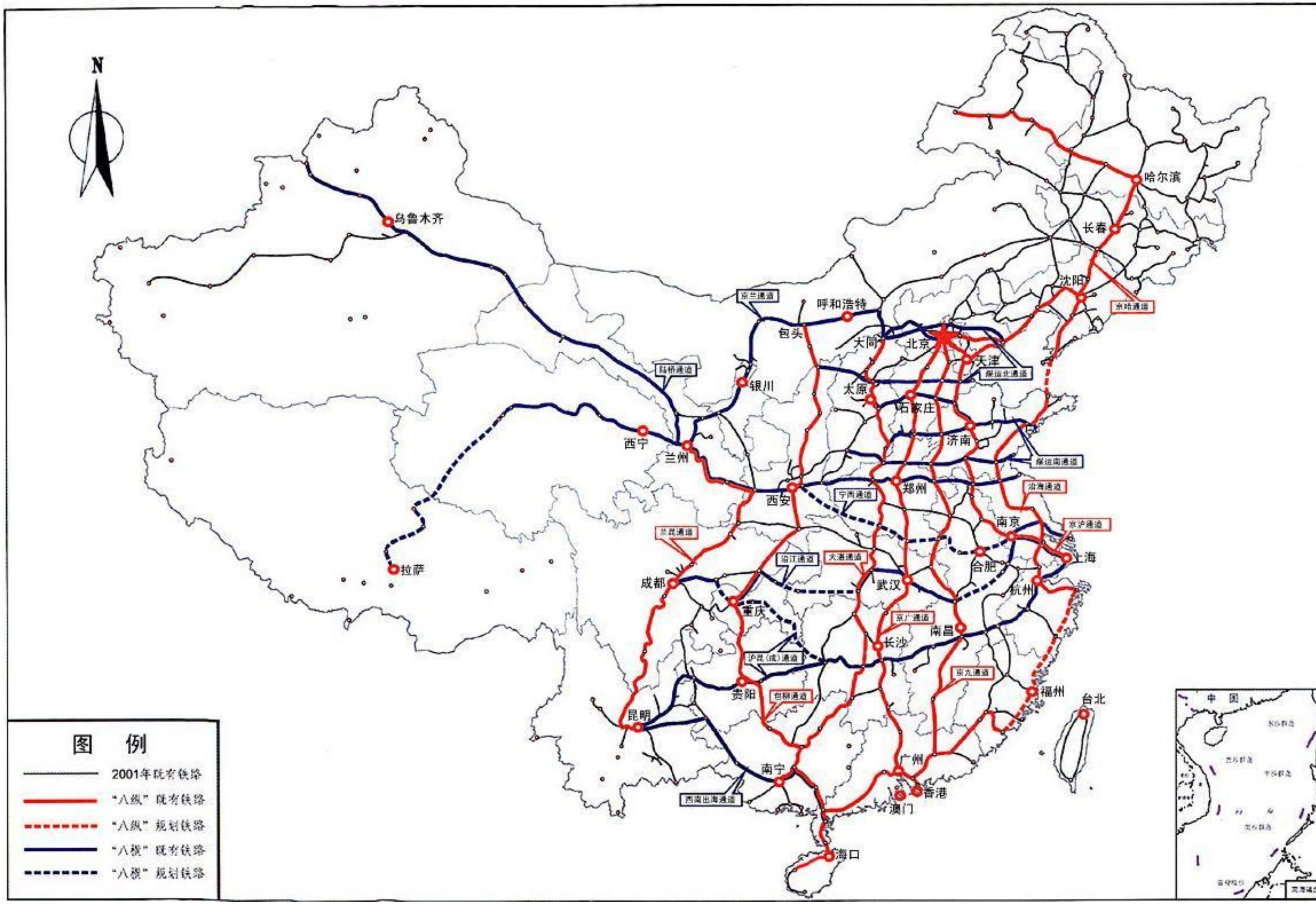
第7章 图


N
乌鲁木齐
拉萨
西宁
兰州
银川
呼和浩特
包头
大同
北京
天津
太原
石家庄
济南
郑州
西安
南京
上海
合肥
武汉
杭州
成都
重庆
长沙
南昌
贵阳
昆明
福州
台北
南宁
广州
香港
澳门
海口
哈尔滨
长春
沈阳
京兰通道
陆桥通道
京哈通道
煤运北通道
煤运南通道
沿海通道
京沪通道
宁西通道
兰昆通道
沿江通道
大湛通道
京广通道
沪昆(成)通道
京九通道
包柳通道
西南出海通道
图 例
2001年既有铁路
"八纵"既有铁路
"八纵"规划铁路
"八横"既有铁路
"八横"规划铁路
中 国

回顾

两种常用的活动网络(Activity Network)

① **AOV网(Activity On Vertices)**—用顶点表示活动的网络

AOV网定义：若用有向图表示一个工程，在图中用顶点表示活动，用弧表示活动间的优先关系。v_i 必须先于活动 v_j 进行。则这样的有向图叫做用顶点表示活动的网络。

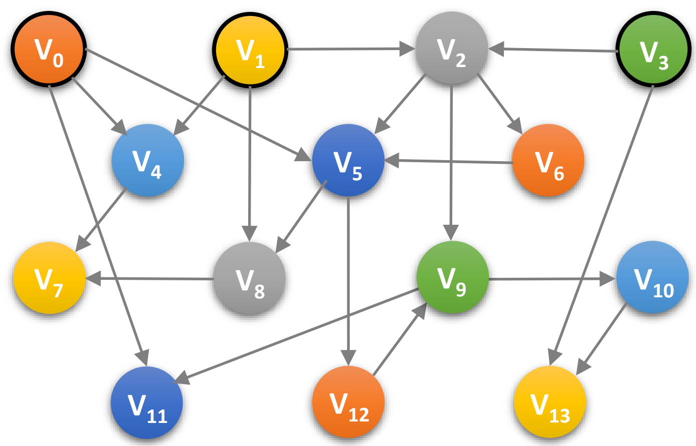
② **AOE网(Activity On Edges)**—用边表示活动的网络

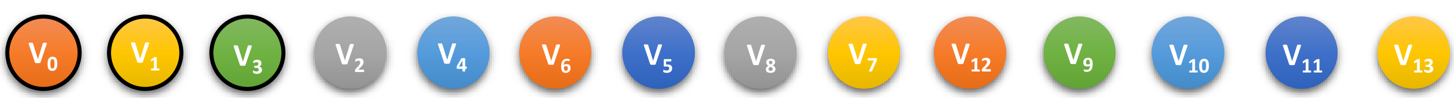
AOE网定义：如果在无环的带权有向图中， 用有向边表示一个工程中的活动，用边上权值表示活动持续时间，用顶点表示事件，则这样的有向图叫做用边表示活动的网络。

回顾

拓扑排序

1. 在有向图中选一个<u>没有前驱</u><u>的顶点并</u>输出
2. 从图中<u>删除该顶点</u>和所有<u>以它为尾的弧</u>
3. 重复上述两步，直至全部顶点均已输出；或者当图中不存在无前驱的顶点为止





第7章 图

7.5.2 关键路径

AOE网--用边表示活动的网(Activity On Edges)

- **AOE网**：如果在带权的有向无环图中
  - ✓ 用有向边表示一个工程中的活动 **(Activity)**
  - ✓ 用边上权值表示活动持续时间 **(Duration)**
  - ✓ 用顶点表示事件 **(Event)**

■ **AOE网在某些工程进度估算方面非常有用。利用它可以解决以下两个问题：**

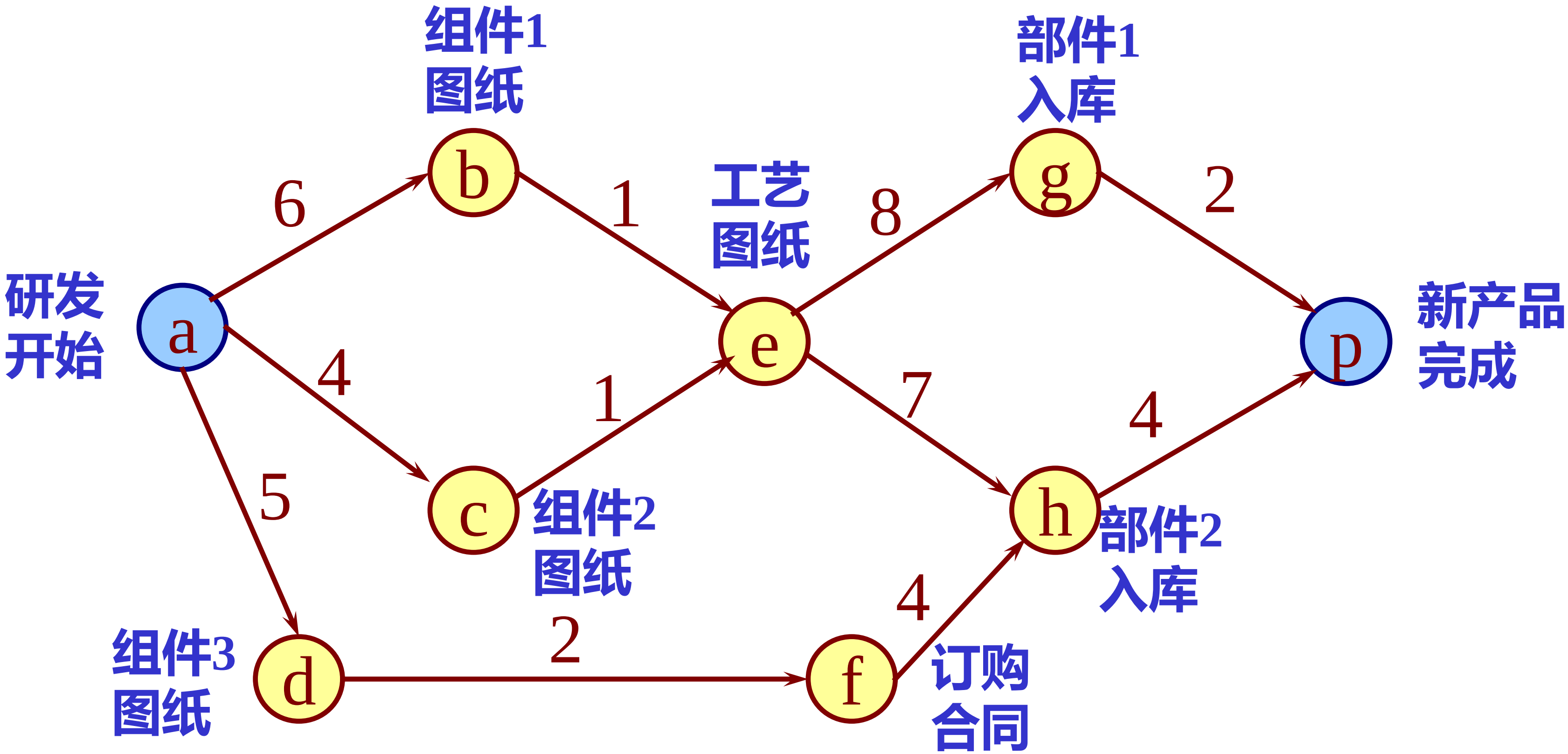
(1) 完成整个工程至少需要多少时间(假设网络中没有环)?

