第三部分

图(网络)结构应用

第四部分

算法设计方法应用

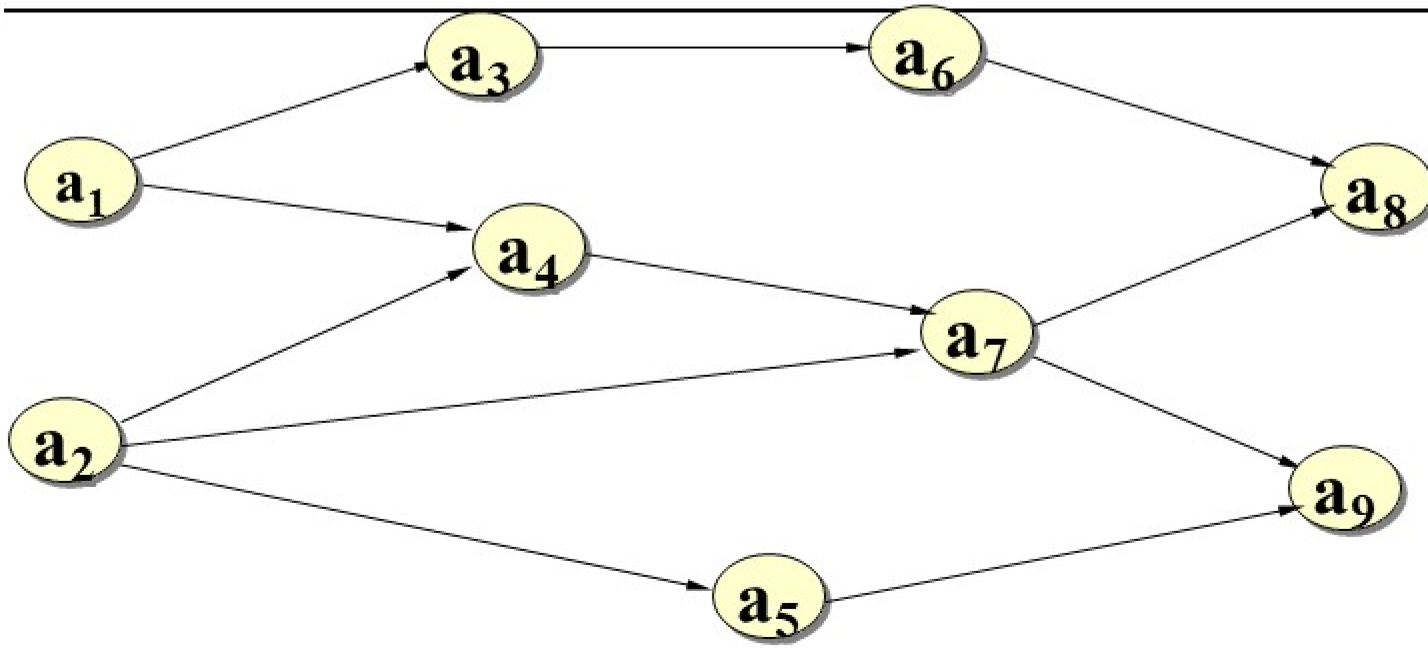
算法设计方法

- 基本的算法设计方法：
  - 贪婪算法
  - 分而治之算法
  - 动态规划
  - 回溯
  - 分支定界

任务调度问题

- 一个工程项目由一组子任务构成，子任务之间有的可以并行执行，有的必须在完成了其它一组子任务后才能执行。
- 给出一个可行的任务调度方案
- 或 判定一个给定的任务调度是否可行。

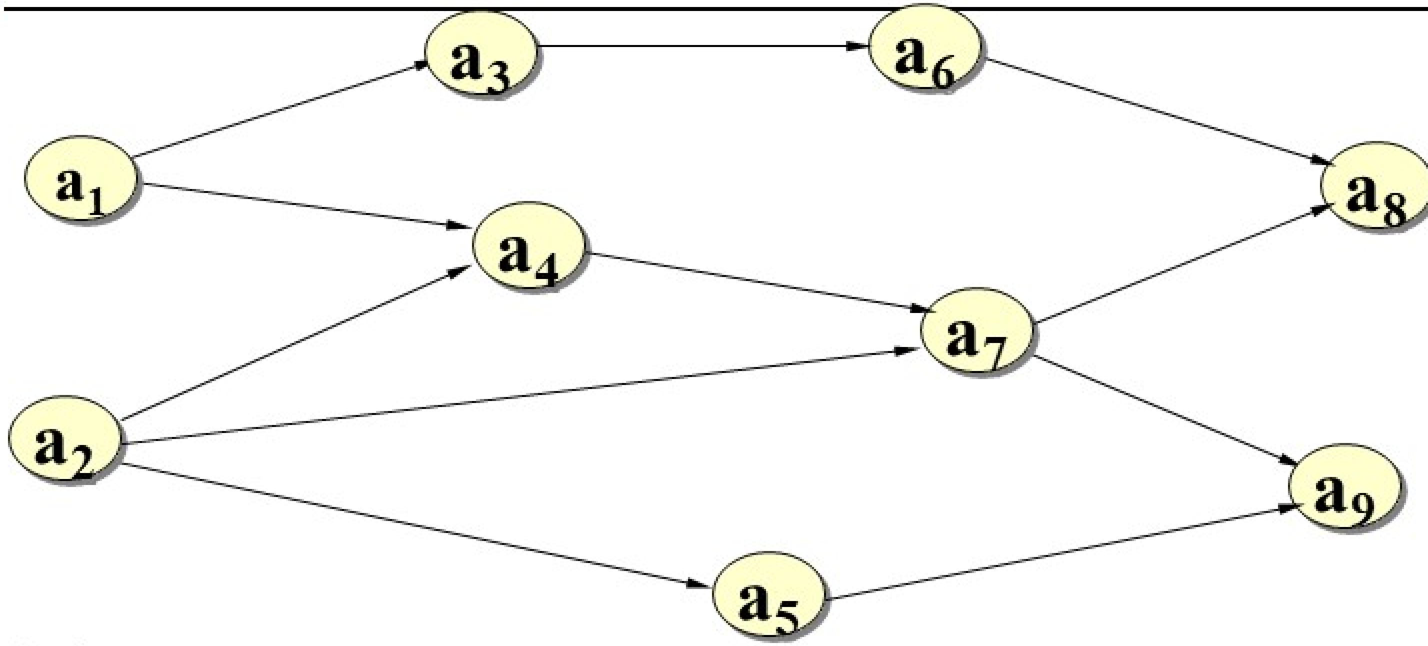
任务调度问题



- 使用**AOV**表示工程项目，
  - 顶点表示子任务，
  - 从$\mathbf{a_i}$到$\mathbf{a_j}$的有向边$(\mathbf{a_i}, \mathbf{a_j})$:

  子任务$\mathbf{a_i}$完成后，子任务$\mathbf{a_j}$才能开始执行；

任务调度问题



- 思考:
  - 如果给出整个工程项目中完成每个活动(子任务)的所需时间，如何计算完成整个工程项目所需要的时间?

