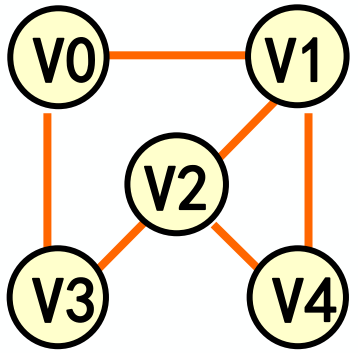
6.6 图的应用

- 最小生成树
- 最短路径
- 拓扑排序
- 关键路径

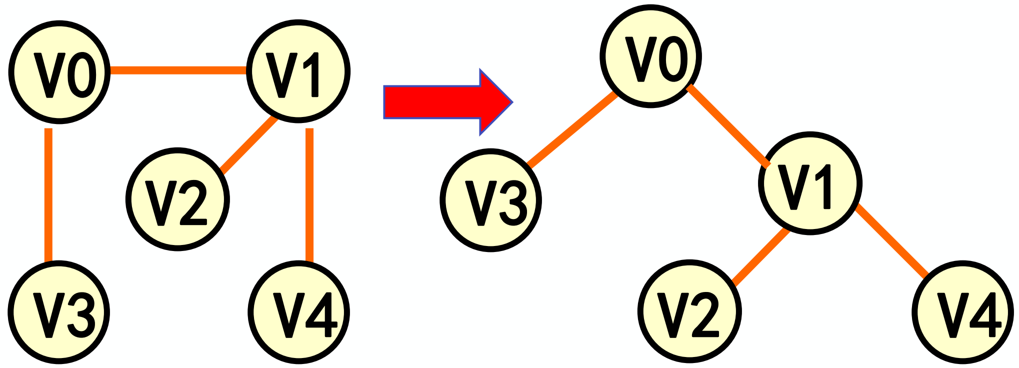
最小生成树

极小连通子图：该子图是G 的连通子图，在该子图中删除任何一条边，子图不再连通。

生成树：包含图G所有顶点的极小连通子图（n-1条边）。

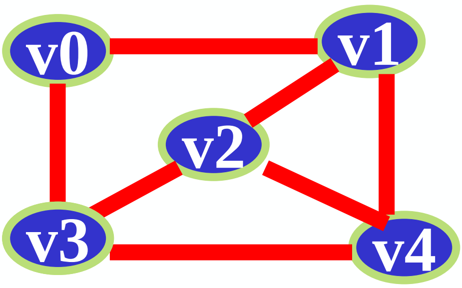


连通图 G1

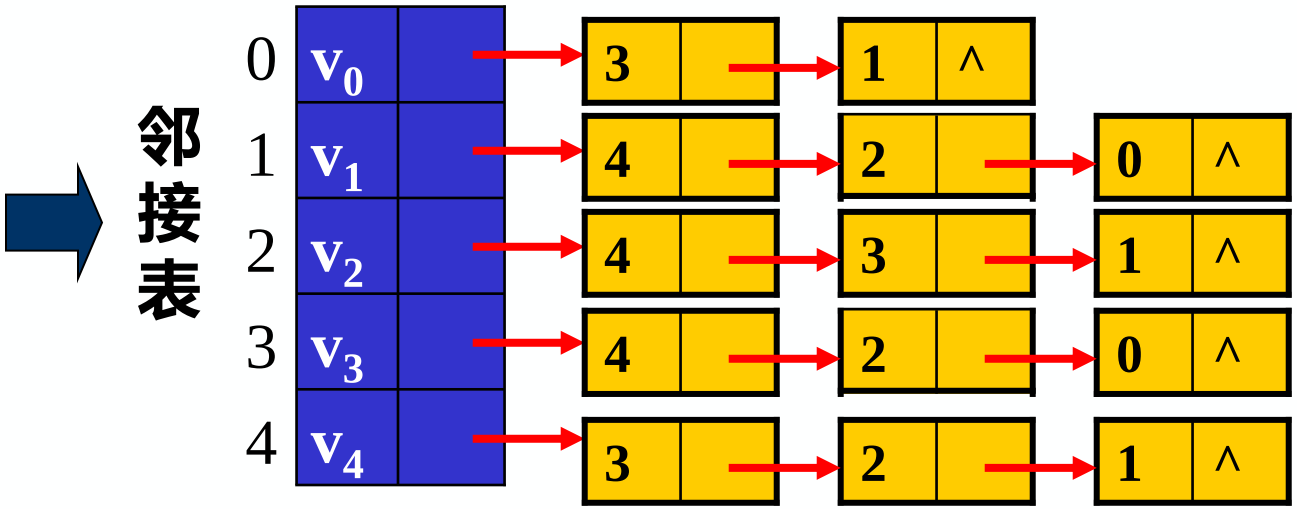


G1的生成树

画出下图的生成树

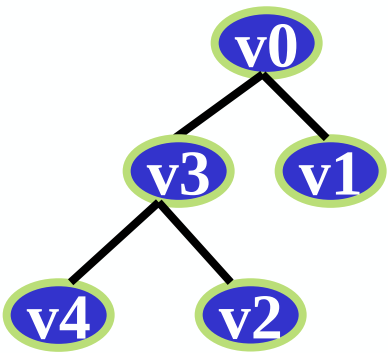


无向连通图



DFS生成树

BFS生成树



2020年6月5日

求最小生成树

首先明确：

- **使用不同的遍历图的方法，可以得到不同的生成树**
- **从不同的顶点出发，也可能得到不同的生成树。**
- **按照生成树的定义，*n* 个顶点的连通网络的生成树有 *n* 个顶点、*n*-1 条边。**

目标：

在网的多个生成树中，寻找一个各边权值之和最小的生成树

构造最小生成树的准则

- 必须只使用该网中的边来构造最小生成树；
- 必须使用且仅使用n-1条边来联结网络中的n个顶点
- 不能使用产生回路的边

最小生成树的典型用途

欲在n个城市间建立通信网，则n个城市应铺n-1条线路；但因为每条线路都会有对应的经济成本，而n个城市可能有n(n-1)/2 条线路，那么，如何选择n–1条线路，使总费用最少?

