dist[i] < maxValue) path[i] = v;
     else path[i] = -1;
  }
 S[v] = true;  dist[v] = 0;       //顶点v加入顶点集合
```

【Dijkstra算法(续)】

时间开销=找最短距离顶点+修正检查

找最短距离顶点=n²-n次

修正检查=n²-n次（邻接矩阵）=n²-n次（邻接表，程序改写为getFirstNeighbor/getNextNeighbor循环组合后，为2e次）

时间开销 $O(n^2)$

```
for (i = 0; i < n-1; i++)   //求解各顶点最短路径，n-1次循环
{   min = maxValue;  int u = v; //选不在S中具有
    for (j = 0; j < n; j++)   //最短路径的顶点u，n次循环
      if (!S[j] && dist[j] < min) {u = j; min = dist[j]; }
    S[u] = true;        //将顶点u加入集合S
    for (k = 0; k < n; k++) {  //修改，n次循环
      w = G.GetWeight(u, k);
      if (!S[k] && w < maxValue &&dist[u]+w < dist[k])
      {   dist[k] = dist[u]+w; //顶点k未加入S，且找到更短路径
          path[k] = u;        //修改k的直接前驱为u
      }
    }  //end of for(k=0;...
}      //end of for(i=0;..
};     //end of void
```

Dijkstra算法性能分析

- 时间代价模型=找离源点最短距离顶点+修正检查
  - 寻找离源点最短距离顶点：n-1趟循环，每趟进行n次扫描dist数组。$O(n^2)$
  - 修正检查、更新dist值：$O(n^2)$
    - 图是邻接矩阵表示，总共扫描n^2-n次。
    - 图是邻接表表示，总共扫描n^2-n次。
  - 所以本方法的总时间代价为$O(n^2)$
- 空间开销：使用S、path、dist数组。O(n)

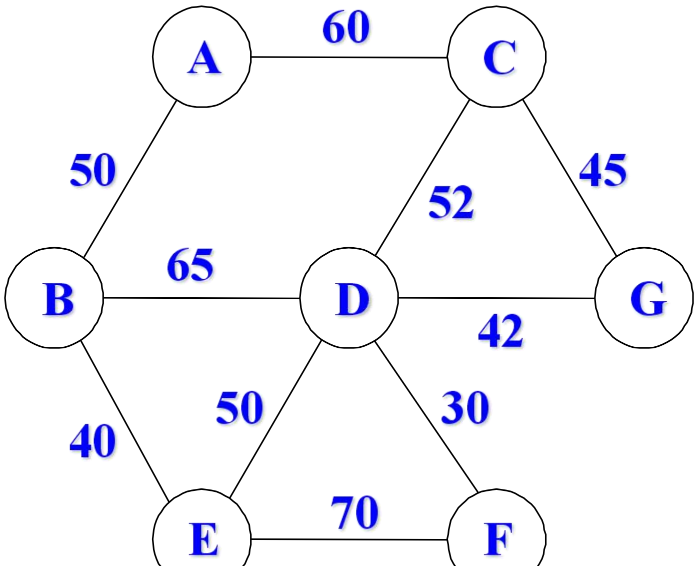
Dijkstra算法优化

优化

- 优化策略1：dist数组使用小顶堆
- 优化策略2：修正程序优化，循环部分改写为getFirstNeighbor/getNextNeighbor循环组合，修正堆中相应顶点的最小距离值
- 时间代价模型=加入S检查+入堆+出堆（找离源点最近顶点）+修正+加入S+得到所有顶点的邻居
- 时间开销：O(n+nlogn+nlogn+elogn+n+...)= { O(elogn+n^2)(邻接矩阵) ; O(elogn+n)(邻接表) (设e>n) }
  - 加入S检查：O(n)
  - 入堆：O(nlogn)
  - 出堆(找与源点距离最小的一个顶点)：O(nlogn)