任务调度问题

- 整个工程项目中每个活动(子任务)的所需时间
- 活动之间的先后关系

⇨ 计算完成整个工程项目所需要的最短时间？

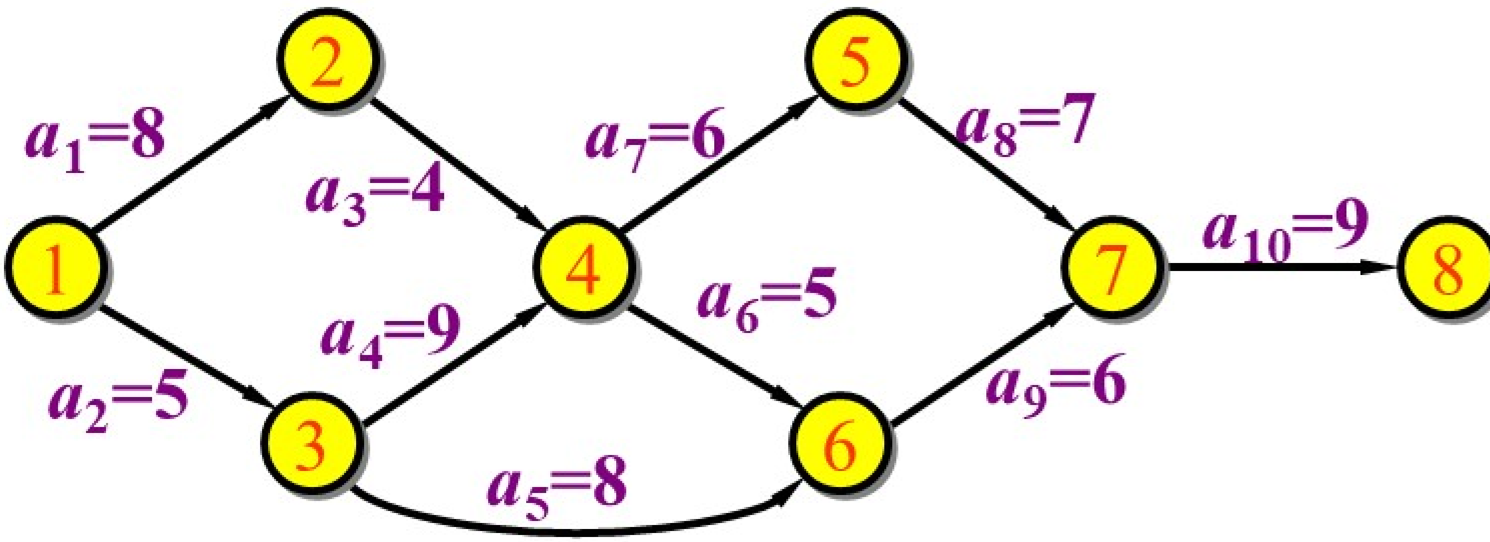
- **为缩短完成工程所需的时间，应当加快哪些活动?**
- 关键活动：在所有的活动中，有些活动即使推迟几天完成，也不会影响全局的工期；但是有些活动必须按时完成，否则真个项目的工期就要因此延误，这种活动就叫“关键活动”。
  - ➢ 如何确定工程中的“关键活动”？

用边表示活动的网络(AOE网络)

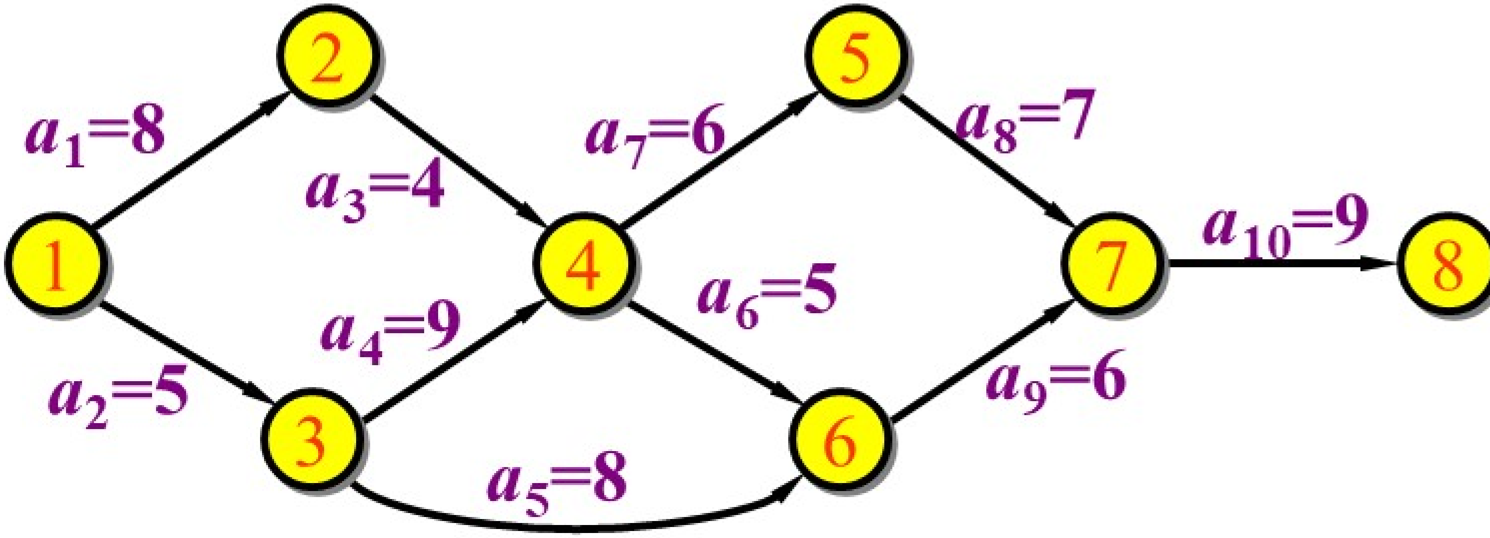
- 如果在加权有向无环图中
  - 用有向边表示一个工程中的各项活动(Activity)(或任务)
  - 用边上的权值表示活动的持续时间(Duration)
  - 用顶点表示事件(Event)
- 则这样的有向图叫做用边表示活动的网络，简称**AOE (Activity On Edges)网络**。
- AOE网络在某些工程估算方面非常有用。例如，计算：
  - (1) 完成整个工程至少需要多少时间(假设网络中没有环)?
  - (2) 为缩短完成工程所需的时间, 应当加快哪些活动?

AOE网络

- 在AOE网络中,有些活动顺序进行,有些活动并行进行。
- 源点(开始事件):入度为0;汇点(结束事件):出度为0。
- 从源点到各个顶点,以至从源点到汇点的有向路径可能不止一条。这些路径的长度也可能不同。完成不同路径的活动所需的时间虽然不同,但只有各条路径上所有活动都完成了,整个工程才算完成。



