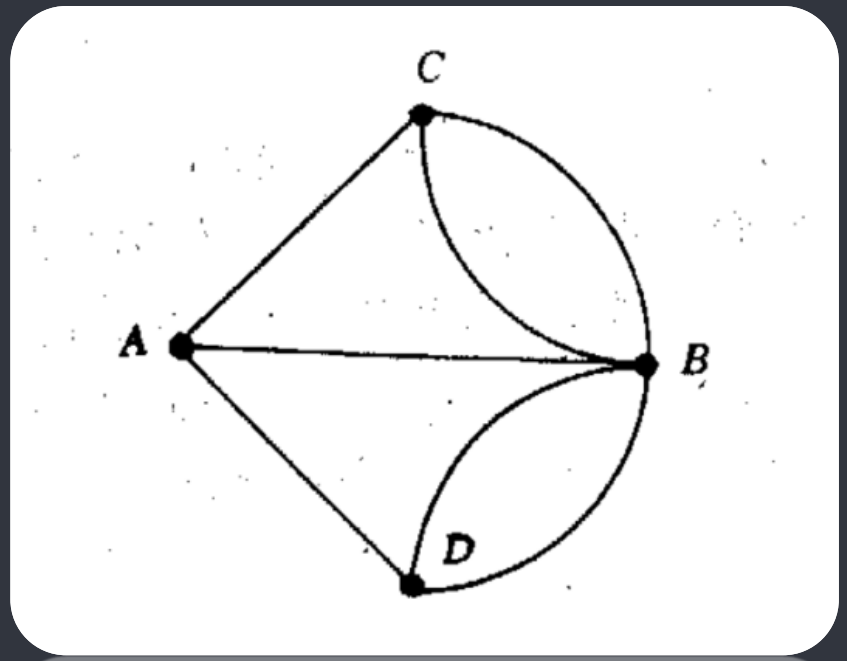
图论基础

邓丝雨

牛客竞赛
AC.NOWCODER.COM

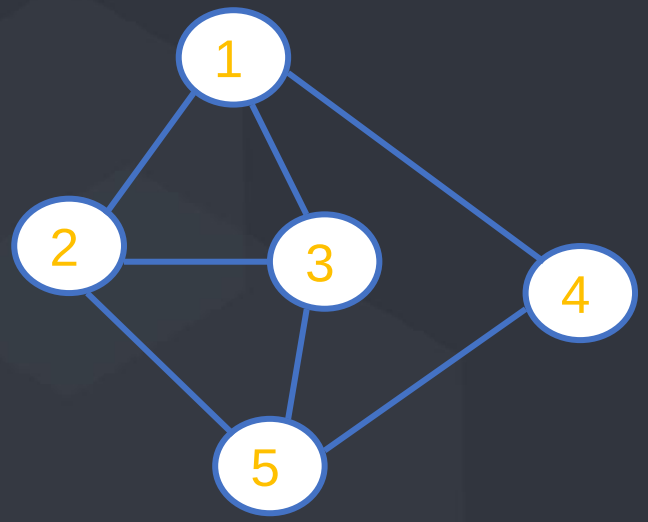
- 图论起源于著名的哥尼斯堡七桥问题——从这四块陆地中任何一块开始，通过每一座桥正好一次，再回到起点。欧拉在1736年解决了这个问题，欧拉证明了这个问题没有解，并且推广了这个问题，给出了对于一个给定的图可以某种方式走遍的判定法则。这就是后来的欧拉路径和欧拉回路。这项工作使欧拉成为图论〔及拓扑学〕的创始人。