(2)为缩短完成工程所需的时间, 应当加快哪些活动?

或者说，哪些活动是影响工程进度的关键?

7.5.2 关键路径

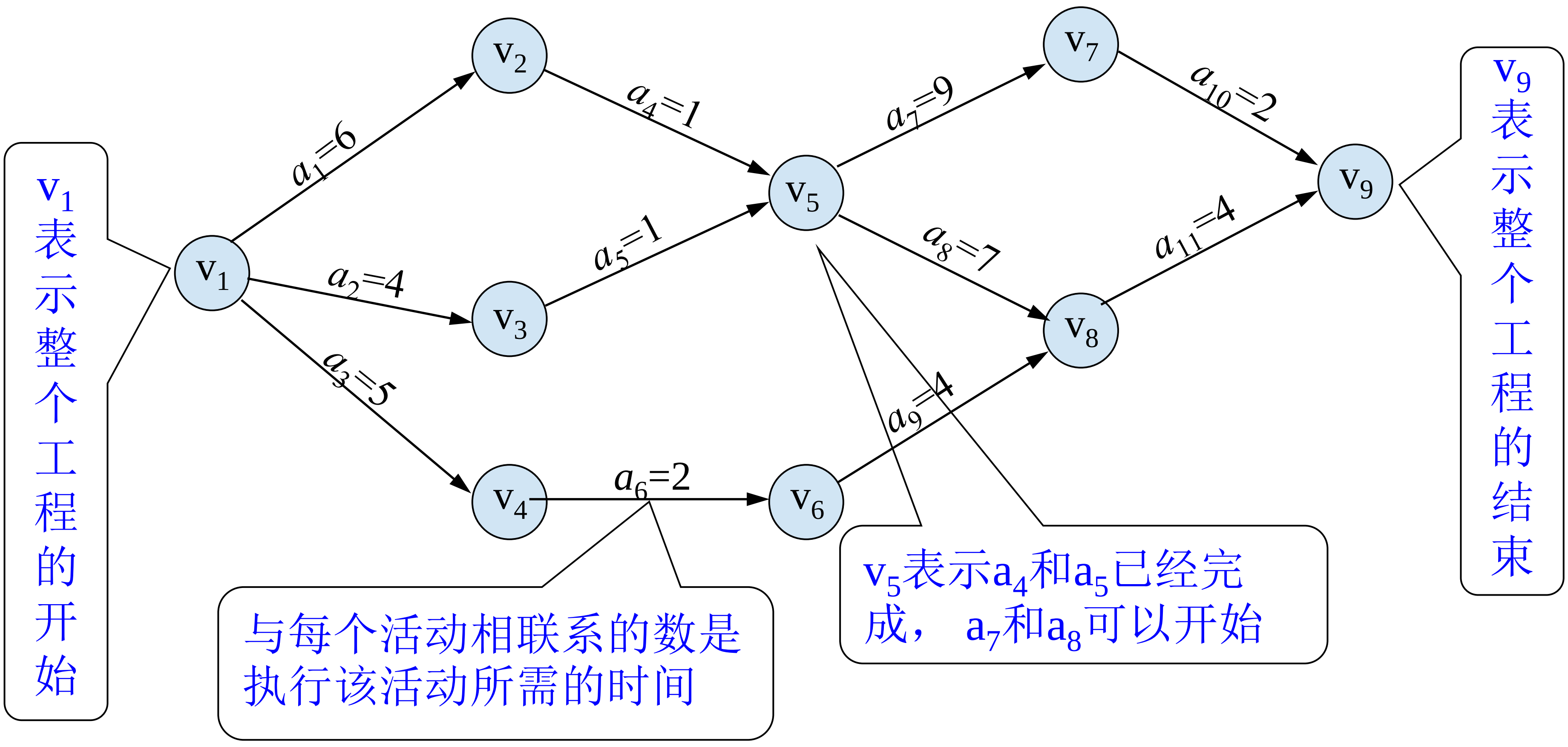
假设以有向网表示一个施工图，弧上的权值表示完成该项子工程所需时间。



- 整个工程的完成时间？
- 哪些子工程将影响整个工程的完成时间？

7.5.2 关键路径

□ AOE网

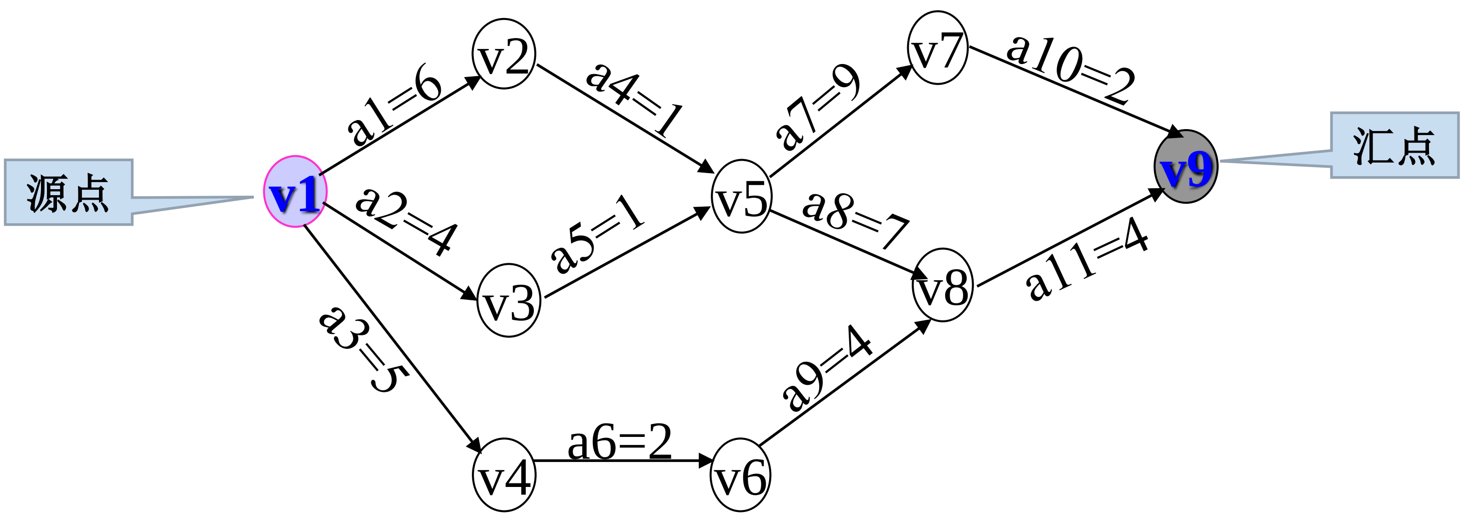


7.5.2 关键路径

实例： 设一个工程有11项活动，9个事件

事件 V1（源点）——表示整个工程开始
事件V9（汇点）——表示整个工程结束
有向边（弧）---表示活动
弧的权值------表示该活动持续时间

每个事件表示在它之前的活动已经结束，在它之后的活动可以开始。



7.5.2 关键路径

□ 关键路径

对整个工程和系统，通常关心三个方面的问题

(1) 工程能否顺利进行

对AOV网进行拓扑排序

(2)完成整项工程至少需要多少时间?

对AOE网求关键路径

(3)哪些活动是影响工程进度的关键?

对AOE网求关键路径

7.5.2 关键路径

AOE网

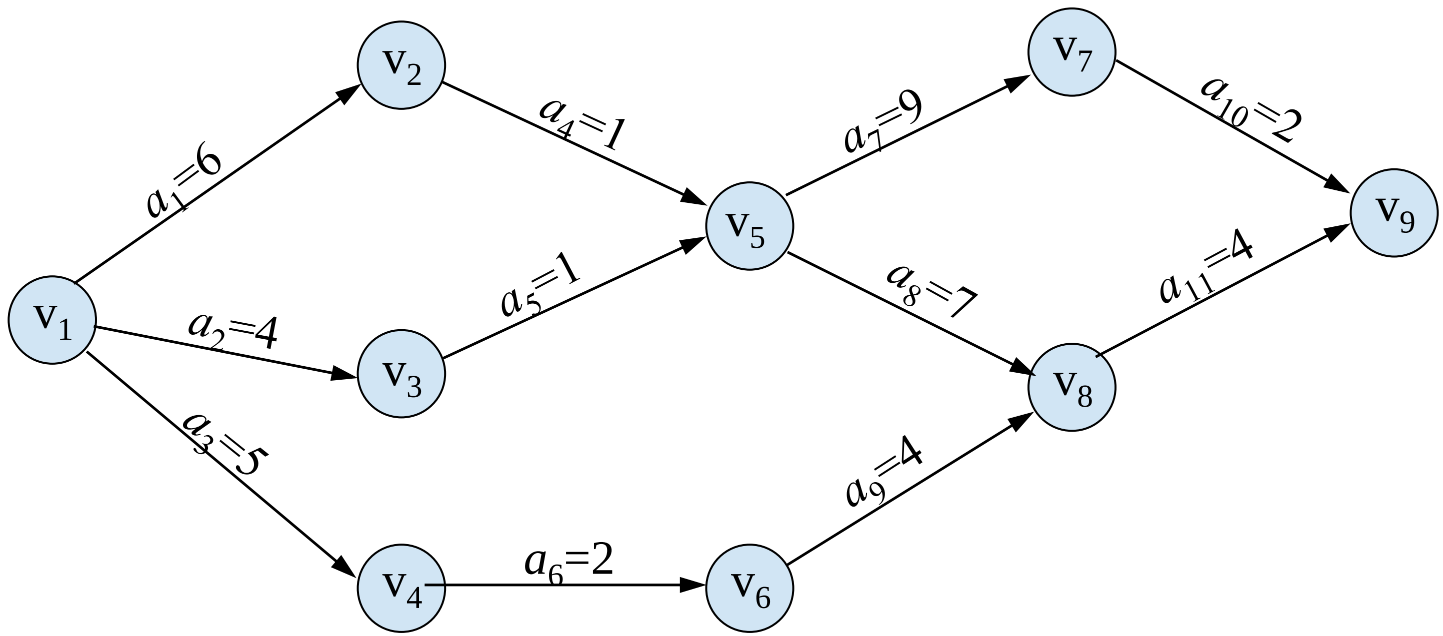
依据AOE-网可以研究什么问题?