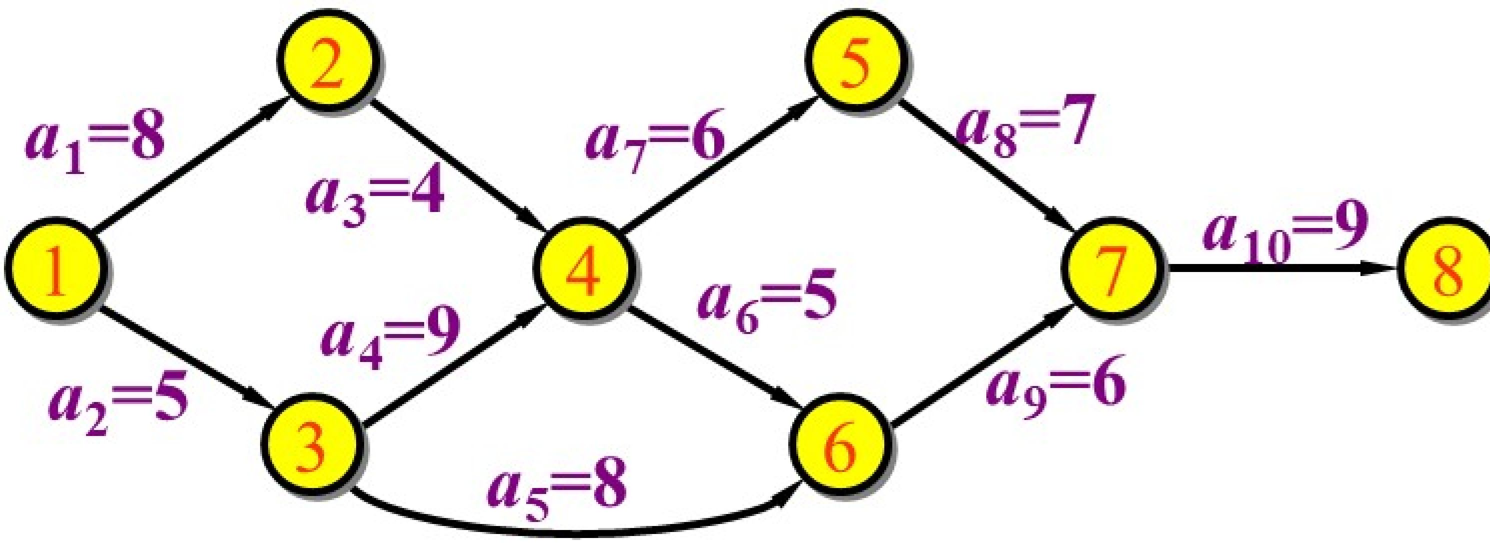
关键路径



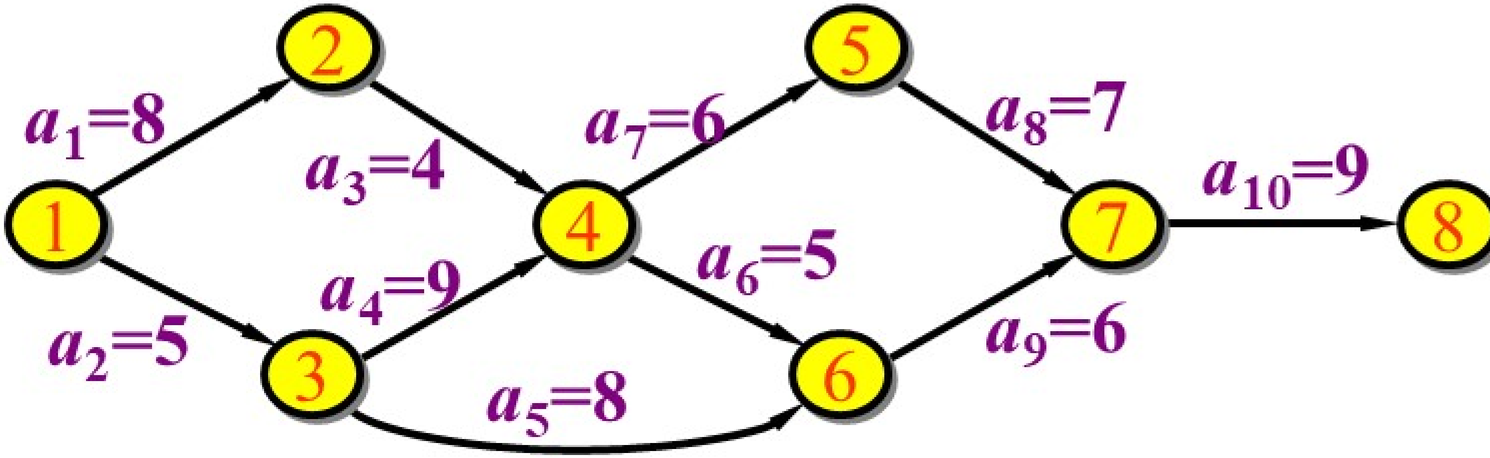
- 完成整个工程所需的时间取决于从源点到汇点的**最长路径长度**，即在这条路径上所有活动的持续时间之和。这条路径长度最长的路径就叫做**关键路径**(Critical Path)。

关键活动

- 要找出关键路径，必须找出关键活动，即**不按期完成就会影响整个工程完成的活动**。
- 关键路径上的所有活动都是关键活动。因此，只要找到了关键活动，就可以找到 关键路径



几个与计算关键活动有关的量



☆ 事件 ***i*** 的最早发生时间 ***Ve*(*i*)**

是从源点 ***1*** 到顶点 ***i*** 的最长路径长度。

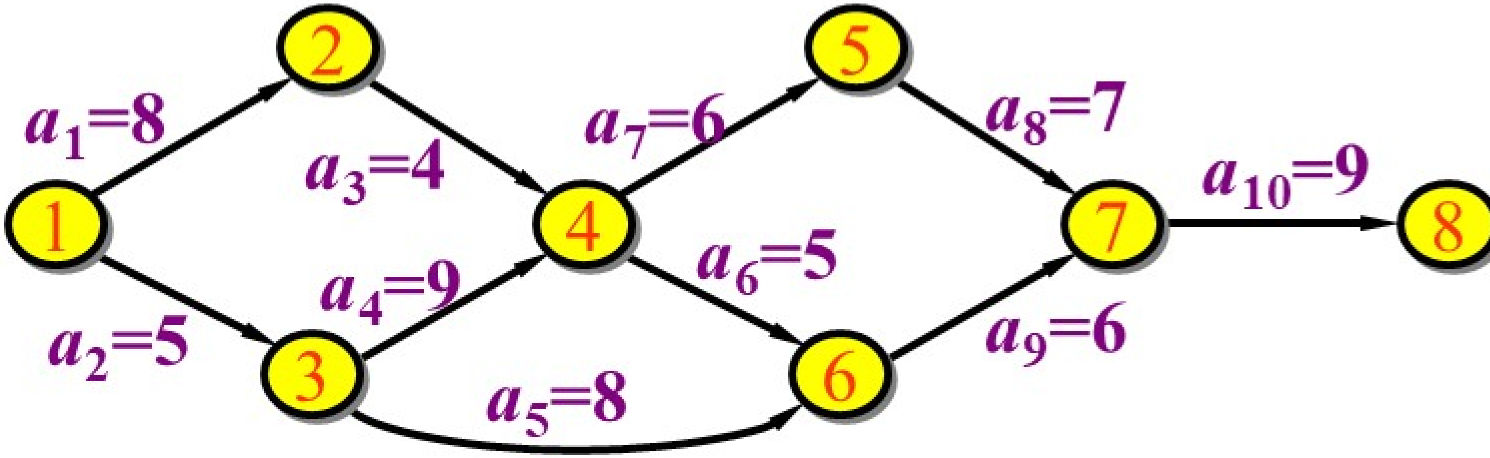
Ve(2)=8; *Ve*(3)=5; ……, *Ve*(8)=36

☆ 事件 ***i*** 的最迟发生时间 ***Vl*(*i*)**

是在保证汇点 ***n*** 在 ***Ve*(*n*)** 时刻发生的前提下，事件 ***i*** 的允许的最迟发生时间。

Vl(*8*)=36; *Vl*(*7*)=27;

几个与计算关键活动有关的量



☆活动 a_k 的最早开始时间 $e(k)$

➢ 设活动 a_k 在边 (i, j) 上，则 $e(k)$ 是从源点 1 到顶点 i 的最长路径长度。因此，$e(k) = Ve[i]$。

☆ 活动 a_k 的最迟开始时间 $l(k)$

➢ $l(k)$ 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。

$$l(k) = Vl[j] - length(i, j)。$$

其中，$length(i, j)$ 是完成 a_k 所需的时间。

几个与计算关键活动有关的量

- 活动 a_k 的延迟时间 $l(k)$ - $e(k)$

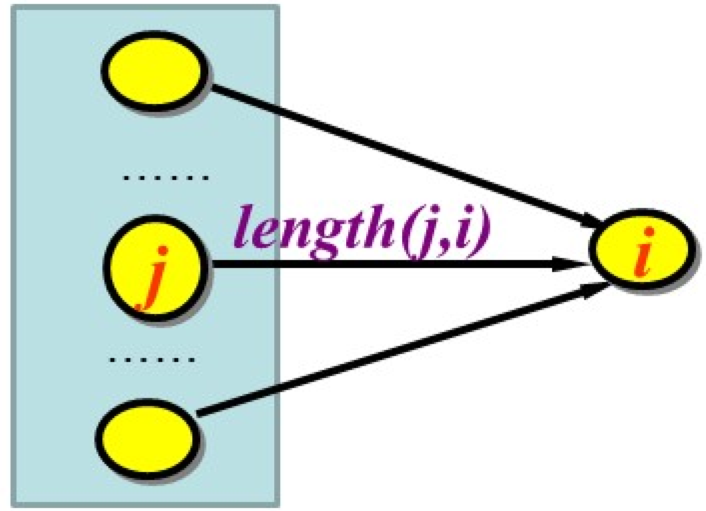
  表示活动 a_k 的最早开始时间和最迟开始时间的延迟时间(时间余量)。

- $l(k) == e(k)$: 表示活动 a_k 没有延迟时间,是关键活动。

- 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 $l(k) == e(k)$.