显然此连通网是一个生成树！

数学模型：

顶点———*表示城市，有n个；*

边————*表示线路，有n–1条；*

边的权值—*表示线路的经济代价；*

连通网——*表示n个城市间通信网。*

补充：贪心算法(Greedy Algorithm)

算法原理：以当前情况为基础作最优选择，而不考虑各种可能的整体情况，所以贪心法不要回溯。

算法优点：因为省去了为寻找解而穷尽所有可能所必须耗费的大量时间，因此算法效率高。

注意：贪婪算法的精神就是“只顾如何获得眼前最大的利益”,有时不一定是最优解。

找零钱问题。

平时购物找钱时，为使找回的零钱的硬币数最少，不考虑找零钱的所有各种发表方案，而是从最大面值的币种开始，按递减的顺序考虑各币种，先尽量用大面值的币种，当不足大面值币种的金额时才去考虑下一种较小面值的币种。

这就是在使用贪心法。

Knapsack问题

- 一个小偷,背着一个可载重W公斤的背包行窃.店内有数种不同的商品,不同商品有不同的重量及价值.考虑商品可以分割的情形(例如,可以只取1/2或1/3个商品).
- 目的在于求出如何偷到最大利益价值的商品.

- 假设小偷的背包可装30公斤的物品
- 假设商品价格/重量如下:

物品	價值	重量	单价
1	50	5	10
2	60	10	6
3	140	20	7

1 以贪婪算法的观念来看,第一步要找到最佳效益的商品.

2 我们知道,物品1最划算,故5公斤全放入背包.(背包还可以装25公斤)

3 再来考虑物品3,一样全部放入背包中.(背包还可以装5公斤)

4 最后考虑物品2,再放入5公斤的物品2,即完成工作.

5 最大利益为220元.

物品	價值	重量	单价
1	50	5	10
2	60	10	6
3	140	20	7

- 若此时商品改成不可分割,也就是说,对一个商品来说,要不就是全取,要不就是不取.
- 此时,贪婪算法不一定能求得最大利益.

因为：小偷的背包可以装下30公斤的物品

物品	價值	重量
1	50	5
2	60	10
3	140	20

贪婪算法:先取物品1,再取物品3;但物品2,不可再选取,否则背包会断裂.
故以贪婪算法所得到的最好的利益为190元.
但最佳的利益为物品2+物品3=200元.

如何求最小生成树

- ❖ Prim（普里姆）算法
- ❖ Kruskal（克鲁斯卡尔）算法

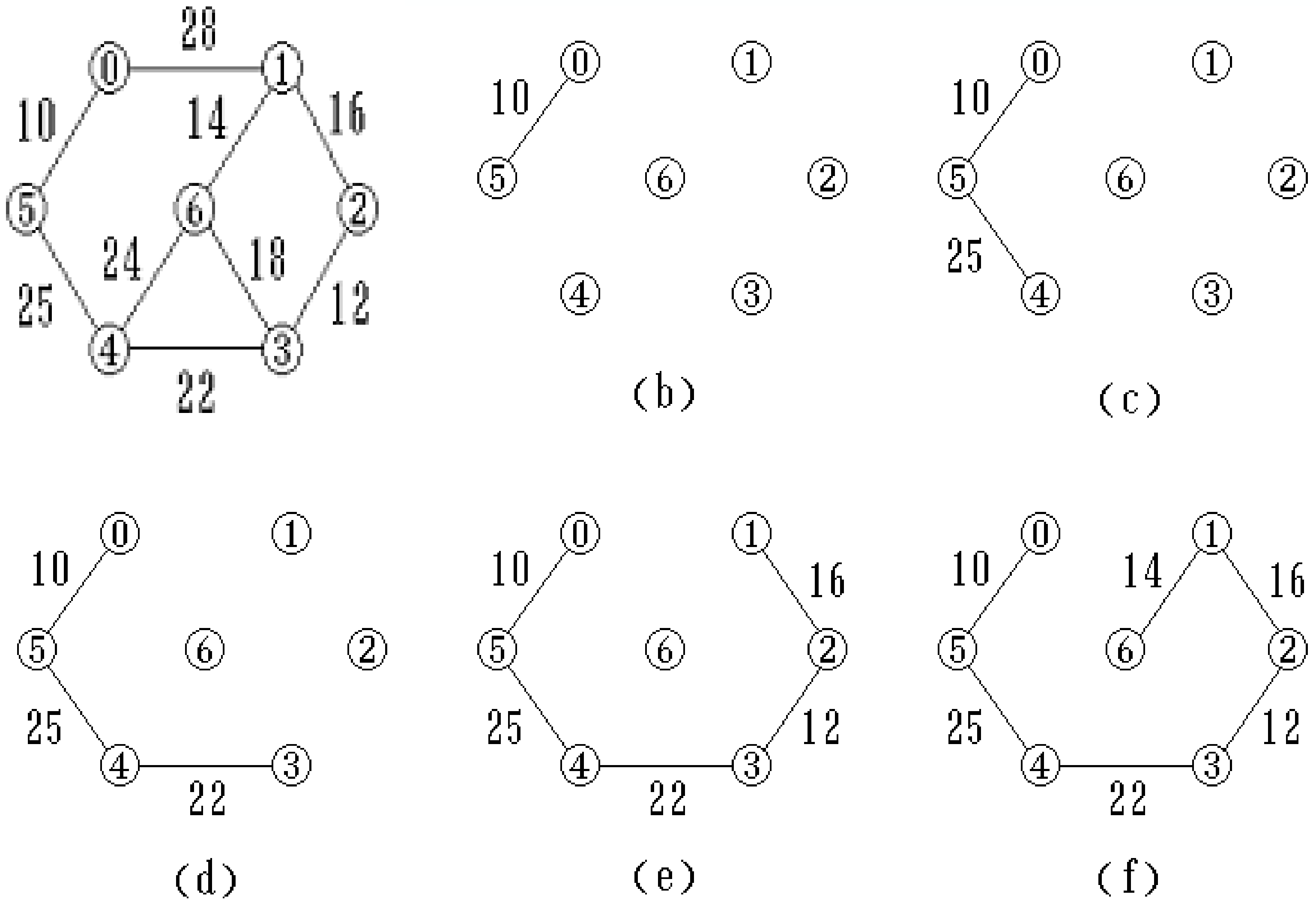
Prim算法: 归并顶点，与边数无关，适于稠密网

Kruskal算法：归并边，适于稀疏网

普里姆算法的基本思想 - - 归并顶点

- 设连通网络 $N = \{ V, E \}$
  - 1. 从某顶点 u_0 出发，选择与它关联的具有最小权值的边(u_0, v)，将其顶点加入到生成树的顶点集合U中
  - 2. 每一步从一个顶点在U中，而另一个顶点不在U中的各条边中选择权值最小的边(u, v),把它的顶点加入到U中
  - 3. 直到所有顶点都加入到生成树顶点集合U中为止