(1) 完成整项工程至少需要多少时间?

(2) 哪些活动是影响工程进度的关键?

7.5.2 关键路径

◻ 关键路径



完成工程的最短时间是从开始点到完成点的最长路径的长度。
路径长度最长的路径叫做关键路径。
从v_1到v_9的最长路径是(v_1,v_2,v_5,v_8,v_9)，路径长度是18。

7.5.2 关键路径

- 事件V_i 的最早可能开始时间*Ve[i]*：是从源点V_0 到顶点V_i 的最长路径长度。
- 事件V_i 的最迟允许开始时间*Vl[i]*：是在保证汇点V_{n-1} 在*Ve[n-1]* 时刻完成的前提下，事件V_i 的允许的最迟开始时间。
- 活动a_k 的最早可能开始时间 *e[k]*：设活动a_k 在边<V_i, V_j>上, 则*e[k]*是从源点V_0到 顶点V_i 的最长路径长度。因此, *e[k] = Ve[i]*。
- 活动a_k 的最迟允许开始时间 *l[k]*：*l[k]*是在不会引起时间延误的前提下, 该活动允许的最迟开始时间。*l[k] = Vl[j]-dur(<i, j>)*。

  其中, *dur(<i, j>)*是完成 a_k 所需的时间。
- 时间余量 *l[k]-e[k]*：表示活动a_k的最早可能开始时间和最迟允许开始时间的时间余量。*l[k] == e[k]*表示活动 a_k 是没有时间余量的关键活动。

为找出关键活动, 需要求各个活动的 *e[k]* 与 *l[k]*，以判别是否 *l[k] == e[k]*。

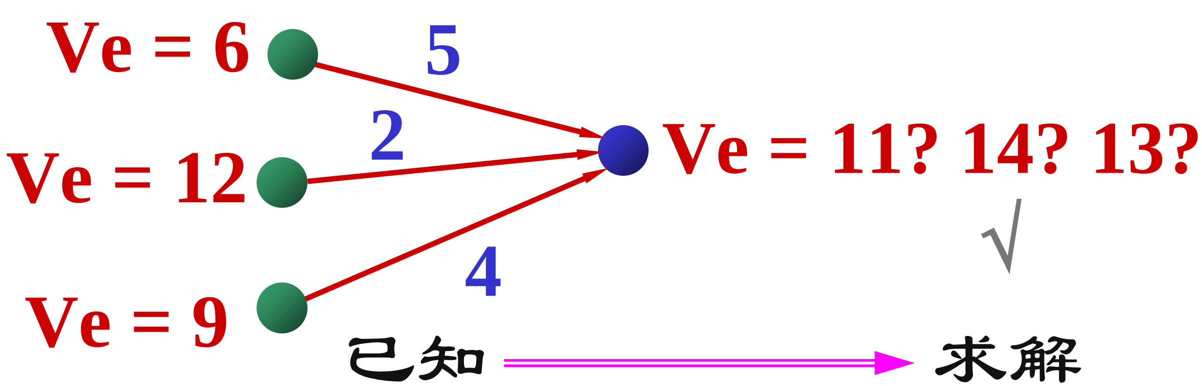
7.5.2 关键路径

- 为求得$e[k]$与$l[k]$，需要先求得从源点V_0到各个顶点V_i的$Ve[i]$和$Vl[i]$。
- 求$Ve[i]$的递推公式

从 $Ve[0] = 0$ 开始，向前递推:

$$Ve[j] = \max_i \{ Ve[i] + dur(<V_i, V_j>) \},$$

其中，$< V_i, V_j > \in S2$，$j = 1, 2, \ldots\ldots, n\text{-}1$。$S2$ 是所有指向V_j的有向边$< V_i, V_j >$的集合。



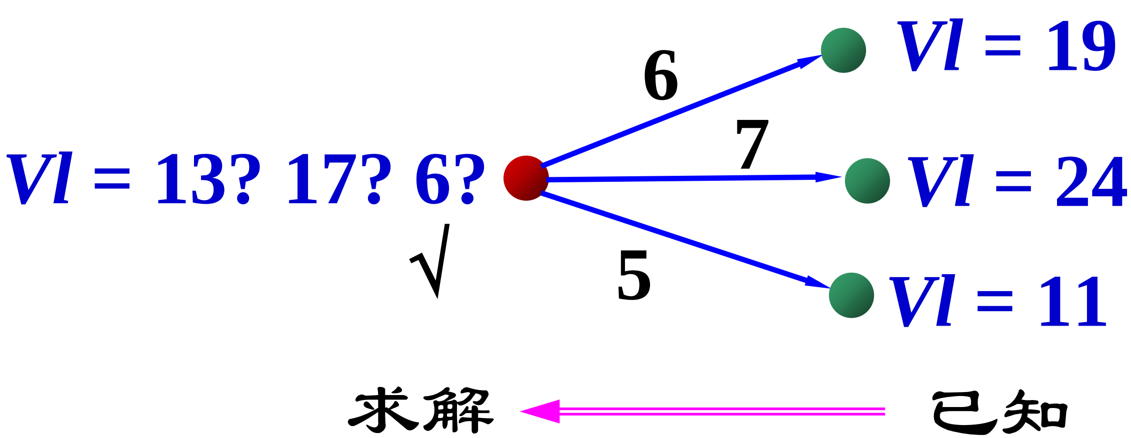
7.5.2 关键路径

- 求*Vl[i]*的递推公式：

  从*Vl[n-1] = Ve[n-1]* 开始，反向递推：

$$Vl[j] = \min_{k}\{Vl[k] - dur(<V_j, V_k>)\},$$

其中，$<V_j, V_k> \in S1, j = n\text{-}2, n\text{-}3, \ldots\ldots, 0$。*S1*是所有源自 V_j 的有向边$<V_j, V_k>$的集合。



这两个递推公式的计算必须分别在拓扑有序及逆拓扑有序的前提下进行。

7.5.2 关键路径

- 设活动a_k (k=1, 2, …, e)在带权有向边< V_i, V_j >上, 其持续时间用dur (< V_i, V_j >)表示, 则有

  e[k] = Ve[i]；

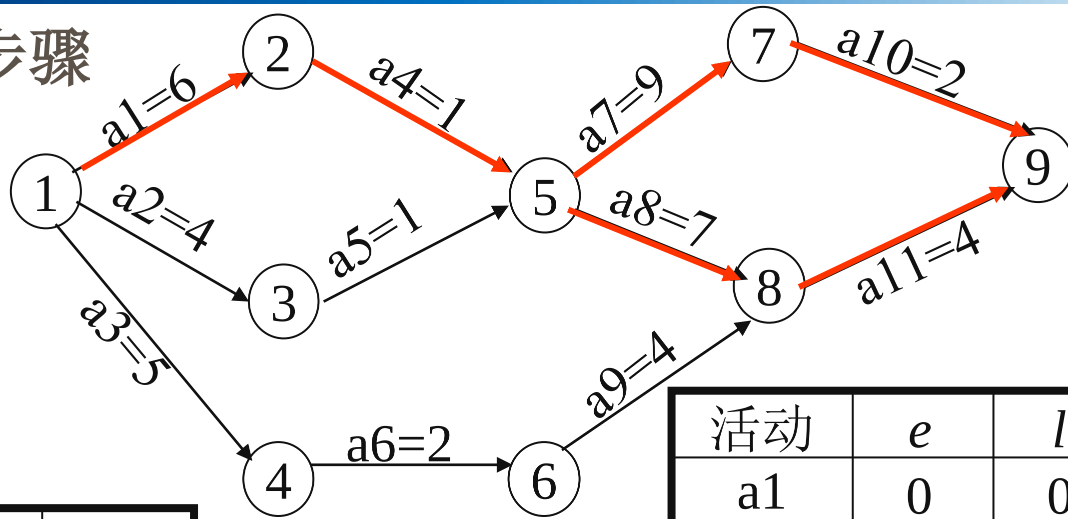
  l[k] = Vl[j]-dur(< V_i, V_j >*)*； *k = 1, 2, …, e*。