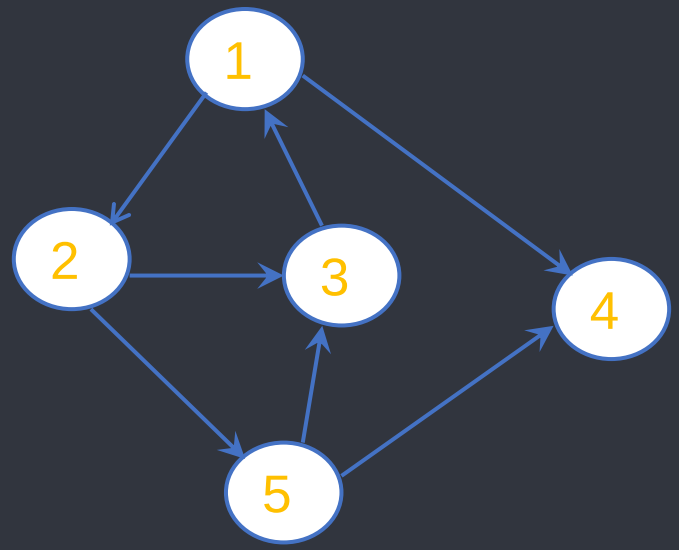


基本概念

图是什么?

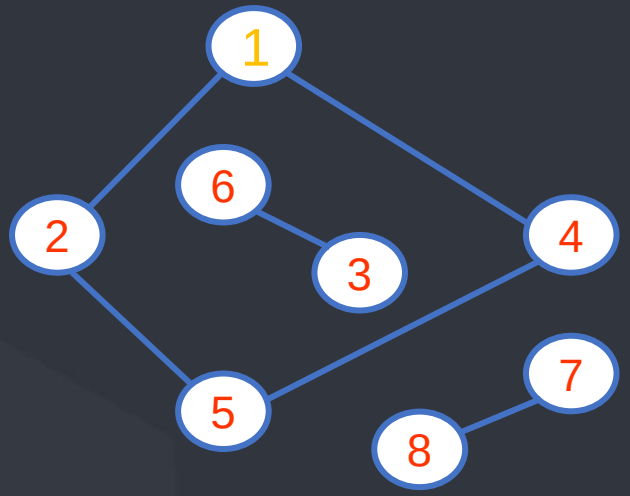
- 图的的定义
- 图是由顶点V的集合和边E的集合组成的二元组：
- 记G=（V，E)
- 存在一个结点v，可能含有多个前驱结点和后继结点。





图的分类

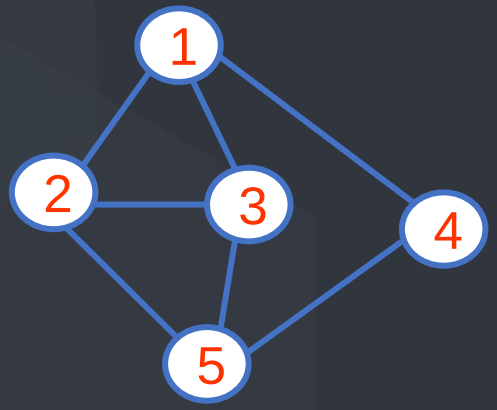
- 有向图
- 无向图
- 无权图
- 带权图
- 连通图
- 二分图
- …

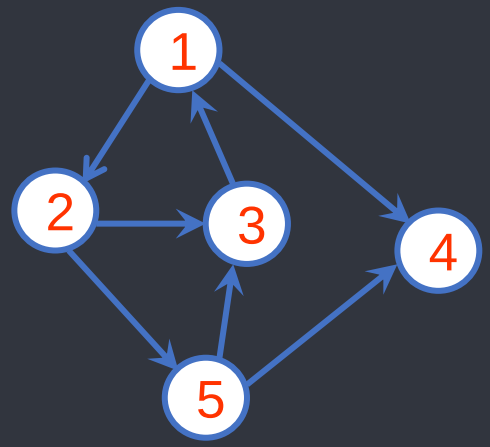


- 有向图:　点与有向边的集合
- 带权图（网）：图中的边加上表示某种含义的数值，数值称为边的权
- 连通：两顶点间有路可通。
- 连通图：能连成一片的图。
- 连通分量：无向图中的极大连通子图