应用普里姆算法构造最小生成树的过程

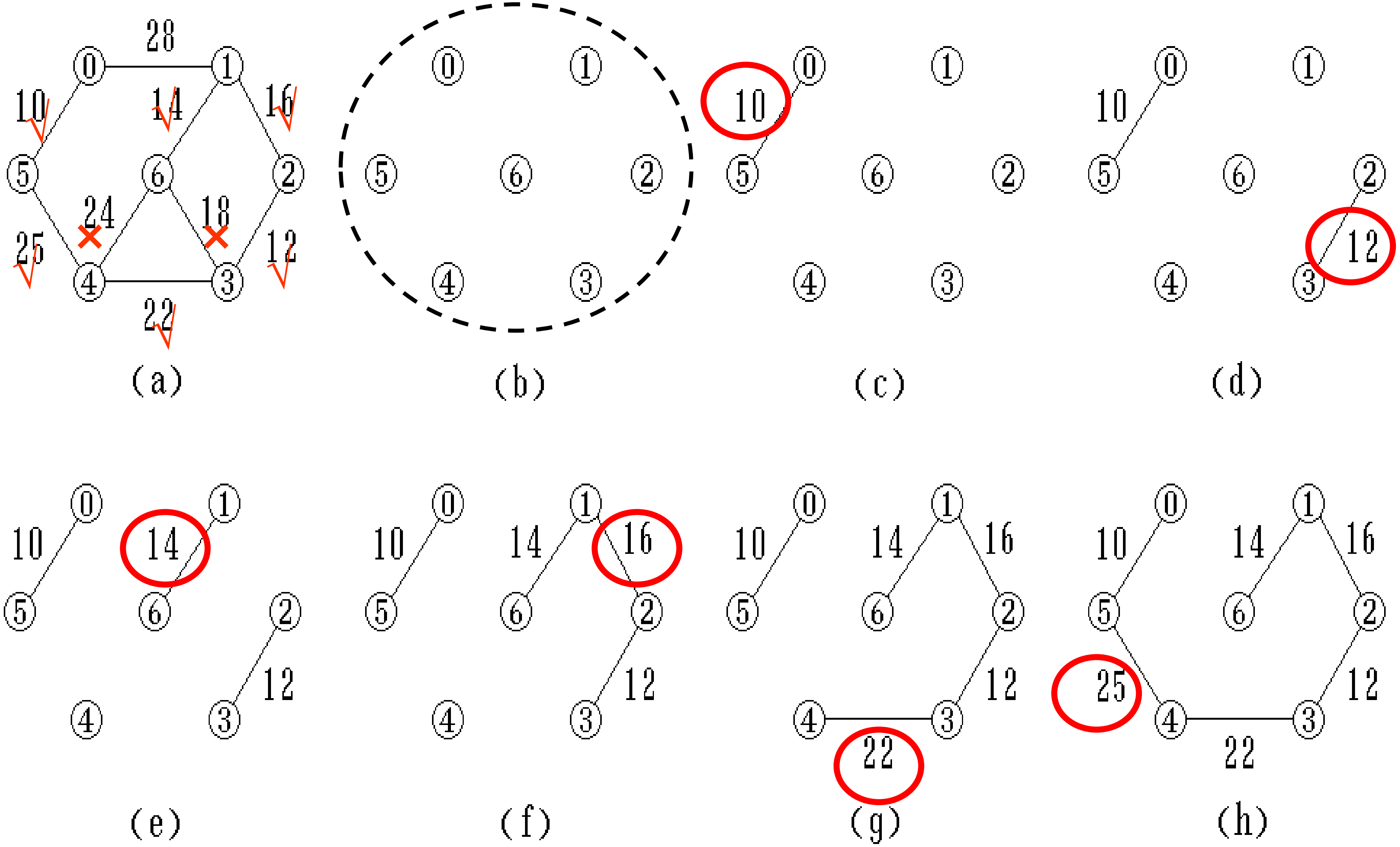


2020年6月5日

克鲁斯卡尔算法的基本思想 - 归并边

- 设连通网络 $N = \{ V, E \}$
  - 1. 构造一个只有 n 个顶点，没有边的非连通图 $T = \{ V, \varnothing \}$, 每个顶点自成一个连通分量
  - 2. 在 E 中选最小权值的边,若该边的两个顶点落在不同的连通分量上，则加入 T 中；否则舍去，重新选择
  - 3. 重复下去，直到所有顶点在同一连通分量上为止

应用克鲁斯卡尔算法构造最小生成树的过程



最短路径

典型用途：交通问题。如：城市A到城市B有多条线路，但每条线路的交通费（或所需时间）不同，那么，如何选择一条线路，使总费用（或总时间）最少?