  这样就得到计算关键路径的算法。

- 为了简化算法， 假定在求关键路径之前已经对各顶点实现了拓扑排序， 并按拓扑有序的顺序对各顶点重新进行了编号。

7.5.2 关键路径

- 求关键路径步骤
  - 求*ve(i)*
  - 求*vl(j)*
  - 求 *e(i)*
  - 求 *l(i)*
  - 计算*l(i)-e(i)*

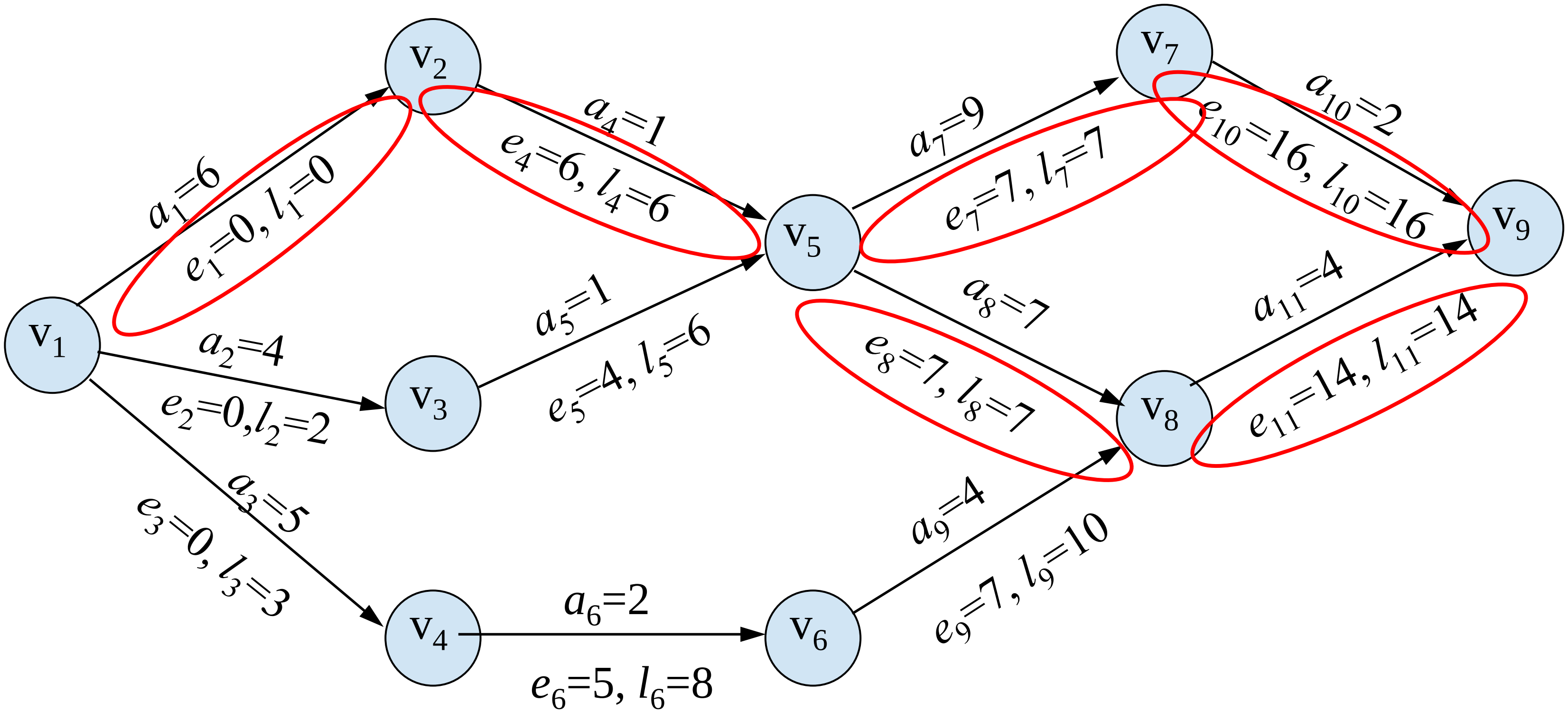


顶点	*ve*	*vl*
V1	0	0
V2	6	6
V3	4	6
V4	5	8
V5	7	7
V6	7	10
V7	16	16
V8	14	14
V9	18	18

活动	*e*	*l*	*l-e*
a1	0	0	0 ✓
a2	0	2	2
a3	0	3	3
a4	6	6	0 ✓
a5	4	6	2
a6	5	8	3
a7	7	7	0 ✓
a8	7	7	0 ✓
a9	7	10	3
a10	16	16	0 ✓
a11	14	14	0 ✓