路径

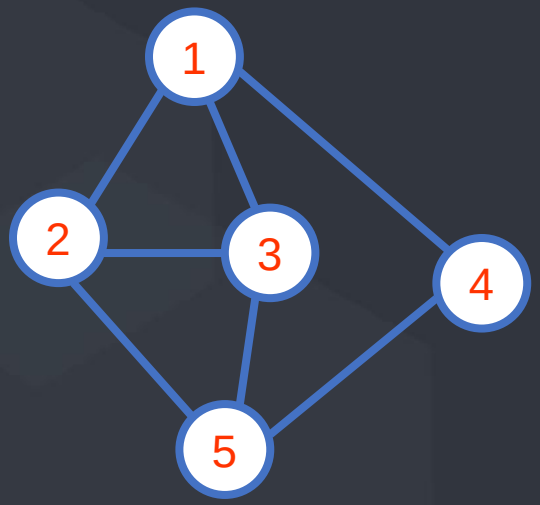
- 在图G=（V，E）中，如果对于结点a，b，存在满足下述条件的结点序列$x_1……x_k(k>1)$
- ⑴ $x_1=a$，$x_k=b$ ⑵ $(x_i，x_{i+1})\in E$ $i=1..k-1$
- 则称结点序列$x_1=a，x_2，…，x_k=b$为结点a到结点b的一条路径，而路径上边的数目（k-1）称为该路径的长度。
- 若起点与终点相同着为环（也叫做回路）

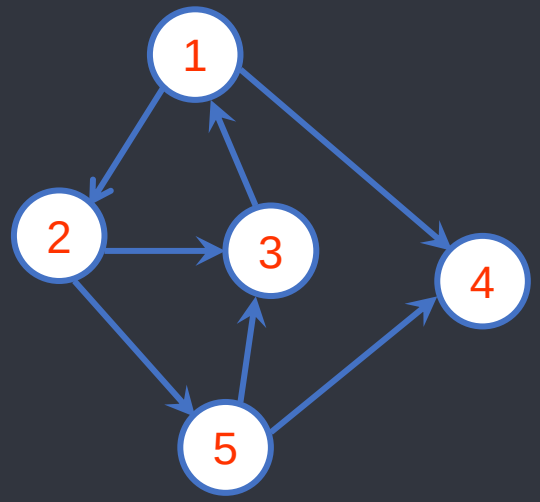




顶点的度、入度、初度

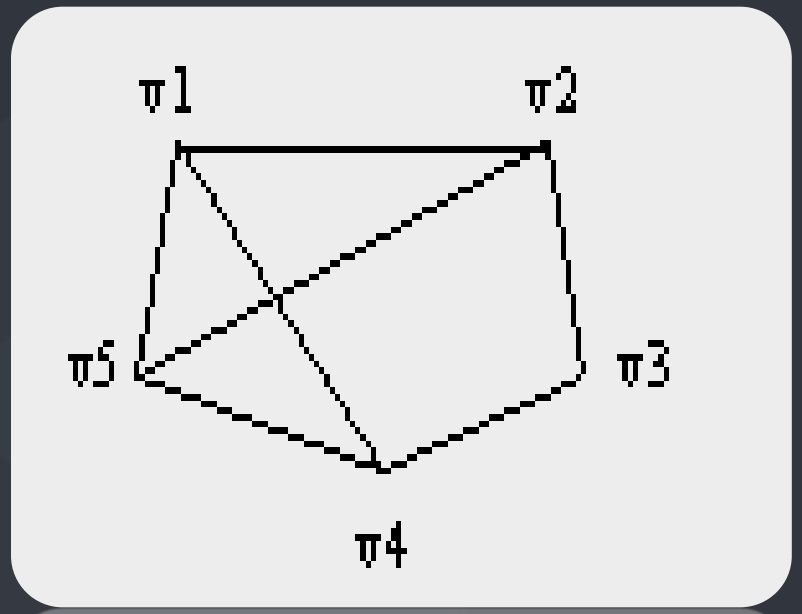
- 在无向图中：顶点v的度是指与顶点v相连的边的数目。D(2)=3
- 在有向图中：
  入度——以该顶点为终点的边的数目 ID(3)=2
- 出度——以该顶点为起点的边的数目 OD(3)=1
- 度：等于该顶点的入度与出度之和。D(5)=ID(5)+OD(5)=1+2=3