问题抽象：在带权有向图中A点（源点）到达B点（终点）的多条路径中，寻找一条各边权值之和最小的路径，即最短路径。

（注：最短路径与最小生成树不同，路径上不一定包含n个顶点）

两种常见的最短路径问题：

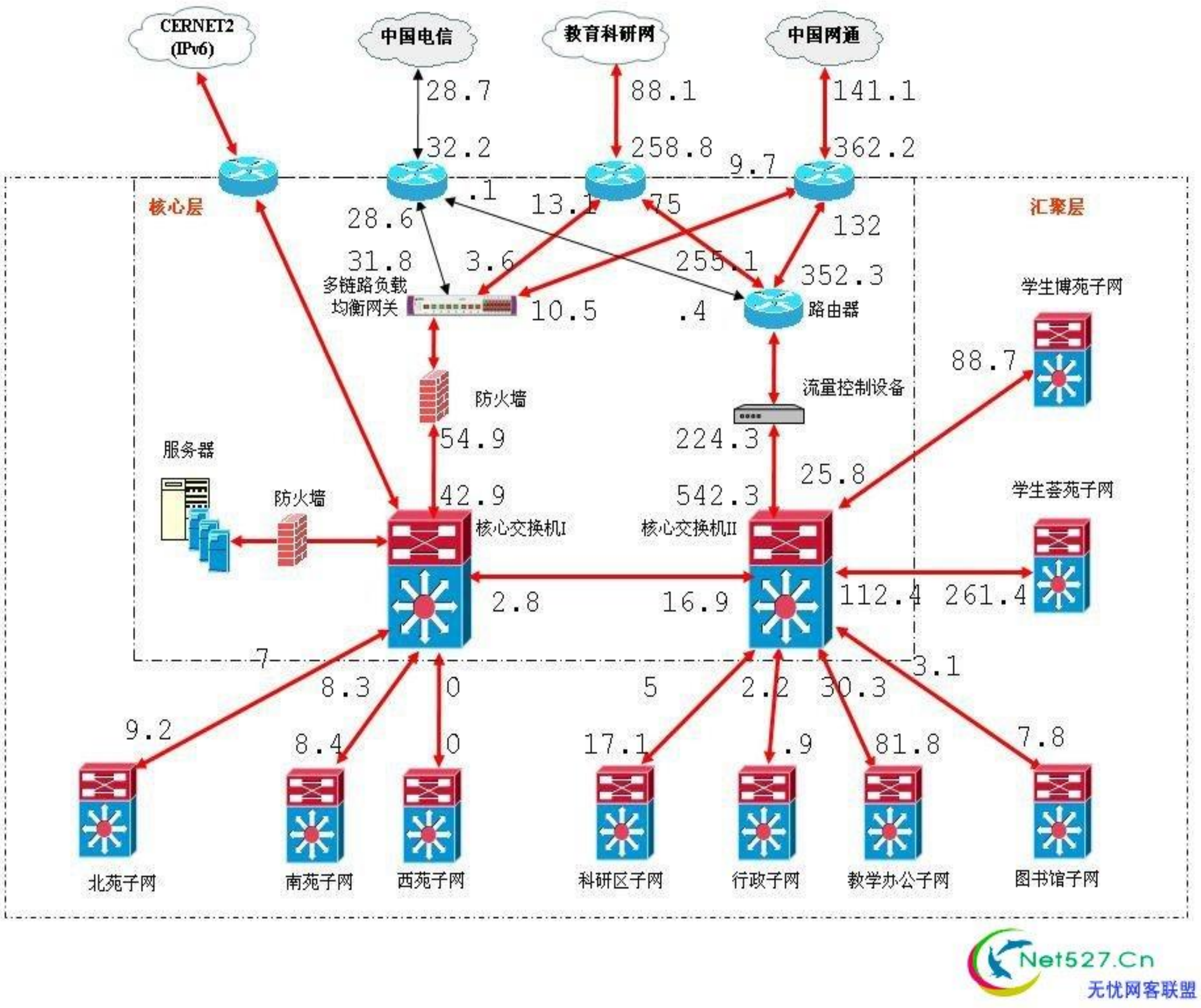
一、 单源最短路径—用Dijkstra（迪杰斯特拉）算法

二、所有顶点间的最短路径—用Floyd（弗洛伊德）算法

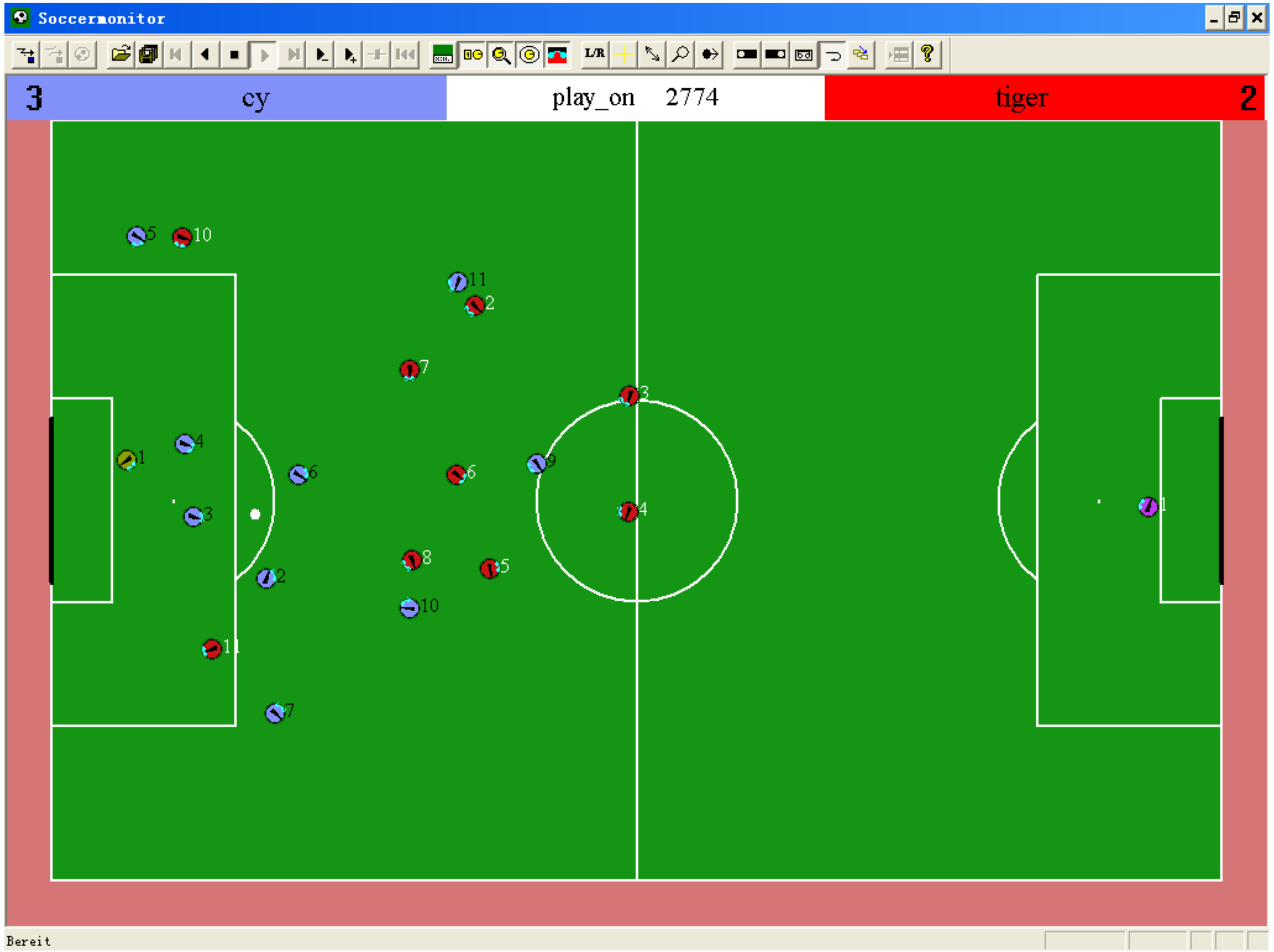
一顶点到其余各顶点

任意两顶点之间

最短路算法典型应用--计算机网络路由



最短路算法典型应用—机器人探路



2020年6月5日

最短路算法典型应用—游戏开发

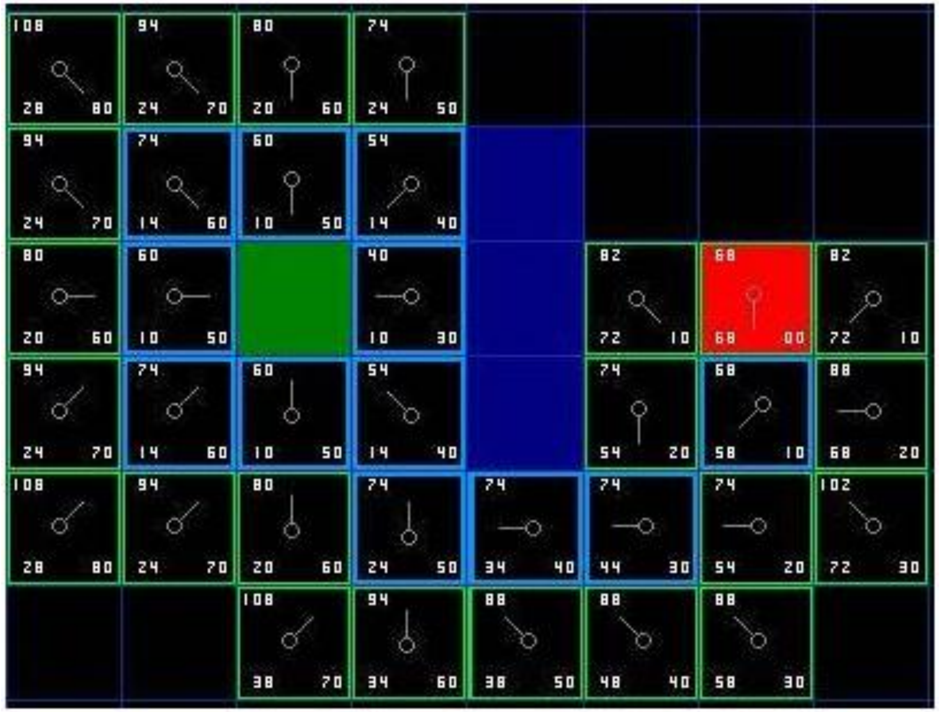
Dijistra算法的改进—A*算法（静态环境）

+估价值

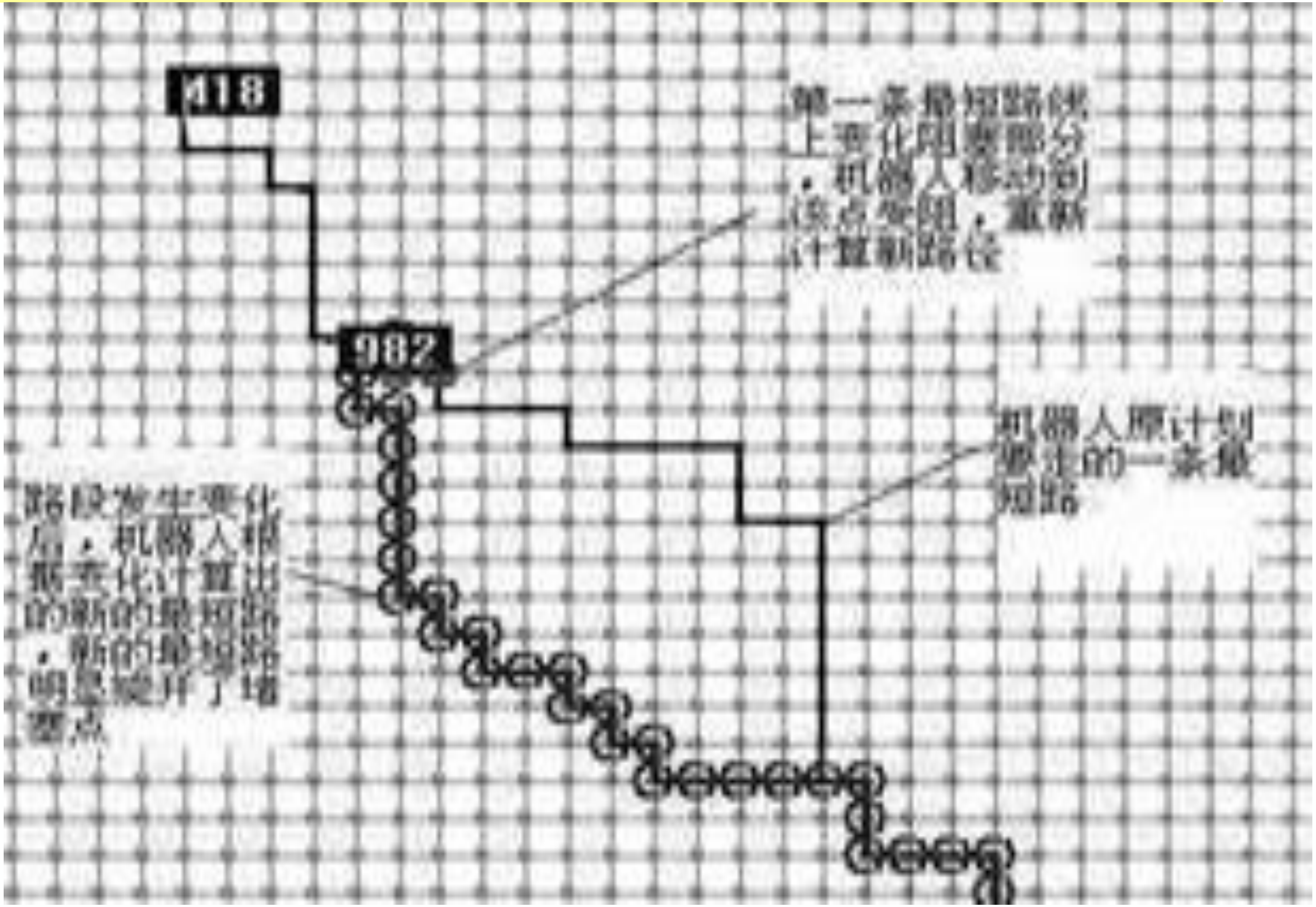
Dijistra算法 → A*算法

估价值=0

静态环境求解最短路最有效的方法



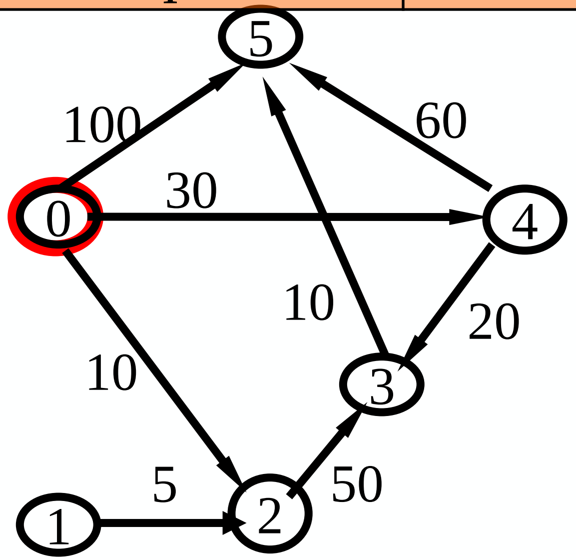
Dijistra算法的改进—D*算法（动态环境）



D*算法典型应用—火星探测器