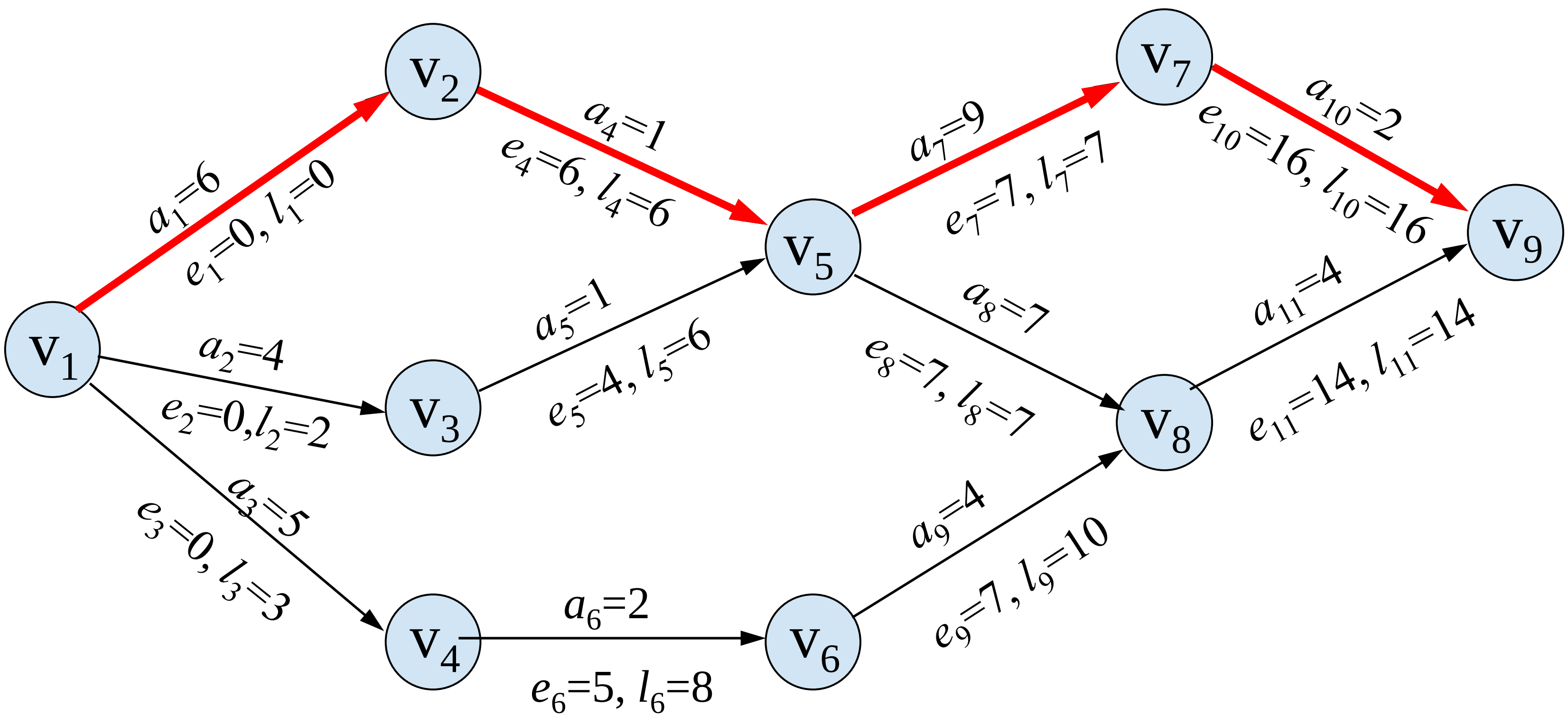
7.5.2 关键路径

关键路径



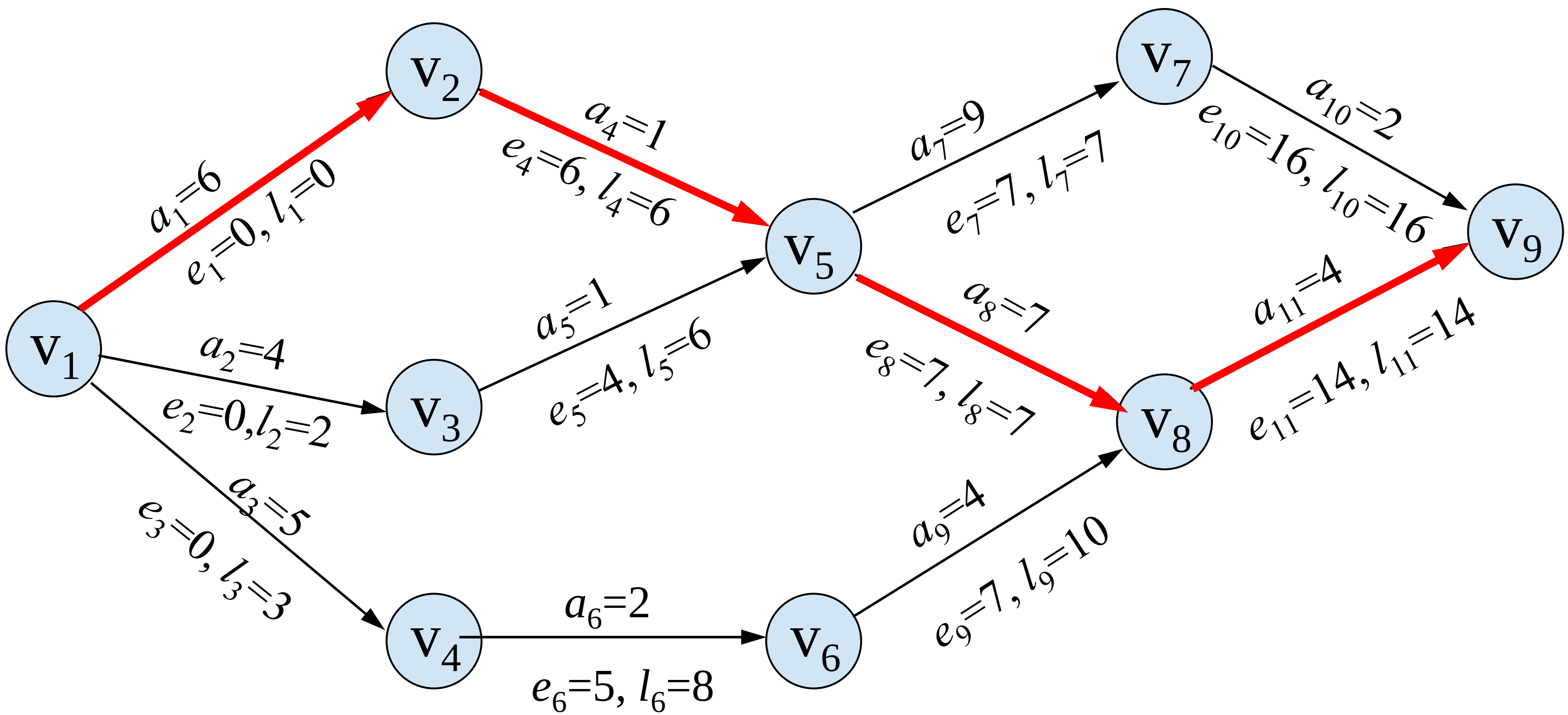
7.5.2 关键路径

- 关键路径



7.5.2 关键路径

□ 关键路径



7.5.2 关键路径

关键路径算法思想

1）输入e条弧（i，j），建立AOE网的存储结构。

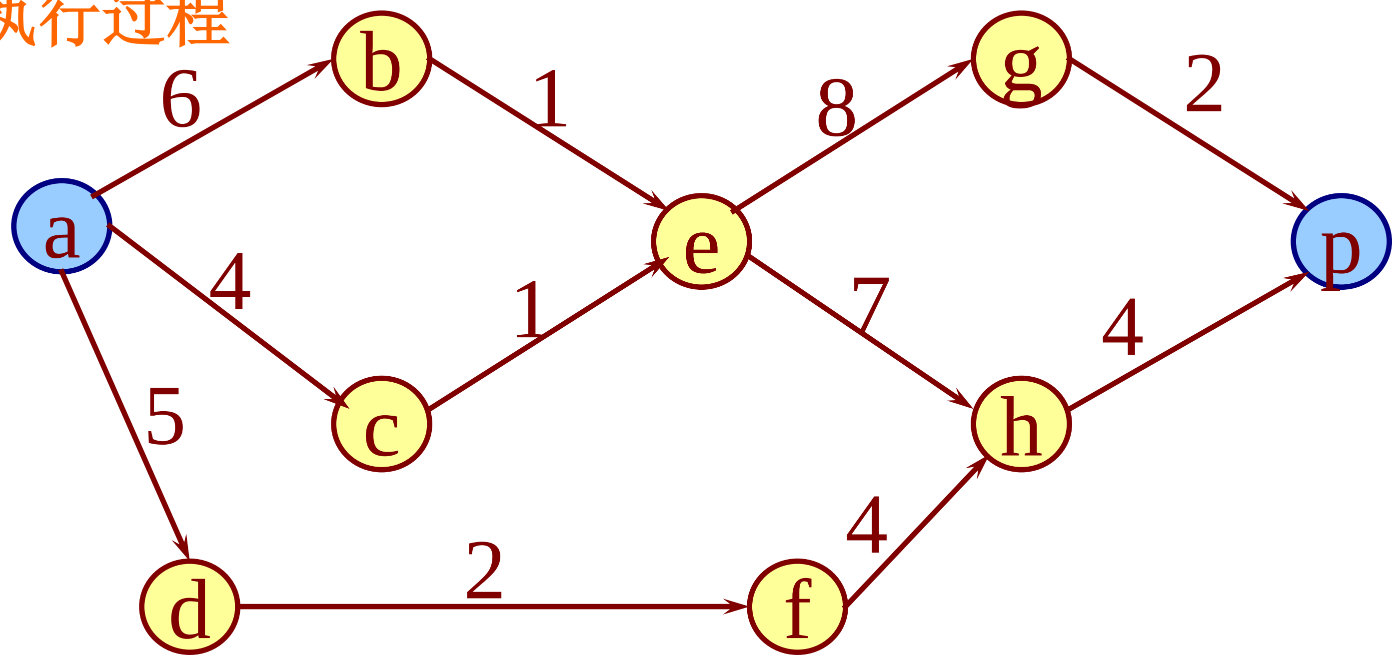
2）从源点v_0出发，令ve[0]＝0按拓扑有序求其余各顶点的最早发生时ve[i]（1≤i≤ n-1）。如果得到的拓扑有序序列中顶点个数小于网中顶点数n，则说明网中存在环，不能求关键路径，算法终止；否则执行步骤（3）。

3）从汇点v_n出发，令vl[n-1]= ve[n-1]，按逆拓扑有序求其余各顶点的最迟发生时间vl[i] (n-2 ≥i≥ 2);

4）根据各顶点的ve和vl值，求每条弧s的最早开始时间e(s)和最迟开始时间l(s)。若某条弧满足条件e(s)=l(s)，则为关键活动。

7.5.2 关键路径

算法的执行过程



	a	b	c	d	e	f	g	h	p
Ve	0	6	4	5	7	7	15	14	18
Vl	0	6	6	8	7	10	16	14	18

拓扑有序序列： **a - d - f - c - b - e - h - g - p**

7.5.2 关键路径

	a	b	c	d	e	f	g	h	p
Ve	0	6	4	5	7	7	15	14	18
Vl	0	6	6	8	7	10	16	14	18

	ab	ac	ad	be	ce	df	eg	eh	fh	gp	hp
权	6	4	5	1	1	2	8	7	4	2	4
e	0	0	0	6	4	5	7	7	7	15	14
l	0	2	3	6	6	8	8	7	10	16	14
	✓			✓				✓			✓

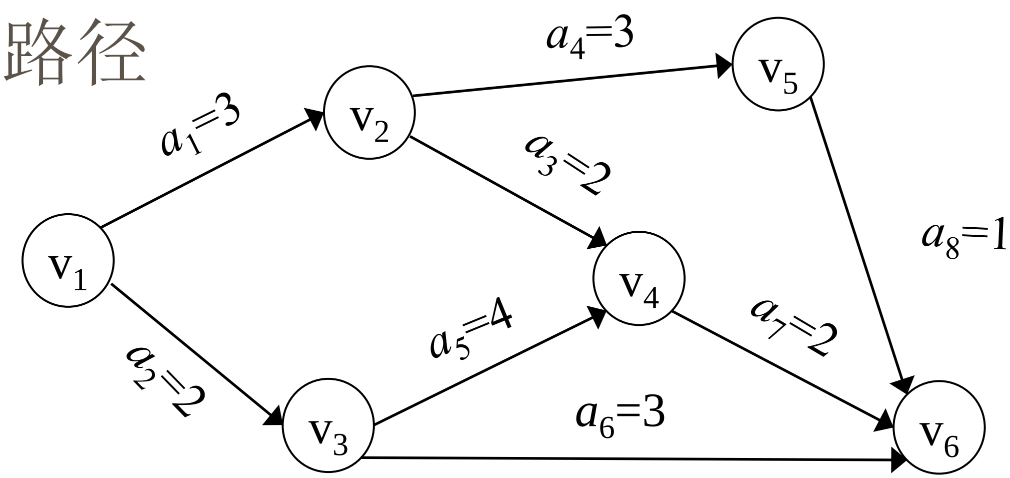
7.5.2 关键路径

```
Status TopologicalSort（ALGraph G, Stack &T)）{
  FindInDegree(G, indegree);  //计算各顶点的入度    T为拓扑排序顶点栈
  InitStack(S); count=0; ve[0 .. G.vexnum-1] = 0,
  for ( i=0; i<G.vexnum; ++i)
    if (!indegree[i])  Push(S, i);      //入度为零的顶点入栈S
  while (!EmptyStack(S)) {              //S为入度为零顶点栈
    Pop(S, v); Push(T, j);  ++count;   printf(v);
    for (w=FirstAdj(v); w;  w=NextAdj(G,v,w)){
      --indegree(w);                     // 弧头顶点的入度减1
      if (!indegree[w])  Push(S, w);  // 入度为零的顶点入栈S
      if (ve[j] + *(p->info)  >  ve[k]) ve[k] = ve[j] + * (p->info);
    } //for
  } //while
  if (count<G.vexnum) printf("图中有回路" );
}
```

7.5.2 关键路径

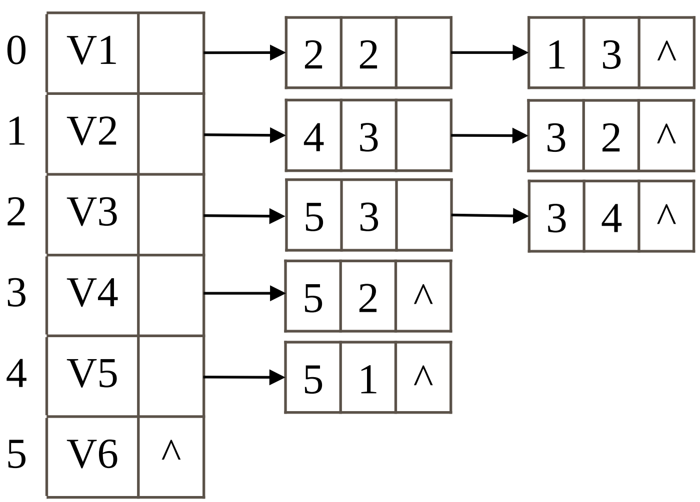
关键路径

例



T

V6
V5
V4
V3
V2
V1



拓扑排序：**V1 V2 V5 V3 V4 V6**

$\boldsymbol{ve}$**(源点) = 0**

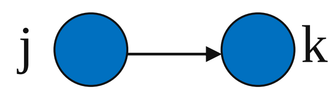
$\boldsymbol{ve(j) = \mathrm{Max}\{ ve(i) + \mathrm{dut}(<i, j>)\}}$

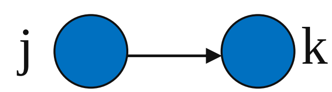
if (ve[j]+*(p->info) > ve[k]) ve[k] = ve[j]+* (p->info);

	V1	V2	V3	V4	V5	V6
Ve	0	3	2	6	6	8

7.5.2 关键路径

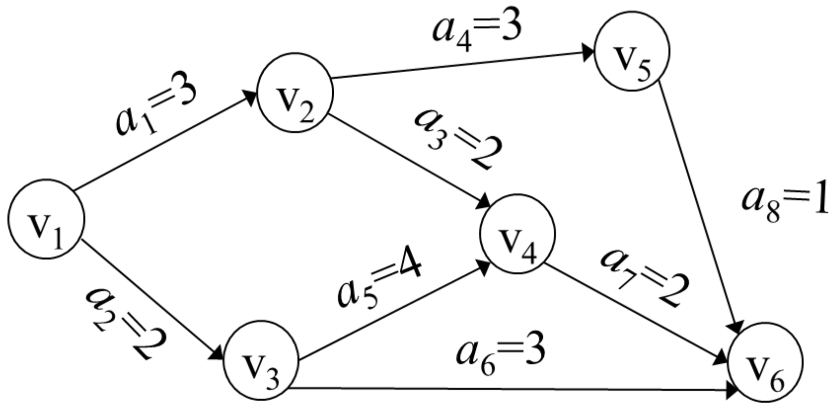
```
Status CriticalPath ( ALGraph G ) {  //输出G的各项关键活动。
  if (!TopologicalOrder (G, T)) return ERROR;
  vl[0..G.vexnum-1]=ve[G.vexnum-1]; //初始化事件最迟发生时间
  while ( ! StackEmpty (T) )          //按拓扑逆序求各顶点的vl 值
    for (Pop(T,j), p=G.vertices[j].firstarc; p; p=p->nextarc) {
      k=p->adjvex; dut=*(p->info);    //dut<j, k>
      if ( vl[k] – dut < vl[j] )    vl[j] = vl[k] – dut;
    } // for                  以j为弧尾的弧的弧头k
  for (j=0; j < G.vexnum; ++j )            //求ee、el和关键活动
    for( p=G.vertices[j].firstarc; p; p=p->nextarc){
      k = p->adjvex ;        dut = *(p->info) ;
      ee = ve[j] ;              el=vl[k]–dut ;
      tag = ( ee = = el ) ?  ‘*’ : ‘ ’ ;
      printf ( j, k, dut, ee, el, tag ) ;          // 输出关键活动
    } // for
}//CriticalPath
```