图的存储

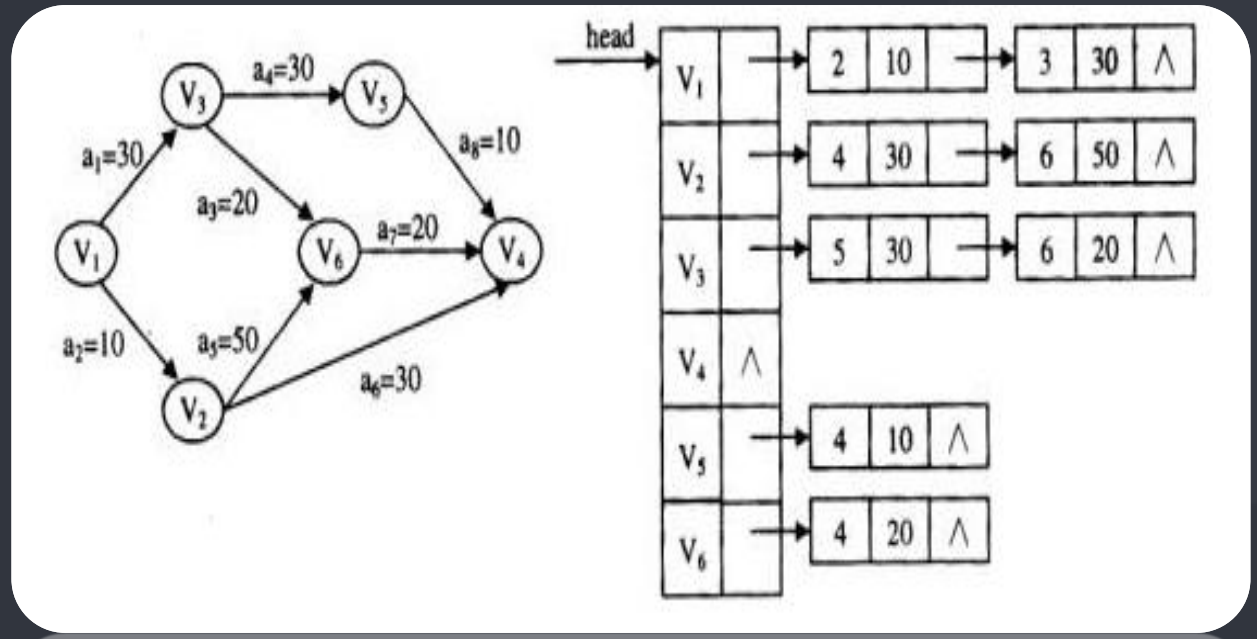
- **邻接矩阵**
  - map[i][j]表示i点到j点的边权
  - 无向图的邻接矩阵是对称的(一条无向边对应两条有向边)



0	1	0	1	1
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
1	1	0	1	0

图的存储

邻接表
伪邻接表（链式前向星）




```
struct ty
{
   int dian,l;
   struct node *next;
}*[60020];

void insert(int x,int y,int z)
{
     node *p=new  node;
     p->dian=y;
     p->l=z;
     p->next=edge[x];
     edge[x]=p;
}
```

```
struct ty
{
     int t, next;
};
void insertedge(int x, int y)
{
     edge[++m].t = y;
     edge[m].next = head[x];
     head[x] = m;
}
```

牛客竞赛
AC.NOWCODER.COM

图的DFS遍历

- 算法步骤：
- 1、从某个节点开始，每次任选一个与它相邻的未访问节点访问下去
- 2、直到当前节点的所有相邻节点都已经被访问过。
- 3、回溯到第一个未被访问过的节点

图的BFS遍历

- 1、用dis[]数组表示各点距离起点S的距离。dis[i]=-1表示i点还未被访问。用map[i][j]表示i点和j点之间是否有边。
- 2、将dis[s]初始化为0，将其它点的dis初始化为-1。将S点入队
- 3、while（队列非空）
- 从队首出队一个元素u
- 对于所有跟u有边相连的点v:
- if(dis[v]==-1)
- dis[v]=dis[u]+1;
- v入队