从v_0到其余各点的最短路径--按路径长度递增次序求解

源　点	终　点	最短路径	路径长度
v_0	v_2	(v_0, v_2)	10
	v_4	(v_0, v_4)	30
	v_3	(v_0, v_4, v_3)	50
	v_5	(v_0, v_4, v_3, v_5)	60
	v_1	无	∞



$$
\begin{array}{c}
\mathbf{0}\\ \mathbf{1}\\ \mathbf{2}\\ \mathbf{3}\\ \mathbf{4}\\ \mathbf{5}
\end{array}
\begin{bmatrix}
0 & \infty & 10 & \infty & 30 & 100 \\
\infty & 0 & 5 & \infty & \infty & \infty \\
\infty & \infty & 0 & 50 & \infty & \infty \\
\infty & \infty & \infty & 0 & \infty & 10 \\
\infty & \infty & \infty & 20 & 0 & 60 \\
\infty & \infty & \infty & \infty & \infty & 0
\end{bmatrix}
$$

2020年6月5日

Dijkstra算法的思想

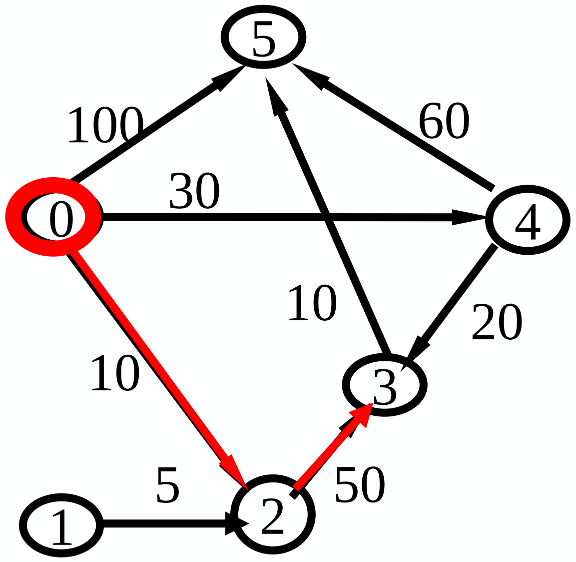
1.**初始化**：先找出从源点v_0到各终点v_k的直达路径（v_0,v_k），即通过一条弧到达的路径。

2.**选择**：从这些路径中找出一条长度最短的路径（v_0,u）。

3.**更新**：然后对其余各条路径进行适当调整：

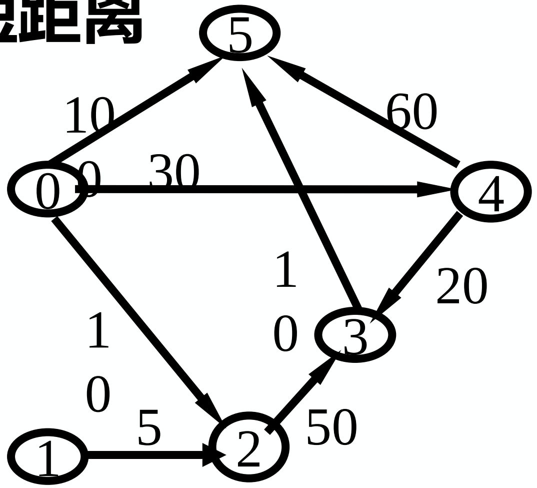
若在图中存在弧（u,v_k），且（v_0,u）+（u,v_k）<（v_0,v_k），则以路径（v_0,u,v_k）代替（v_0,v_k）。

在调整后的各条路径中，再找长度最短的路径，依此类推。



存储结构（顶点个数为n）

- 主：邻接矩阵G[n][n] (或者邻接表)
- 辅：
  - 数组S[n]：记录相应顶点是否已被确定最短距离
  - 数组D[n]：记录源点到相应顶点路径长度
  - 数组Path[n]：记录相应顶点的前驱顶点



算法初始化结果

	$v=0$	$v=1$	$v=2$	$v=3$	$v=4$	$v=5$
S	true	false	false	false	false	false
D	0	∞	10	∞	30	100
$Path$	-1	-1	0	-1	0	0