7.5.2 关键路径

关键路径



vl(汇点) = ve(汇点);
$vl(i) = \text{Min}\ \{\ vl(j) - \text{dut}(\langle i, j\rangle)\}$

```
while ( ! StackEmpty (T) )
    for (Pop(T,j), p=G.vertices[j].firstarc; p; p=p->nextarc) {
      k=p->adjvex; dut=*(p->info);    //dut<j, k>
      if ( vl[k] – dut < vl[j] )  vl[j] = vl[k] – dut;
    } // for
```

	V1	V2	V3	V4	V5	V6
Ve[ ]	0	3	2	6	6	8
Vl[ ]	**0**	**6**	**4**	**6**	**7**	**8**

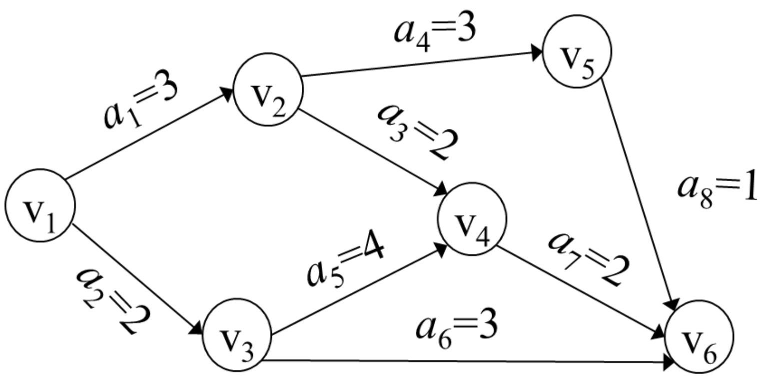
T

V6
V5
V4
V3
V2
V1

7.5.2 关键路径

◘ 关键路径

	V1	V2	V3	V4	V5	V6
Ve[ ]	0	3	2	6	6	8
Vl[ ]	**0**	**4**	**2**	**6**	**7**	**8**



$e(s) = Ve(i)$

$l(s) = Vl(j)$ - dut(<i, j>)

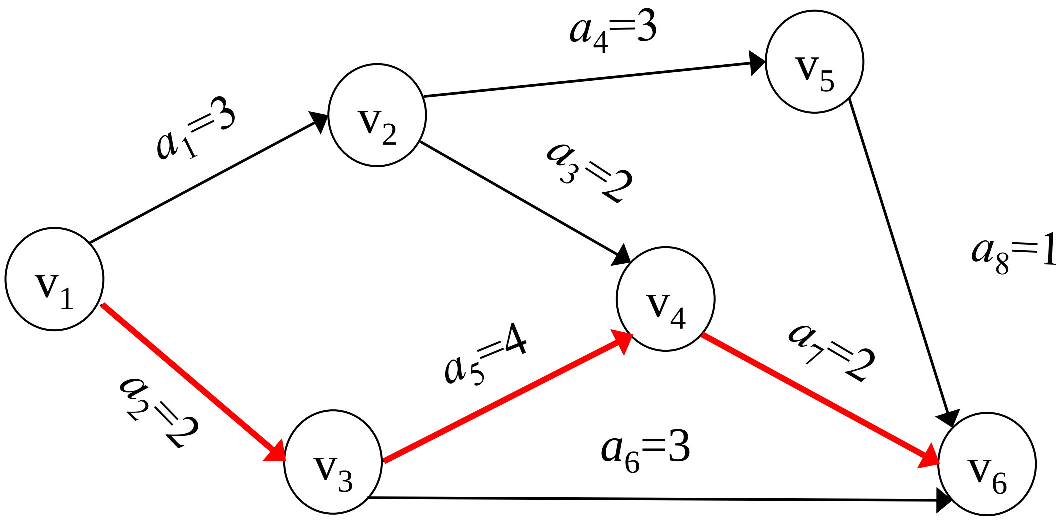
```
for (j=0; j < G.vexnum; ++j )
    for( p=G.vertices[j].firstarc; p; p=p->nextarc){
      k = p->adjvex ; dut = *(p->info) ;
      ee = ve[j] ;        el=vl[k]–dut ;
      tag = ( ee = = el ) ? '*' : ' ' ;
      printf ( j, k, dut, ee, el, tag ) ;
    } // for
```

	a1	a2	a3	a4	a5	a6	a7	a8
e[ ]	0	0	3	3	2	2	6	6
l[ ]	1	0	4	4	2	5	6	7
l-e	1	0	1	1	0	3	0	1

7.5.2 关键路径

- 关键路径

例



第7章 图

7.1 图的定义和术语

7.2 图的存储结构

7.3 图的遍历与连通性

7.4 最小生成树

7.5 活动网络

7.6 最短路径

最短路径

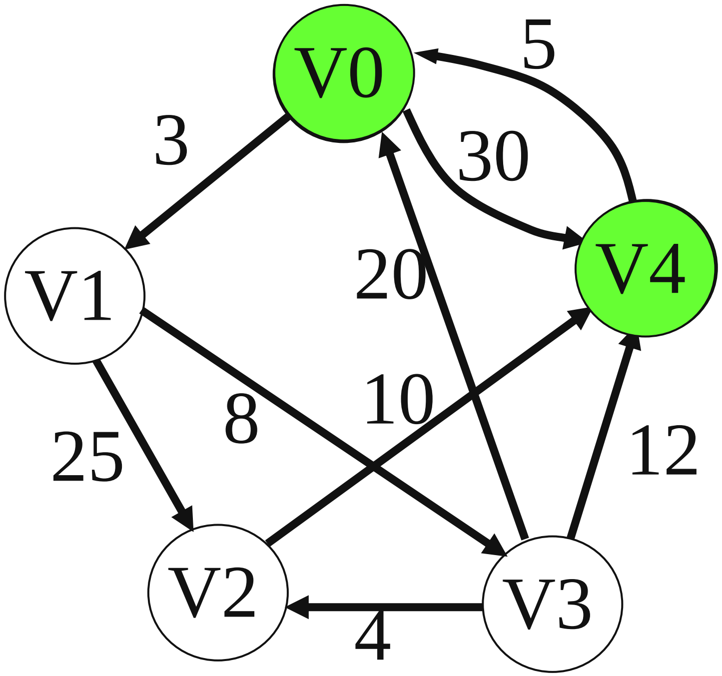
最短路径

- 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）

  - 旅游者:最经济的出行路线
  - 路由器: 最快地将数据包传送到目标位置
  - 路径规划:多边形区域内的自主机器人
  - ……

最短路径



$$
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & \\
\left(\begin{matrix} 0 & 3 & \infty & \infty & 30 \\ \infty & 0 & 25 & 8 & \infty \\ \infty & \infty & 0 & \infty & 10 \\ 20 & \infty & 4 & 0 & 12 \\ 5 & \infty & \infty & \infty & 0 \end{matrix}\right) & & & & & \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix}
\end{array}
$$

从V0到V4共有三条路径：

{V0,V4}， {V0, V1, V3, V4}， {V0, V1, V2, V4}

30 **23** **38**

最短路径

一顶点到其余各顶点

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

任意两顶点之间

最短路径——迪杰斯特拉算法

目的： 设一有向图 G=（V, E），已知各边的权值，以某指定点v_0为源点，求从v_0到图的其余各点的最短路径。限定各边上的权值大于或等于0。

最短路径

- 为求得这些最短路径，Dijkstra提出按路径长度的递增次序，逐步产生最短路径的算法。首先求出长度最短的一条最短路径，再参照它求出长度次短的一条最短路径，依次类推，直到从顶点v到其它各顶点的最短路径全部求出为止。

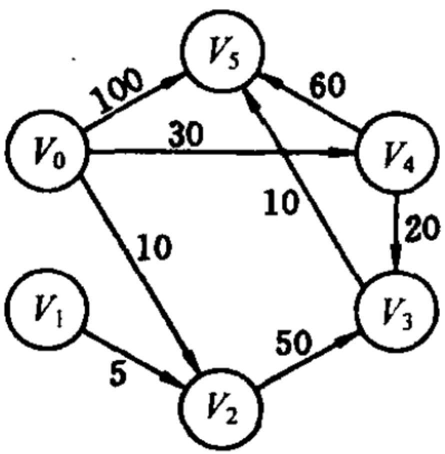


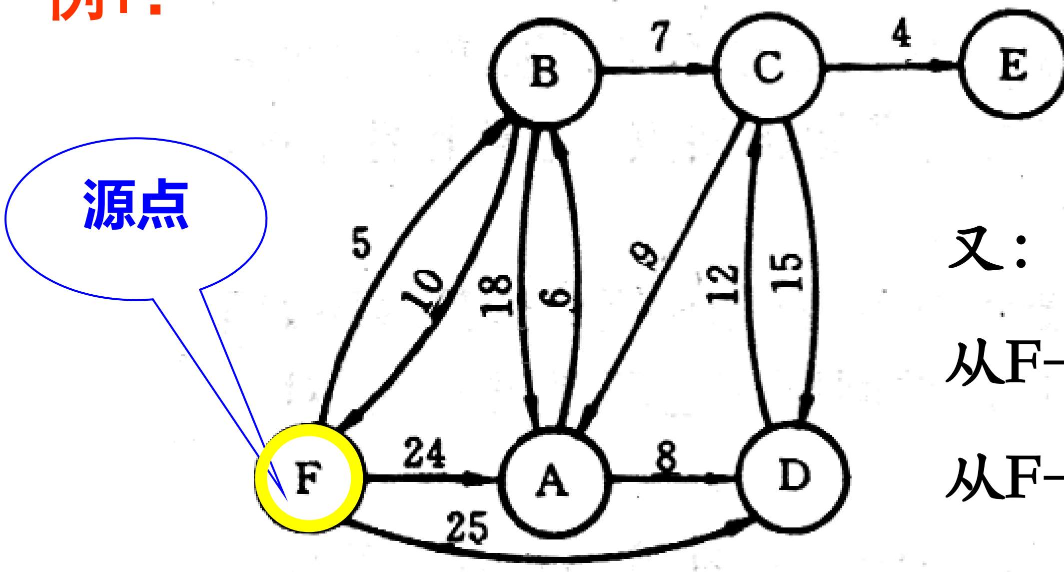
图 7.34　带权有向图 G_6

始点	终点	最短路径	路径长度
v_0	v_1	无	
	v_2	(v_0, v_2)	10
	v_3	(v_0, v_4, v_3)	50
	v_4	(v_0, v_4)	30
	v_5	(v_0, v_4, v_3, v_5)	60

图 7.35　有向图 G_6 中从 v_0 到其余各点的最短路径

最短路径——迪杰斯特拉算法

例1：



又：

从F→B的最短路径是哪条?