- 为求得 $e(k)$ 与 $l(k)$,需要先求得各个顶点 i 的 $Ve(i)$ 和 $Vl(i)$。

计算*Ve(i)*, *Vl(i)*

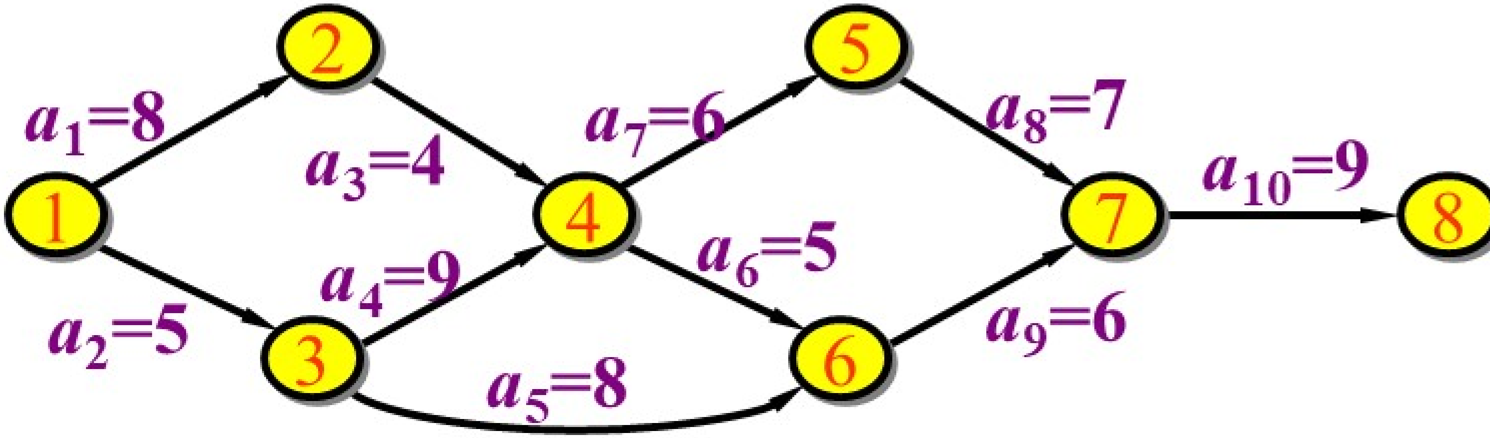


- 求$Ve(i)$ 的递推公式
  - 从$\mathbf{Ve(1) = 0}$开始，向前递推
  - $Ve(i) = \max\{Ve(j)+\text{length}(j,i)\}$

$$<j, i> \in E, \quad i = 2, 3, \ldots, n$$

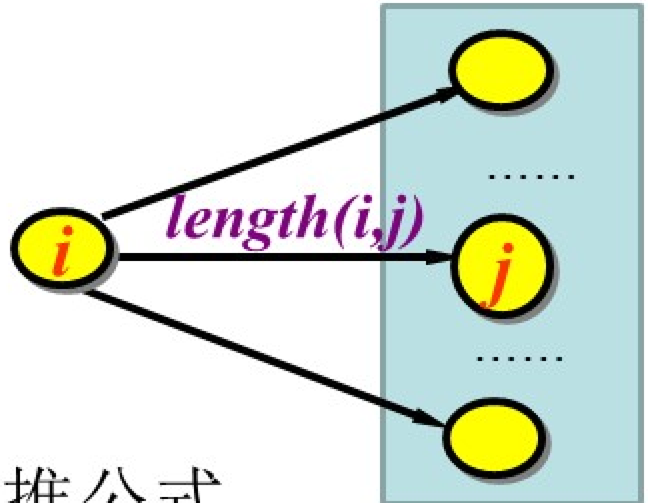
- 递推公式的计算必须在**拓扑有序**的前提下进行。

计算*Ve(i)*示例



- $Ve(1) = 0$
- 拓扑序列：1，2，3，4，5，6，7，8
- $Ve(2) = 8$； $Ve(3) = 5$；
- $Ve(4) = \max\{Ve(2) + 4, Ve(3) + 9\} = 14$;
- $Ve(5) = Ve(4) + 6 = 20$；
- $Ve(6) = \max\{Ve(3) + 8, Ve(4) + 5\} = 19$;
- $Ve(7) = \max\{Ve(5) + 7, Ve(6) + 6\} = 27$;
- $Ve(8) = Ve(7) + 9 = 36$；

计算$Ve(i)$, $Vl(i)$

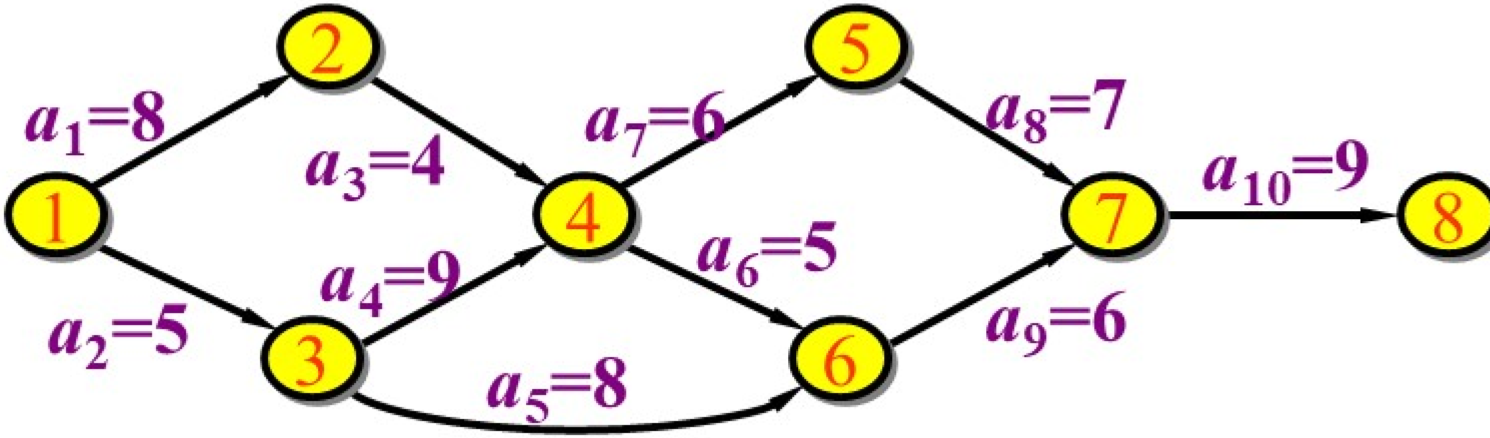


- 求$Vl(i)$的递推公式
  - 从$Vl(n) = Ve(n)$开始，反向递推
  - $Vl(i) = \min\{Vl(j) - \text{length}(i,j)\}$

    $<i,j> \in E, \quad n-1, n-2, \ldots, 1$

- 递推公式的计算必须在**逆拓扑有序**的前提下进行。

计算$Vl(i)$示例



- $Vl(8) = Ve(8) = 36$
- $Vl(7) = Vl(8) - 9 = 27$;
- $Vl(6) = Vl(7) – 6 = 21$; $Vl(5) = Vl(7) – 7 = 20$；
- $Vl(4) = min\{Ve(6) -5, Vl(5) -6\} = 14$;
- $Vl(3) = min\{Vl(4)-9, Vl(6)-8\} = 5$;
- $Vl(2) = Vl(4) – 4 = 10$;
- $Vl(1) = 0$