判断是否为欧拉图

- 如果图G中的一个路径包括每个边恰好一次，则该路径称为欧拉路径(Euler path)。
- 如果一个回路是欧拉路径，则称为欧拉回路(Euler circuit)。
- 具有欧拉回路的图称为欧拉图（简称E图）。具有欧拉路径但不具有欧拉回路的图称为半欧拉图。

- 无向图存在欧拉回路的充要条件
- 一个无向图存在欧拉回路，当且仅当该图所有顶点度数都为偶数,且该图是连通图。
- 有向图存在欧拉回路的充要条件
- 一个有向图存在欧拉回路，所有顶点的入度等于出度且该图是连通图。

拓扑排序

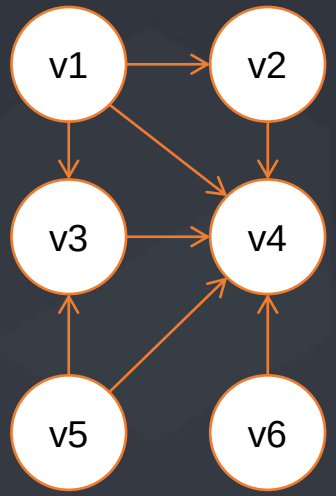
- 概念引入：
- 一个工程常被分为多个小的子工程，这些子工程被称为活动（Activity），在有向图中若以顶点表示活动，有向边表示活动之间的先后关系，这样的图简称为AOV网。在AOV网中为了更好地完成工程，必须满足活动之间先后关系，需要将各活动排一个先后次序即为拓扑排序

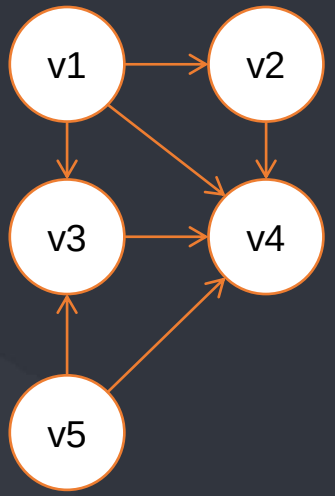
拓扑排序

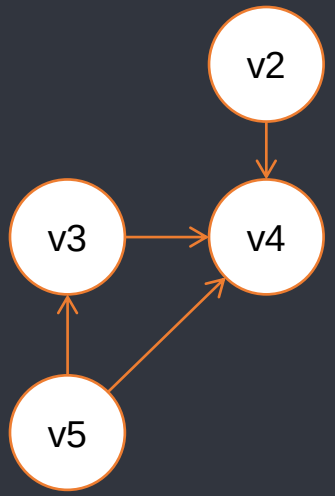
- 1.从有向图中选取一个没有前驱的顶点，并输出之；
- 2.从有向图中删去此顶点以及所有以它为尾的弧；
- 重复上述两步，直至图空，或者图不空但找不到无前驱的顶点为止。没有前驱 -- 入度为零，删除顶点及以它为尾的弧-- 弧头顶点的入度减1。

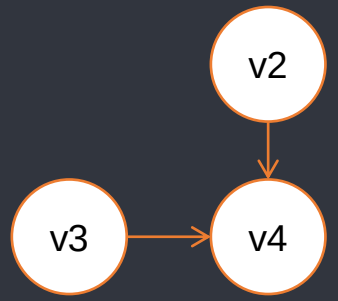
拓扑排序

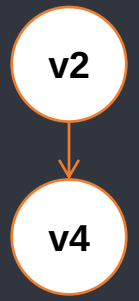
- 根据算法思想：
- 找到的点的顺序依次为v6，v1，v5，v3，v2，v4
- 可以看到，拓扑排序可能有多解
- （注意一边输出当前点，一边更新其他点的入度）