  - 修正堆中源点与各顶点的当前最小距离：O(elogn)
  - 加入S：O(n)
  - **得到所有顶点的邻居：**
    - 图是邻接矩阵表示，总共扫描n^2-n次。O(n^2)
    - 图是邻接表表示，总共扫描e次，每条边检查2次。O(e)
- 空间开销：使用S数组、Path数组、H堆(n个分量）。O(n)

Dijkstra算法与Prim算法

- 优化后的Dijkstra算法与优化后的Prim算法十分相似
- 优化后的Dijkstra算法与优化后的Prim算法的不同之处
  - 唯一不同点在于一个值的修改不同
    - Dijkstra： dist[k] = dist[u] + G.GetWeight(u, k);
      - Dijkstra算法是在未找到最短路径的顶 点集合中寻找离固定顶点v_0距离最近的顶点（要看 若干条边）
    - 优化后的Prim： dist[k] = G.GetWeight(u, k);
      - Prim算法是要在未加入V(MST)的顶点集合中寻找 离已加入V(MST)的顶点距离最近的顶点（只看一 条边）
- 所以，优化后的Dijkstra算法时间复杂度分析与优化后的Prim算法相同



(a)G

广优

	A	B	C	D	E	F	G
A	0	50	60	∞	∞	∞	∞
B	50	0	∞	65	40	∞	∞
C	60	∞	0	52	∞	∞	45
D	∞	65	52	0	50	30	42
E	∞	40	∞	50	0	70	∞
F	∞	∞	∞	30	70	0	∞
G	∞	∞	45	42	∞	∞	0

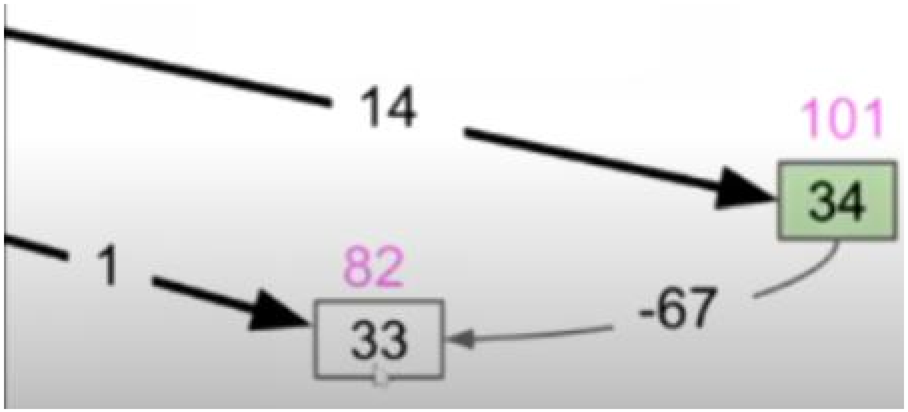
深优

	A	B	C	D	E	F	G
A	0	50	60	∞	∞	∞	∞
B	50	0	∞	65	40	∞	∞
C	60	∞	0	52	∞	∞	45
D	∞	65	52	0	50	30	42
E	∞	40	∞	50	0	70	∞
F	∞	∞	∞	30	70	0	∞
G	∞	∞	45	42	∞	∞	0

$\boxed{\text{？ Dijkstra}}$

负权值图

Dijkstra算法对负权值图无效



遍历应用算法性能比较

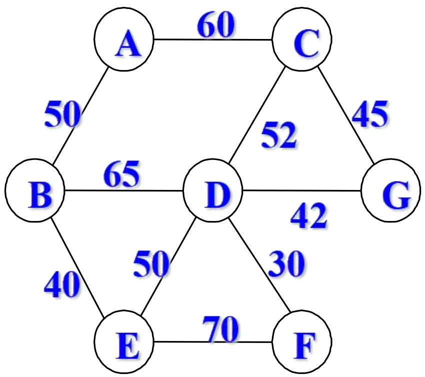
设n为图的顶点数，e为边数，

算法	时间复杂度(邻接矩阵表示)	时间复杂度(邻接表表示)	空间复杂度
DFS	$O(n^2)$	O(n+e)	O(n)
BFS	$O(n^2)$	O(n+e)	O(n)
优化Prim	$O(n^2+elogn)$	O(n+elogn)	O(n)
优化Kruskal	$O(n^2+eloge)$	O(n+eloge)	O(n+e)
优化Dijkstra	$O(n^2+elogn)$	O(n+elogn)	O(n)

作业18——最短路径

概念题

- 3、补充题：画出Dijkstra算法的流程图
- 4、补充题：下图采用邻接矩阵存储，写出使用Dijkstra算法求源点A到其余各顶点的最短路径的过程和结果，并画出邻接矩阵的扫描过程。



电子作业

- 5、（选作）使用最小堆优化Dijkstra算法