从F→C的最短路径是哪条?

从F→A的路径有4条:

① F→A:　　24

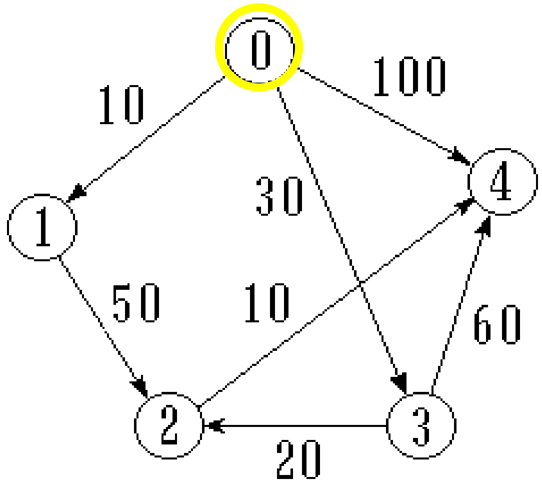
② F→B→A:　5＋18=23

③ F→B→C→A: 5+7+9=21

④ F→D→C→A: 25+12+9=36

最短路径——迪杰斯特拉算法

例2：



（a）带权有向图

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

（b）邻接矩阵

源点	终点	最　短　路　径	路径长度
v_0	v_1	(v_0, v_1)	10
	v_2	(v_0, v_1, v_2) (v_0, v_3, v_2)	60 50
	v_3	(v_0, v_3)	30
	v_4	(v_0, v_4) (v_0, v_3, v_4) (v_0, v_1, v_2, v_4) (v_0, v_3, v_2, v_4)	100 90 70 60

最短路径——迪杰斯特拉算法

算法思想：

（1）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）从这些路径中找出一条长度最短的路径（v_0,u）,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）。其中，（v_0,u）为上次求得的v_0到u最短路径；

（3）在调整后的各条路径中，再找长度最短的路径，依此类推。

总之，按路径“长度”递增的次序来逐步产生最短路径

最短路径——迪杰斯特拉算法

（1）设A[n][n]为有向网的带权邻接矩阵，A[i][j]表示弧（v_i, v_j)的权值，S为已找到从源点v_0出发的最短路径的终点集合，它的初始状态为{ v_0 }，辅助数组D[n]为源点v_0到各终点当前找到的最短路径的长度，它的初始值为：D[i] = A[v_0,i] //即邻接矩阵中第v_0行的权值;

最短路径——迪杰斯特拉算法

（2）选择j，使得

$D[j] = \min\{D[i] \mid v_i \in V\text{-}S\}$

/* v_i是S集之外的顶点，j是当前离源点距离最短的顶点， D[j]是从源点v_0到S集合之外所有顶点的路径中距离最短的一条*/

$S = S \cup \{v_j\}$ //将j加入S集

最短路径——迪杰斯特拉算法

（3）对于所有不在S中的终点v_k，若

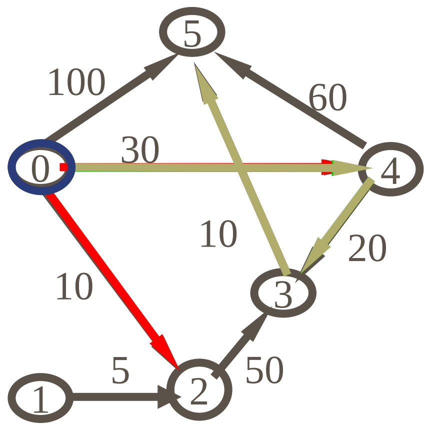
$D[j]+A[j,k]<D[k]$ // 即$(v_0,j)+(j,k)<(v_0,k)$

则修改D[k]为：$D[k]=D[j]+A[j,k]$

D[j]为上次求得从源点到v_j点的最短路径，v_j为上次求得最短路径的点。

（4）重复操作(2)、(3)共n-1次，由此求得从v_0到各终点的最短路径。

最短路径——迪杰斯特拉算法



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

S之外的当前最短路径之顶点

终点	D[w]	从v_0到各终点的D值和最短路径		
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	$\{v_0,v_2\}$	$\{v_0,v_2,v_4\}$	$\{v_0,v_2,v_4,v_3\}$	$\{v_0,v_2,v_4,v_3,v_5\}$

(v0,v2)+ (v2,v3)$<$(v0,v3)

与最小生成树的不同点：路径可能是累加的！

最短路径——迪杰斯特拉算法

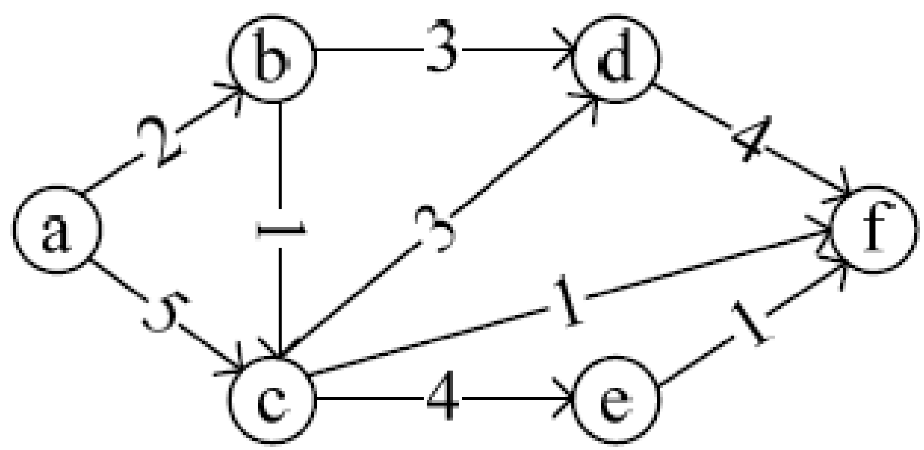
如下有向带权图,若采用迪杰斯特拉算法求源点a到其他各顶点的最短路径,得到的第一条最短路径的目标顶点是b,第二条最短路径的目标顶点是c,后续得到的其余各最短路径的目标顶点依次是()

A **d,e,f**

B **e,d,f**

C **f,d,e**

D **f,e,d**

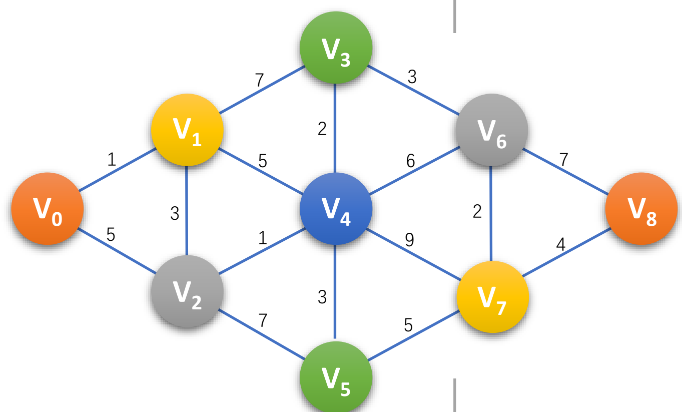


最短路径——迪杰斯特拉算法

关键代码

```
void ShortestPath_Dijkstra(MGraph G, int v0, Patharc &P, ShortPathTable &D){
    int v,w,k,min;
    int final[MAXVEX];
    for(v=0; v<G.numVertexes; v++) {
        final[v] = 0;
        D[v] = G.arc[v0][v];
        P[v] = -1;
    }

    D[v0] = 0;
    final[v0] = 1;
    for(v=1; v<G.numVertexes; v++) {
        min=GRAPH_INFINITY;
        for(w=0; w<G.numVertexes; w++) {
            if(!final[w] && D[w]<min) {
                k=w;
                min = D[w];
            }
        }
        final[k] = 1;
        for(w=0; w<G.numVertexes; w++)    {
            if(!final[w] && (min+G.arc[k][w]<D[w]))    {
                D[w] = min + G.arc[k][w];
                P[w]=k;
            }
        }
    }
}
```



数据变化

$$\begin{array}{c} \\ V_0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \\ V_8 \end{array}
\begin{array}{c} \begin{array}{ccccccccc} V_0 & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 & V_8 \end{array} \\
\begin{bmatrix}
0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\
1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\
5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\
\infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\
\infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\
\infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\
\infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\
\infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\
\infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0
\end{bmatrix} \end{array}$$

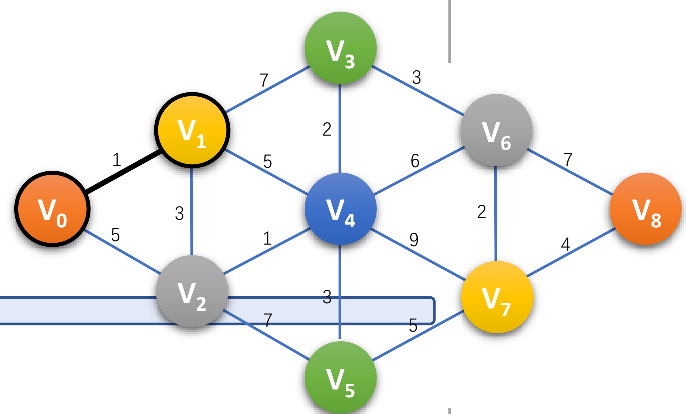
final	D	P
1	0	-1
0	1	-1
0	5	-1
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1

最短路径——迪杰斯特拉算法

关键代码

```
void ShortestPath_Dijkstra(MGraph G, int v0, Patharc &P, ShortPathTable &D){
    int v,w,k,min;
    int final[MAXVEX];
    for(v=0; v<G.numVertexes; v++) {
        final[v] = 0;
        D[v] = G.arc[v0][v];
        P[v] = -1;
    }

    D[v0] = 0;
    final[v0] = 1;
    for(v=1; v<G.numVertexes; v++) {
        min=GRAPH_INFINITY;
        for(w=0; w<G.numVertexes; w++) {
            if(!final[w] && D[w]<min) {
                k=w;
                min = D[w];
            }
        }
        final[k] = 1;
        for(w=0; w<G.numVertexes; w++)    {
            if(!final[w] && (min+G.arc[k][w]<D[w]))    {
                D[w] = min + G.arc[k][w];
                P[w]=k;
            }
        }
    }
}
```