计算$e(k), l(k)$

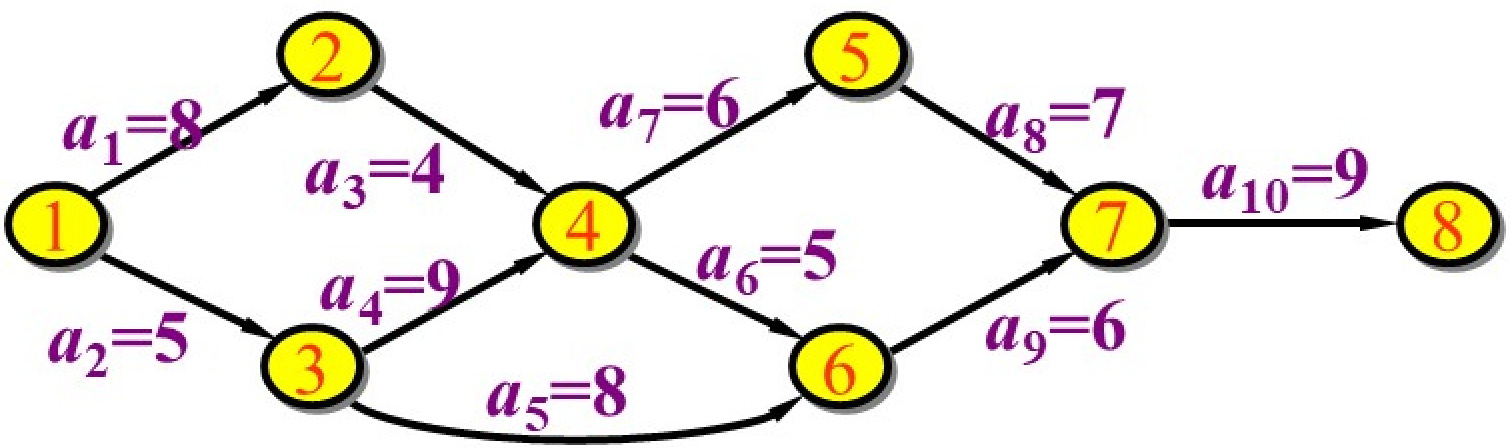
- 设活动 a_k (k =1,2,…,e)在带权有向边 (i, j) 上, 它的持续时间用length(i, j) 表示，则有

$e(k) = Ve(i)$;

$l(k) = Vl(j)$ - length (i, j); k=1,2,…,e。

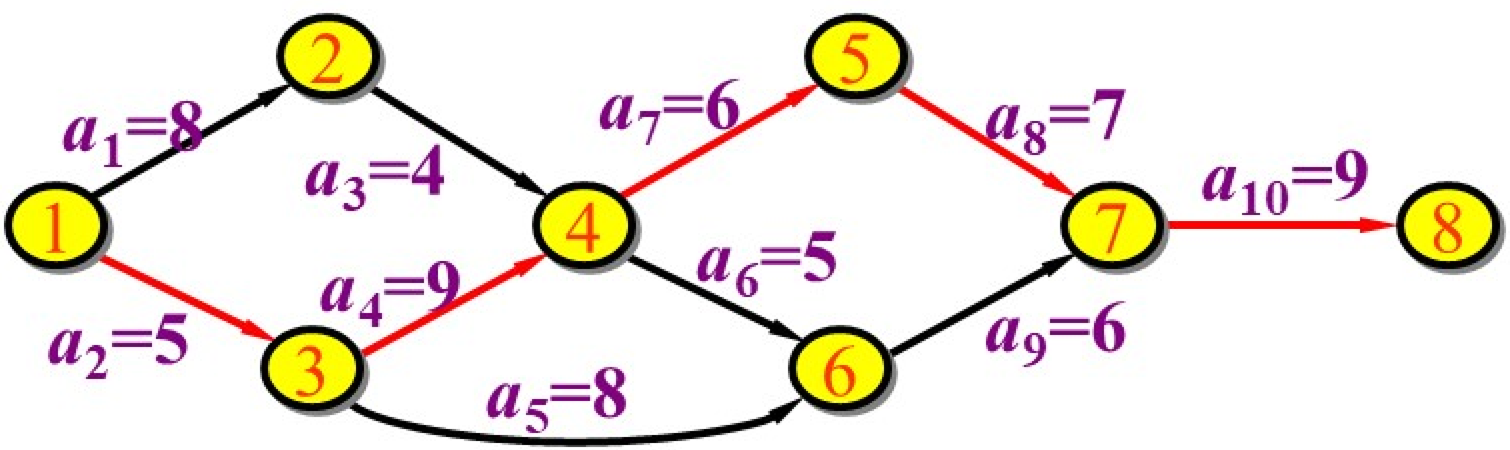
	1	2	3	4	5	6	7	8
Ve	0	8	5	14	20	19	27	36
Vl	0	10	5	14	20	21	27	36

	1	2	3	4	5	6	7	8	9	10
e	0	0	8	5	5	14	14	20	19	27
l	2	0	10	5	13	16	14	20	21	27



	1	2	3	4	5	6	7	8
Ve	0	8	5	14	20	19	27	36
Vl	0	10	5	14	20	21	27	36

	1	2	3	4	5	6	7	8	9	10
e	0	0	8	5	5	14	14	20	19	27
l	2	0	10	5	13	16	14	20	21	27

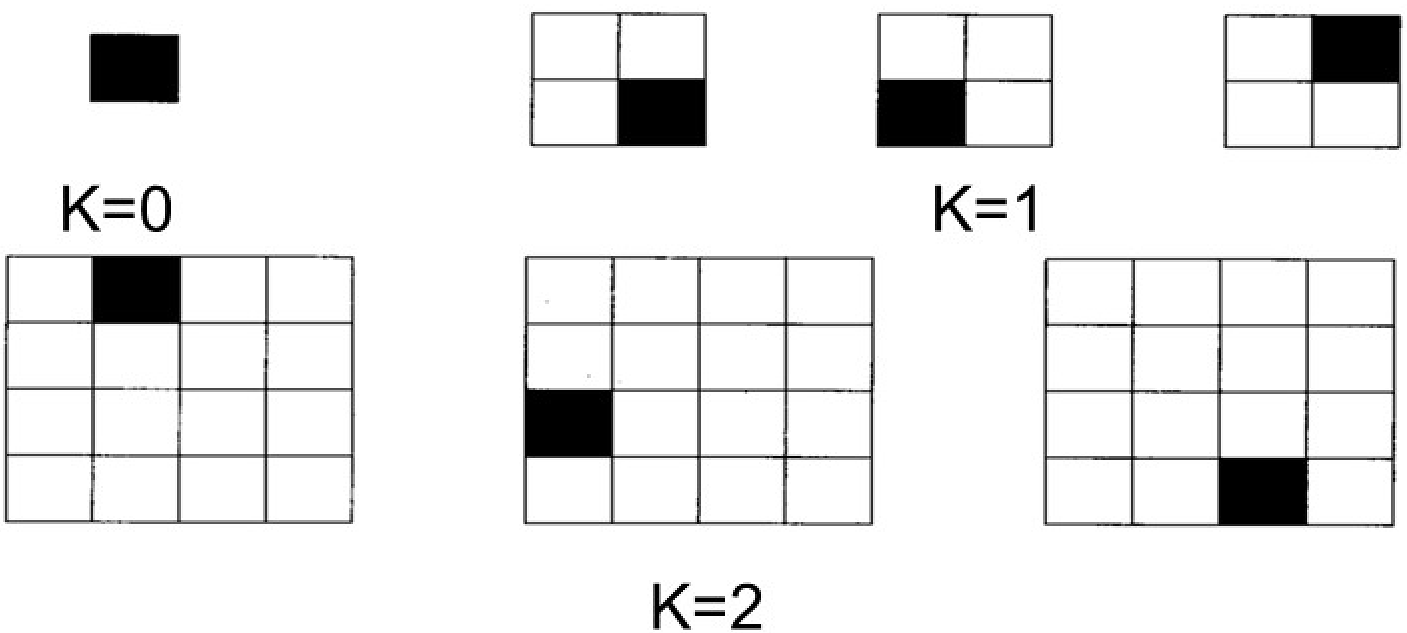


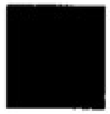
算法设计

- 设计算法时
  - 可以一边进行拓扑排序一边计算各顶点的 $Ve(i)$。
  - 如果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。算法在求 $Ve(i)$, i=1,2,…,n时按拓扑有序的顺序计算，在求 $Vl[i]$,i=n,n-1,n-2,…,1时按逆拓扑有序的顺序计算。