【算法思想】

① **初始化**：

- 将源点v0加到S中，即S[v0] = true；
- 将v0到各个终点的最短路径长度初始化为权值，

即D[i] = G.arcs[v0][vi]，(vi$\in V-S$)；

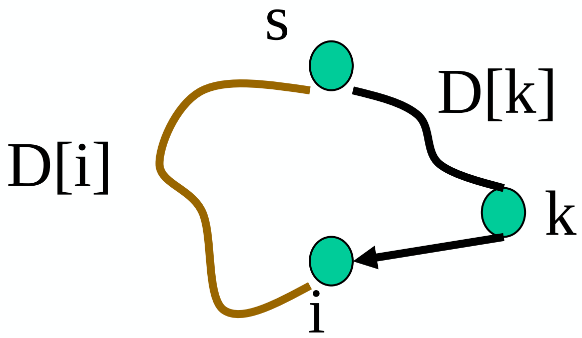
- 如果v0和顶点vi之间有弧，则将vi的前驱置为v0，

即Path[i] = v0，否则Path[i] = −1。

② **选择**下一条最短路径的终点vk，使得：

$$D[k] = \mathrm{Min}\{D[i] \mid v_i \in V - S\}$$

【算法思想】

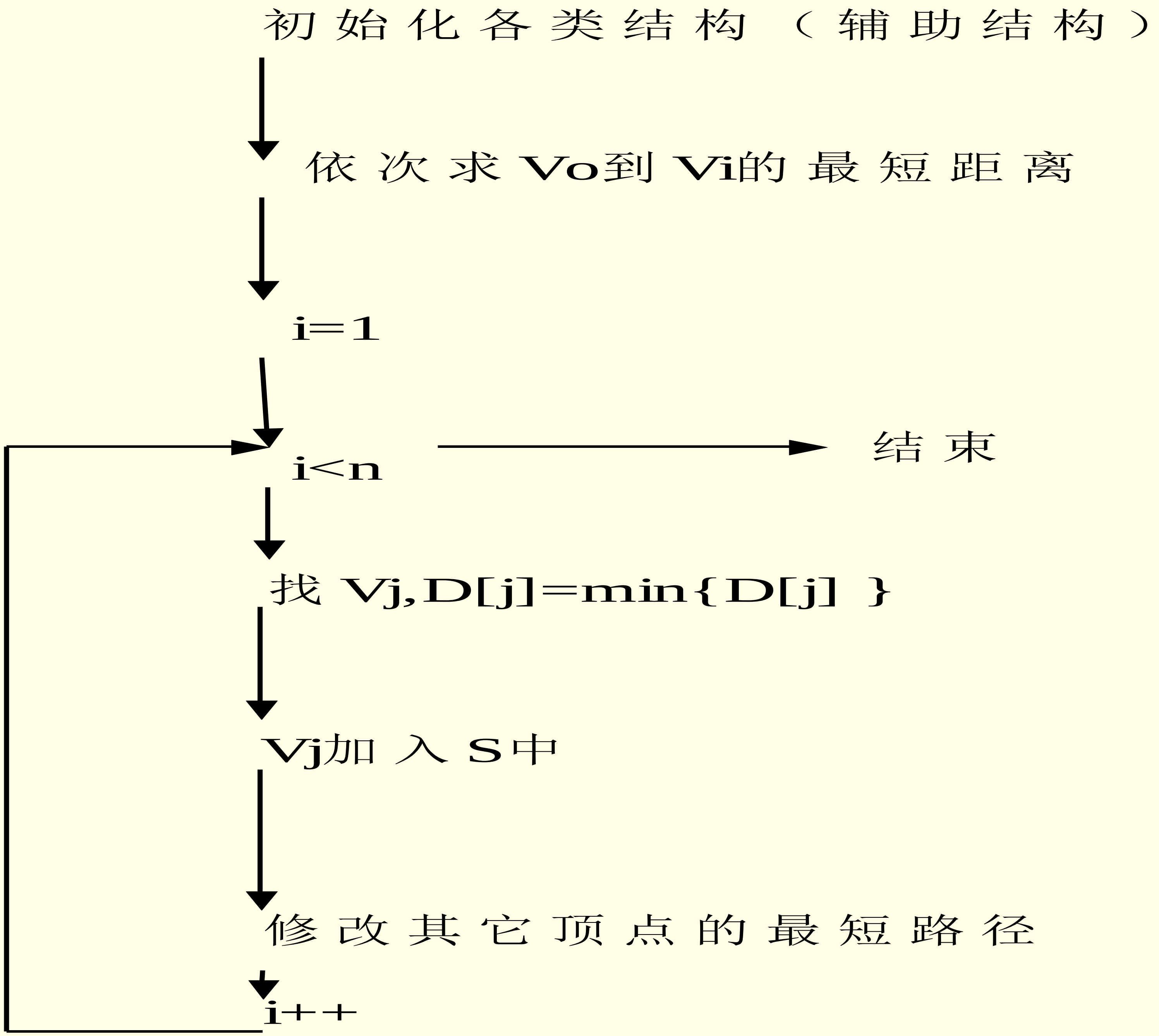


③ 将vk加到*S*中，即*S*[vk] = true。

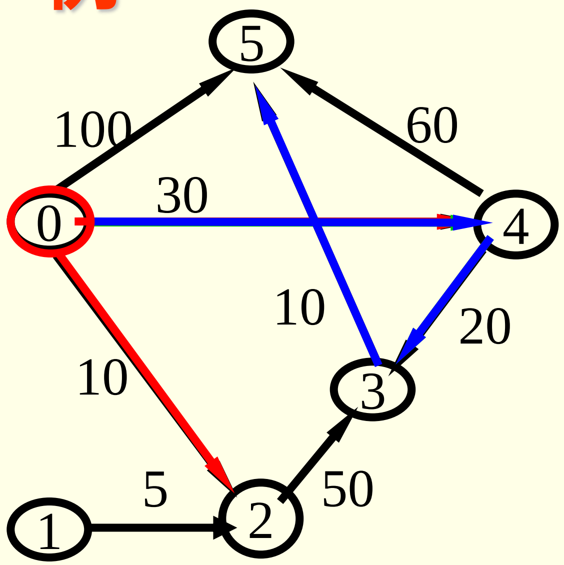
④ 更新从*v*0出发到集合*V* − *S*上任一顶点的最短路径的长度，同时更改vi的前驱为vk。

若S[i]=false 且 D[*k*]+G.arcs[*k*][*i*]<D[*i*]，则D[*i*]=D[*k*]+ G.arcs[*k*][*i*]; Path [*i*]=k;。

⑤ 重复②～④ *n* − 1次，即可按照路径长度的递增顺序，逐个求得从*v*0到图上其余各顶点的最短路径。


初 始 化 各 类 结 构 （ 辅 助 结 构 ）
依 次 求 Vo到 Vi的 最 短 距 离
i=1
i<n
结 束
找 Vj,D[j]=min{D[j] }
Vj加 入 S中
修 改 其 它 顶 点 的 最 短 路 径
i++