数据变化

$$
\begin{array}{c|ccccccccc}
 & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\
\hline
v_0 & 0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\
v_1 & 1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\
v_2 & 5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\
v_3 & \infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\
v_4 & \infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\
v_5 & \infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\
v_6 & \infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\
v_7 & \infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\
v_8 & \infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0
\end{array}
$$

final	D	P
1	0	-1
1	1	-1
0	5	-1
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1

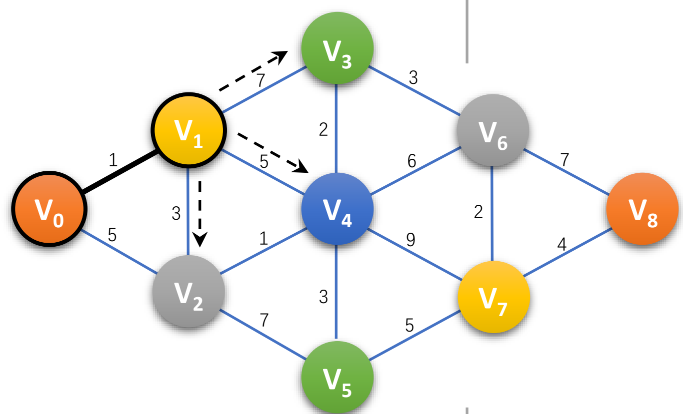
v=1
w=0~9
min=1
k=1
final[k]=final[1]=1

最短路径——迪杰斯特拉算法

关键代码

```
void ShortestPath_Dijkstra(MGraph G, int v0, Patharc &P, ShortPathTable &D){
    int v,w,k,min;
    int final[MAXVEX];
    for(v=0; v<G.numVertexes; v++) {
        final[v] = 0;
        D[v] = G.arc[v0][v];
        P[v] = -1;
    }

    D[v0] = 0;
    final[v0] = 1;
    for(v=1; v<G.numVertexes; v++) {
        min=GRAPH_INFINITY;
        for(w=0; w<G.numVertexes; w++) {
            if(!final[w] && D[w]<min) {
                k=w;
                min = D[w];
            }
        }
        final[k] = 1;
        for(w=0; w<G.numVertexes; w++)   {
            if(!final[w] && (min+G.arc[k][w]<D[w]))   {
                D[w] = min + G.arc[k][w];
                P[w]=k;
            }
        }
    }
}
```



数据变化

$$
\begin{array}{c|ccccccccc}
 & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \\
\hline
v_0 & 0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\
v_1 & 1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\
v_2 & 5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\
v_3 & \infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\
v_4 & \infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\
v_5 & \infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\
v_6 & \infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\
v_7 & \infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\
v_8 & \infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0
\end{array}
$$

final	D	P
1	0	-1
1	1	-1
0	**4**	**1**
0	**8**	**1**
0	**6**	**1**
0	65535	-1
0	65535	-1
0	65535	-1
0	65535	-1

v=1
w=0~9
min=1
k=1
final[k]=final[1]=1

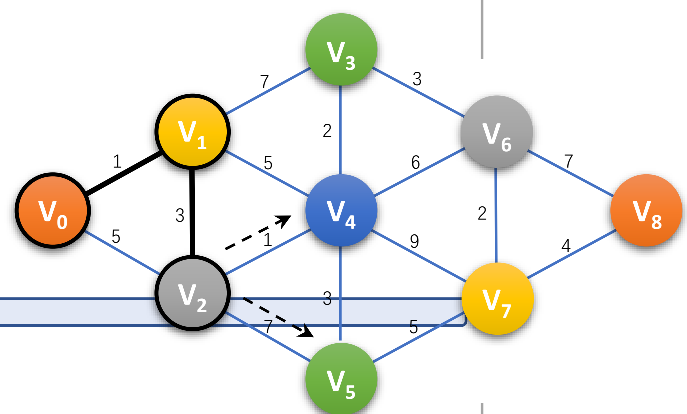
v_0-v_1-v_2: 1+3=4
v_0-v_1-v_3: 1+7=8
v_0-v_1-v_4: 1+5=6

最短路径——迪杰斯特拉算法

关键代码

```
void ShortestPath_Dijkstra(MGraph G, int v0, Patharc &P, ShortPathTable &D){
    int v,w,k,min;
    int final[MAXVEX];
    for(v=0; v<G.numVertexes; v++) {
        final[v] = 0;
        D[v] = G.arc[v0][v];
        P[v] = -1;
    }

    D[v0] = 0;
    final[v0] = 1;
    for(v=1; v<G.numVertexes; v++) {
        min=GRAPH_INFINITY;
        for(w=0; w<G.numVertexes; w++) {
            if(!final[w] && D[w]<min) {
                k=w;
                min = D[w];
            }
        }
        final[k] = 1;
        for(w=0; w<G.numVertexes; w++)    {
            if(!final[w] && (min+G.arc[k][w]<D[w]))    {
                D[w] = min + G.arc[k][w];
                P[w]=k;
            }
        }
    }
}
```



数据变化

$$
\begin{array}{c}
 \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{array} \\
\begin{bmatrix}
0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\
1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\
5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\
\infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\
\infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\
\infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\
\infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\
\infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\
\infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0
\end{bmatrix}
\end{array}
$$

final	D	P
1	0	-1
1	1	-1
1	4	1
0	8	1
0	**5**	**2**
0	**11**	**2**
0	65535	-1
0	65535	-1
0	65535	-1

v=2
w=0~9
min=4
k=2
final[k]=final[2]=1

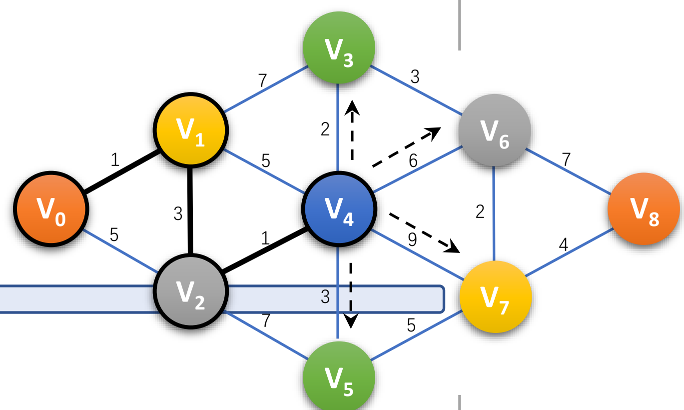
v_0-v_1-v_2-v_4: 1+3+1=5
v_0-v_1-v_2-v_5: 1+3+7=11

最短路径——迪杰斯特拉算法

关键代码

```
void ShortestPath_Dijkstra(MGraph G, int v0, Patharc &P, ShortPathTable &D){
    int v,w,k,min;
    int final[MAXVEX];
    for(v=0; v<G.numVertexes; v++) {
        final[v] = 0;
        D[v] = G.arc[v0][v];
        P[v] = -1;
    }

    D[v0] = 0;
    final[v0] = 1;
    for(v=1; v<G.numVertexes; v++) {
        min=GRAPH_INFINITY;
        for(w=0; w<G.numVertexes; w++) {
            if(!final[w] && D[w]<min) {
                k=w;
                min = D[w];
            }
        }
        final[k] = 1;
        for(w=0; w<G.numVertexes; w++)    {
            if(!final[w] && (min+G.arc[k][w]<D[w]))    {
                D[w] = min + G.arc[k][w];
                P[w]=k;
            }
        }
    }
}
```



数据变化

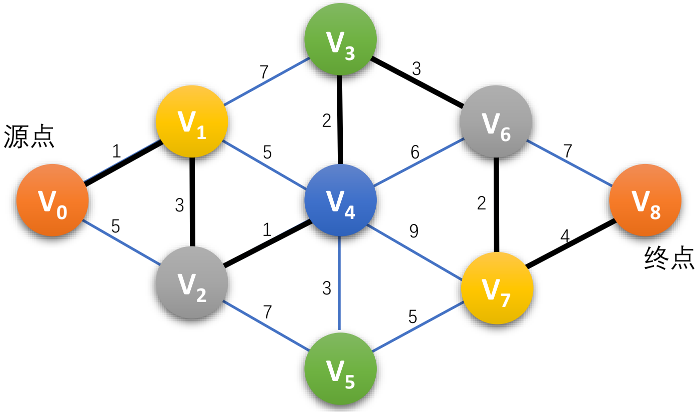
$$
\begin{array}{c|c}
 & \begin{matrix} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{matrix} \\
\begin{matrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{matrix} &
\begin{bmatrix}
0 & 1 & 5 & \infty & \infty & \infty & \infty & \infty & \infty \\
1 & 0 & 3 & 7 & 5 & \infty & \infty & \infty & \infty \\
5 & 3 & 0 & \infty & 1 & 7 & \infty & \infty & \infty \\
\infty & 7 & \infty & 0 & 2 & \infty & 3 & \infty & \infty \\
\infty & 5 & 1 & 2 & 0 & 3 & 6 & 9 & \infty \\
\infty & \infty & 7 & \infty & 3 & 0 & \infty & 5 & \infty \\
\infty & \infty & \infty & 3 & 6 & \infty & 0 & 2 & 7 \\
\infty & \infty & \infty & \infty & 9 & 5 & 2 & 0 & 4 \\
\infty & \infty & \infty & \infty & \infty & \infty & 7 & 4 & 0
\end{bmatrix}
\end{array}
$$

final	D	P
1	0	-1
1	1	-1
1	4	1
0	**7**	**4**
1	5	2
0	**8**	**4**
0	**11**	**4**
0	**14**	**4**
0	65535	-1

v=4
w=0~9
min=5
k=4
final[k]=final[4]=1

v_0-v_1-v_2-v_4-v_3: 1+3+1+2=7
v_0-v_1-v_2-v_4-v_5: 1+3+1+3=8
v_0-v_1-v_2-v_4-v_6: 1+3+1+6=11
v_0-v_1-v_2-v_4-v_7: 1+3+1+9=14

最短路径——迪杰斯特拉算法



正在答疑