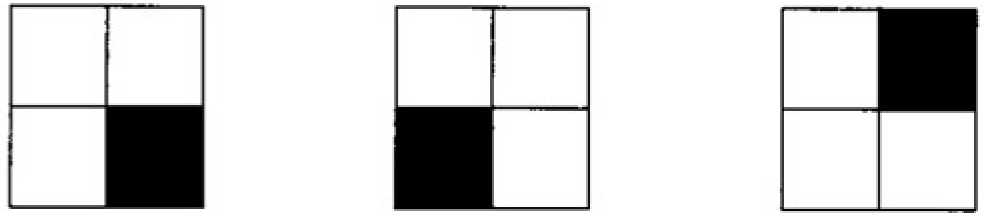
残缺棋盘的问题

- 残缺棋盘（defective chessboard）：
- 是一个有$2^k \times 2^k$ 个方格的棋盘，其中恰有一个方格残缺。
- k=0，1，2时各种可能的残缺棋盘，其中残缺方格用阴影表示。

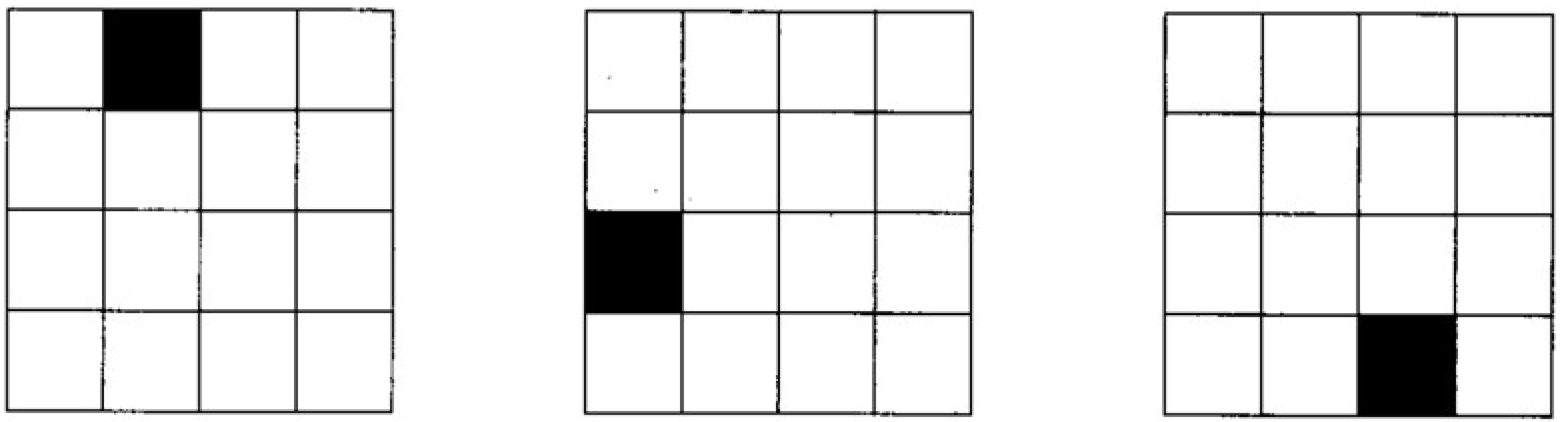




K=0



K=1

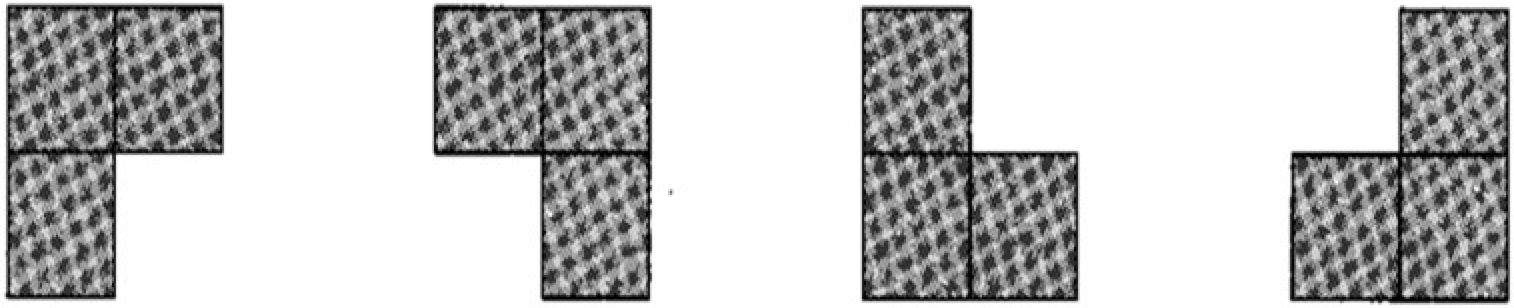


K=2

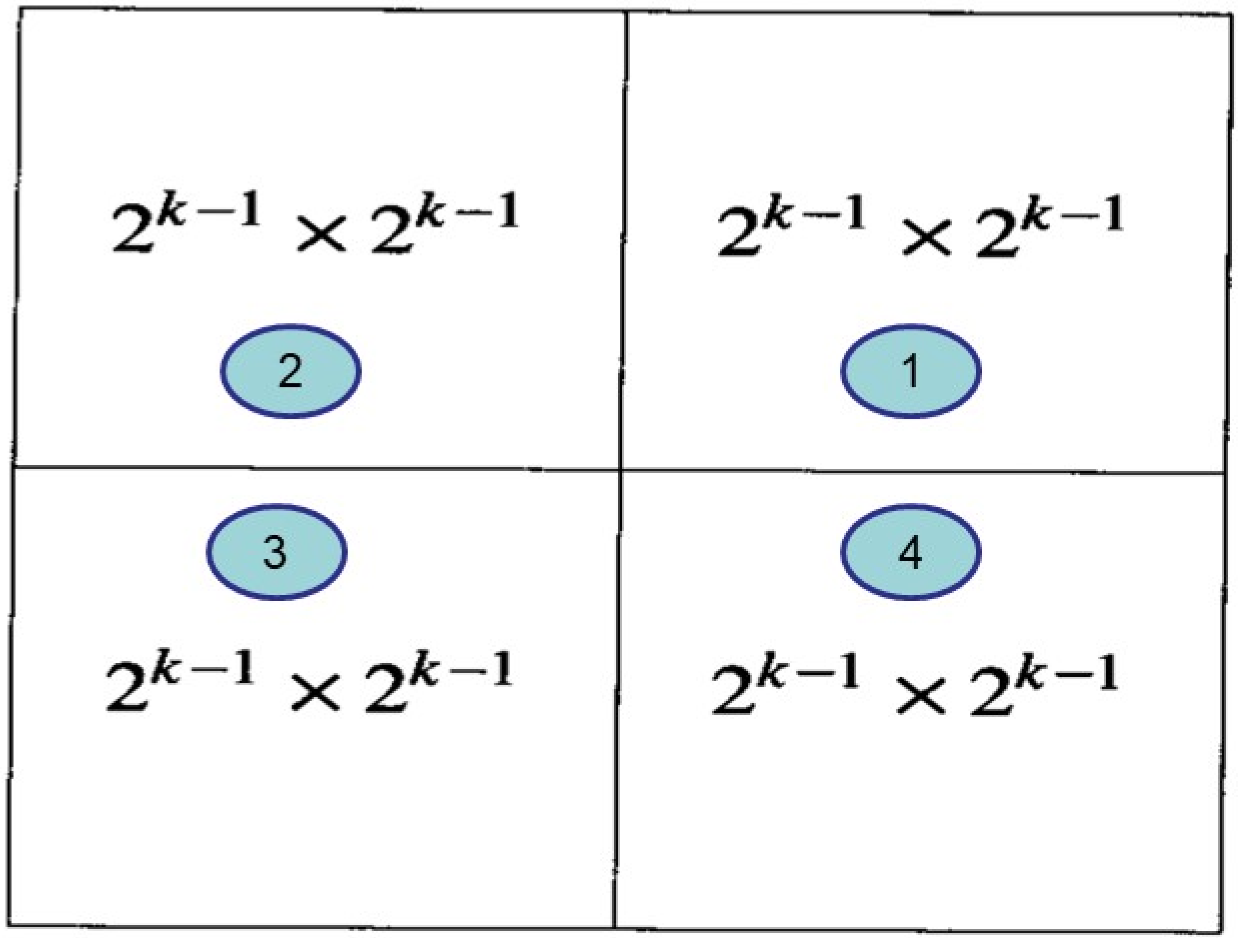
- 对于任意k，恰好存在2^{2k}种不同的残缺棋盘。

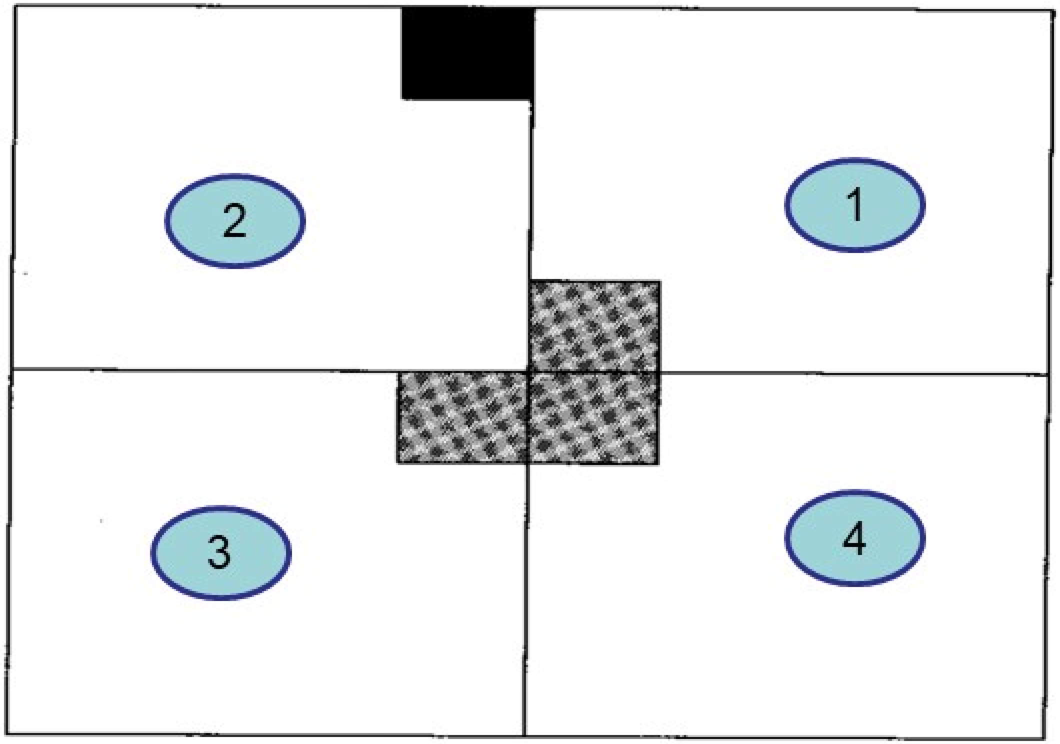