例



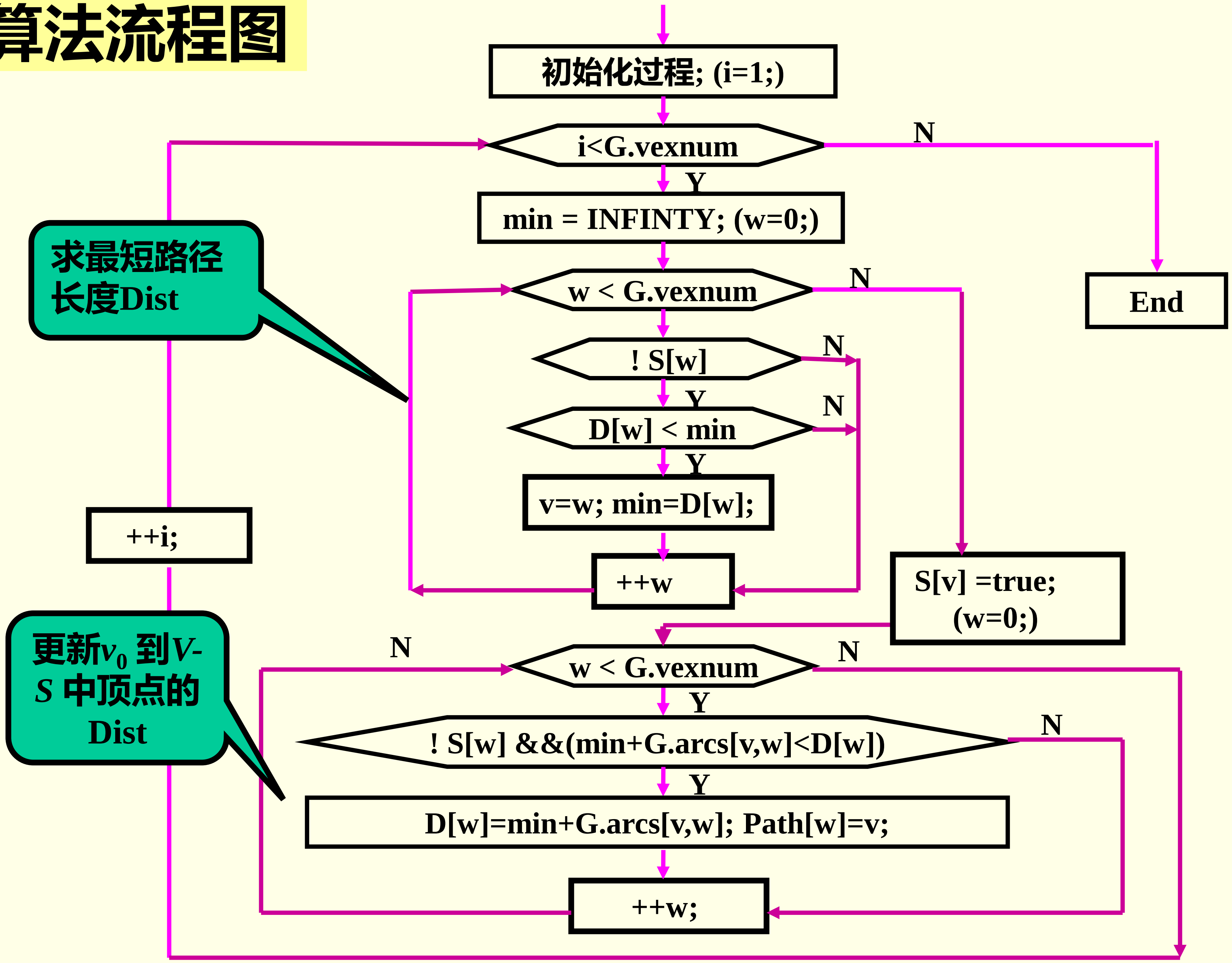
$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & \infty & 10 & \infty & 30 & 100 \\
1 & \infty & 0 & 5 & \infty & \infty & \infty \\
2 & \infty & \infty & 0 & 50 & \infty & \infty \\
3 & \infty & \infty & \infty & 0 & \infty & 10 \\
4 & \infty & \infty & \infty & 20 & 0 & 60 \\
5 & \infty & \infty & \infty & \infty & \infty & 0
\end{array}
$$

终点	D[w] 从v_0到各终点的长度和最短路径			
v_1	∞	∞	∞	∞
v_2	10 $\{v_0,v_2\}$			10 $\{v_0,v_2\}$
v_3	∞	60 $\{v_0,v_2,v_3\}$	50 $\{v_0,v_4,v_3\}$	50 $\{v_0,v_4,v_3\}$
v_4	30 $\{v_0,v_4\}$	30 $\{v_0,v_4\}$		30 $\{v_0,v_4\}$
v_5	100 $\{v_0, v_5\}$	100 $\{v_0, v_5\}$	90 $\{v_0,v_4, v_5\}$	60 $\{v_0,v_4,v_3,v_5\}$
v_j	v_2	v_4	v_3	v_5
s	$\{v_0,v_2\}$	$\{v_0,v_2,v_4\}$	$\{v_0,v_2,v_4,v_3\}$	$\{v_0,v_2,v_4,v_3,v_5\}$

S之外的当前最短路径之顶点

$(v_0,v_2)+(v_2,v_3)<(v_0,v_3)$

算法流程图



初始化过程; (i=1;)
i<G.vexnum
N
Y
min = INFINTY; (w=0;)
求最短路径长度Dist
w < G.vexnum
N
End
! S[w]
N
Y
D[w] < min
N
Y
v=w; min=D[w];
++i;
++w
S[v] =true;
(w=0;)
更新v_0 到*V*-*S* 中顶点的Dist
N
w < G.vexnum
N
Y
! S[w] &&(min+G.arcs[v,w]<D[w])
N
Y
D[w]=min+G.arcs[v,w]; Path[w]=v;
++w;

【算法描述】

```
void ShortestPath_DIJ(AMGraph G, int v0){
    //用Dijkstra算法求有向网G的v0顶点到其余顶点的最短路径
    n=G.vexnum;                                    //n为G中顶点的个数
    for(v = 0; v<n; ++v){                 //n个顶点依次初始化
       S[v] = false;                      //S初始为空集
       D[v] = G.arcs[v0][v];              //将v0到各个终点的最短路径长度初始化
       if(D[v]< MaxInt)  Path [v]=v0; //v0和v之间有弧，将v的前驱置为v0
       else Path [v]=-1;                  //如果v0和v之间无弧，则将v的前驱置为-1
     }//for
     S[v0]=true;                          //将v0加入S
     D[v0]=0;                             //源点到源点的距离为0
```

【算法描述】

时间复杂度：O(n^2)

```
/*—开始主循环，每次求得v0到某个顶点v的最短路径，将v加到S集—*/
   for(i=1;i<n; ++i){                       //对其余n−1个顶点，依次进行计算
    min= MaxInt;
    for(w=0;w<n; ++w)
     if(!S[w]&&D[w]<min)
       {v=w; min=D[w];}                     //选择一条当前的最短路径，终点为v
    S[v]=true;                              //将v加入S
    for(w=0;w<n; ++w)  //更新从v0出发到集合V−S上所有顶点的最短路径长度
    if(!S[w]&&(D[v]+G.arcs[v][w]<D[w])){
       D[w]=D[v]+G.arcs[v][w];              //更新D[w]
       Path [w]=v;                          //更改w的前驱为v
    }//if
  }//for
}//ShortestPath_DIJ
```