残缺棋盘的问题

- 残缺棋盘的问题:
  - 要求用三格板（triominoes）覆盖残缺棋盘。
  - 在此覆盖中，两个三格板不能重叠，三格板不能覆盖残缺方格，但必须覆盖其他所有的方格。



划分棋盘





- 覆盖 $2^k \times 2^k$ 残缺棋盘的问题转化为4个$2^{k-1} \times 2^{k-1}$覆盖棋盘
- 设 $2^k \times 2^k$ 残缺棋盘的边长为size,
  - $2^{k-1} \times 2^{k-1}$覆盖棋盘的边长为 size/2

思考

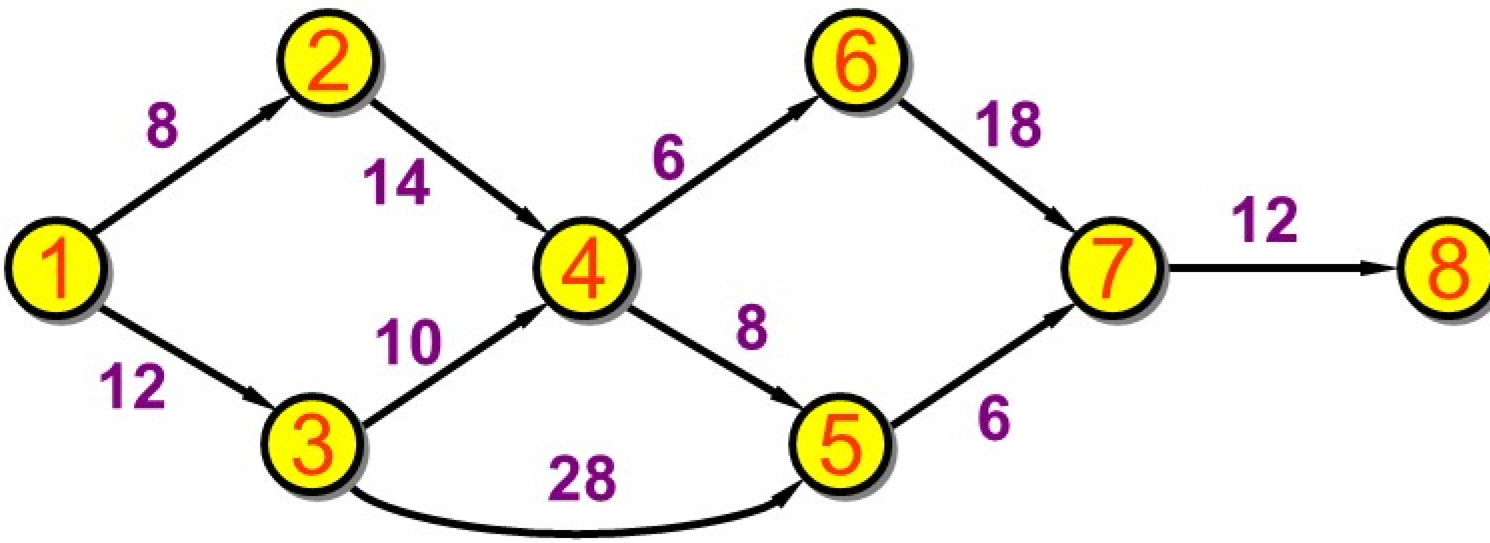
- 输出覆盖后的棋盘，输出棋盘时要着色，共享同一边界的覆盖应着不同的颜色。棋盘是平面图，因此最多只需4种颜色，为覆盖着色，要求设计算法，以尽量使用较少的颜色。

选址问题

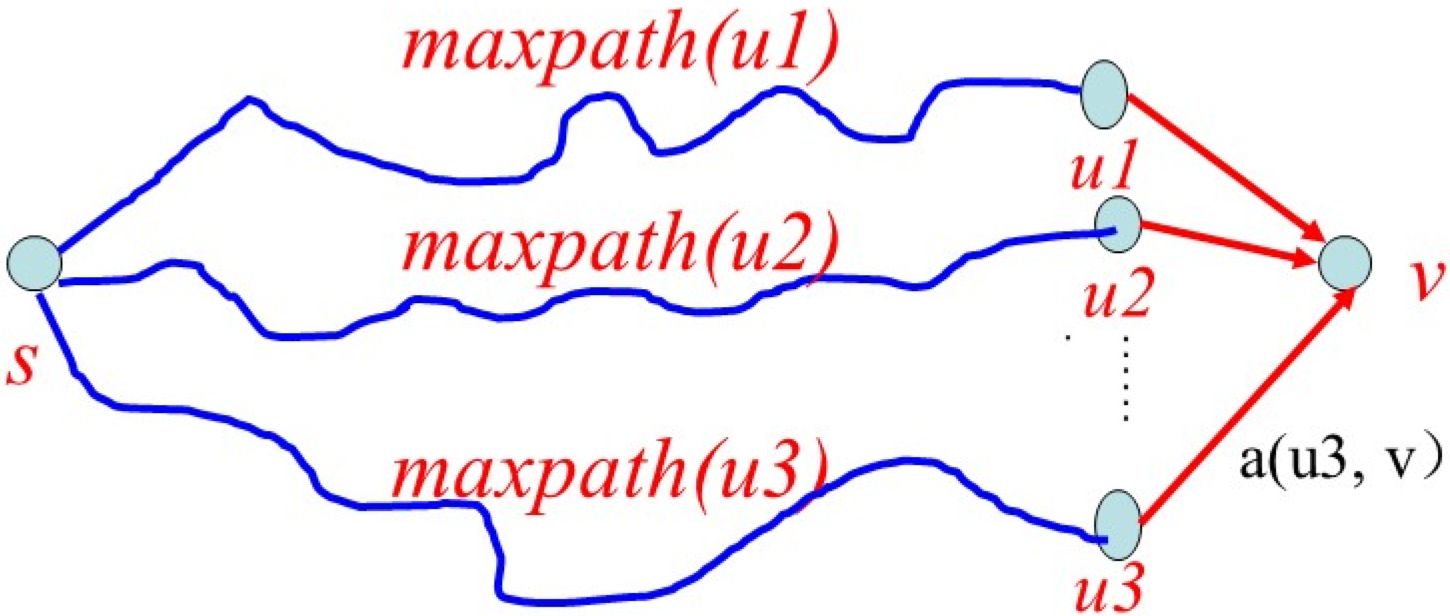
- 选址问题，是指为一个和几个服务设施在一定区域内选定它的位置，使某一指标达到最优解。这类问题，在规划建设中经常可以碰到，这里所谓的服务设施，可以是某些公共服务设施，如医院，消防站，物流中心等。也可以是生产服务设施，如仓库，转运站等等。

加权有向图最长路径问题

- 问题描述
  - 给定一个有向加权无环图G=(V,E)，从G中找出无入度的顶点s，求从s出发到其他各顶点的最长路径。



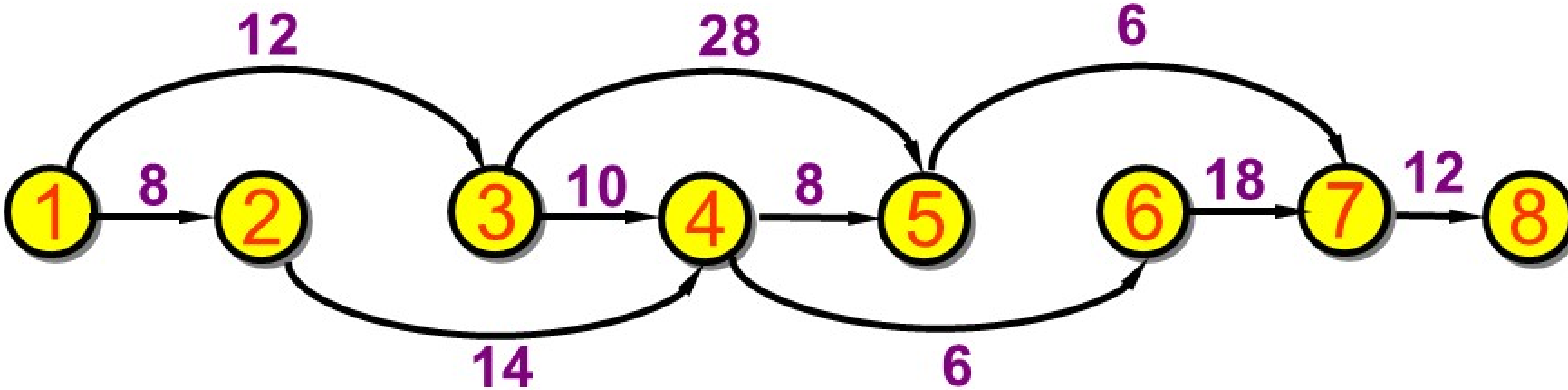
maxpath(v)



- maxpath(v):从s出发到顶点v的最长路径
- |V|=n，|E|=e,图的邻接矩阵a
- **maxpath(s)= 0**
- **maxpath(v)=max{maxpath(u)+ a(u,v);}**

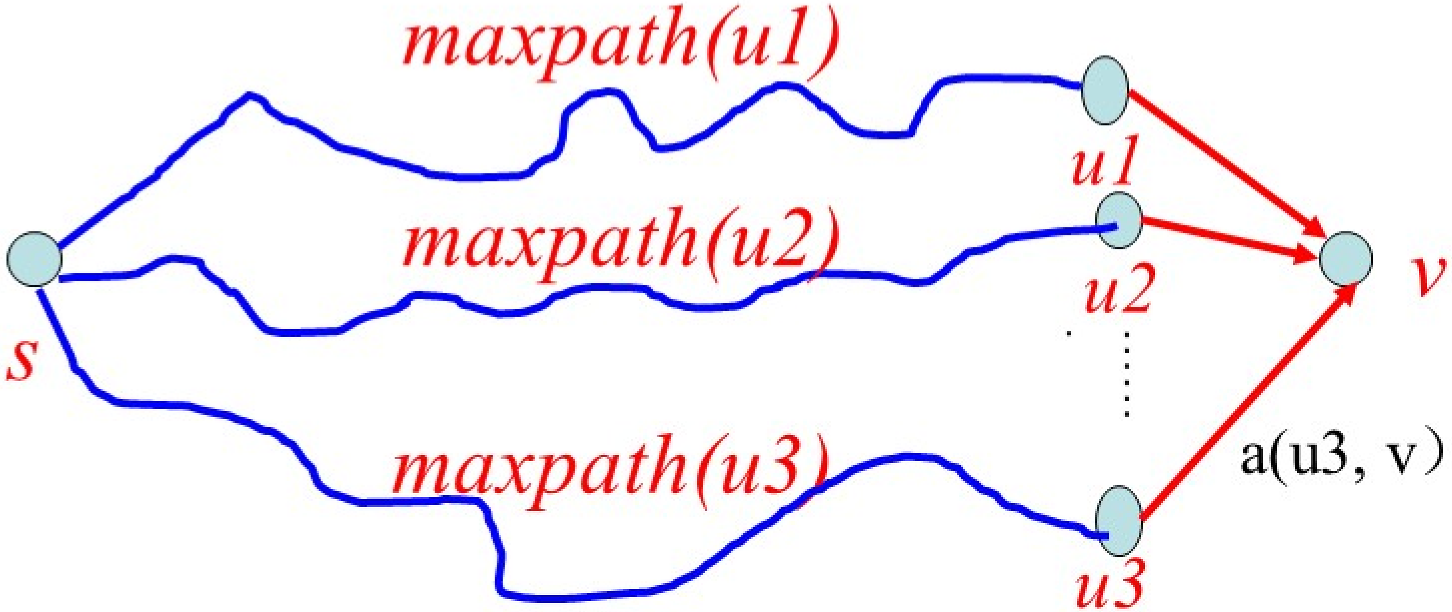
$$\text{maxpath}(v)=\max_{U}\{\text{maxpath}(u)+a(u,v);\ (u,v)\in E\}$$

*maxpath(v)*示例



- s=1
- maxpath(5)=?
- $a(3,5)=28$; $a(4,5)=8$;
- maxpath(5)=max{maxpath(3)+28; maxpath(4)+8}

maxpath(v)



- **maxpath(s)= 0**
- $\mathbf{maxpath(v)=\max_{U}\{maxpath(u)+\ a(u,v);\}_{(u,v)\in E}}$

- 如何高效实现？