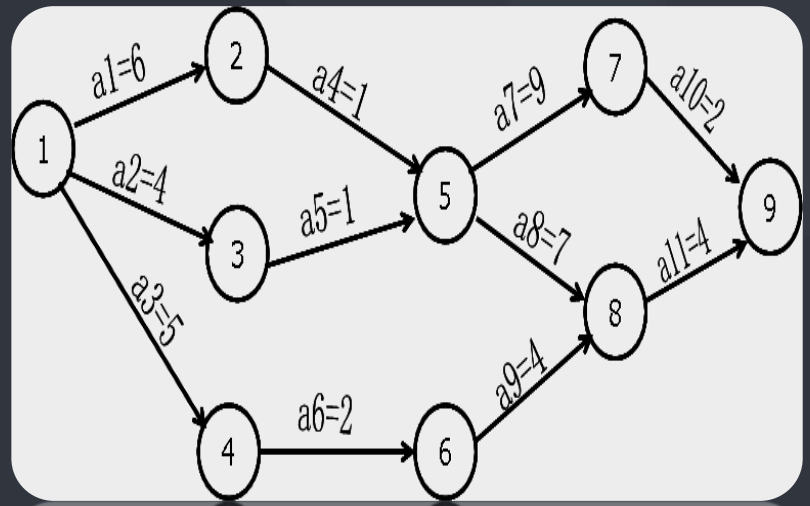


拓扑排序的使用

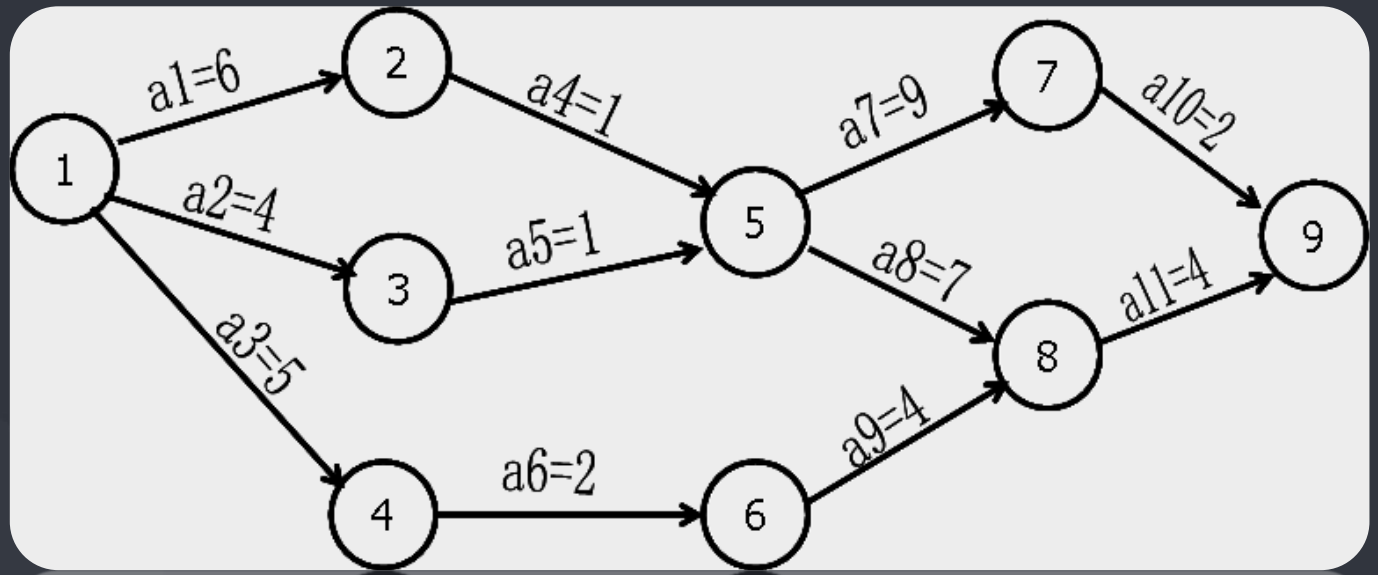
- 判断一个有向图中是否有环。无环的图所有点都能进行拓扑排序。

关键路径

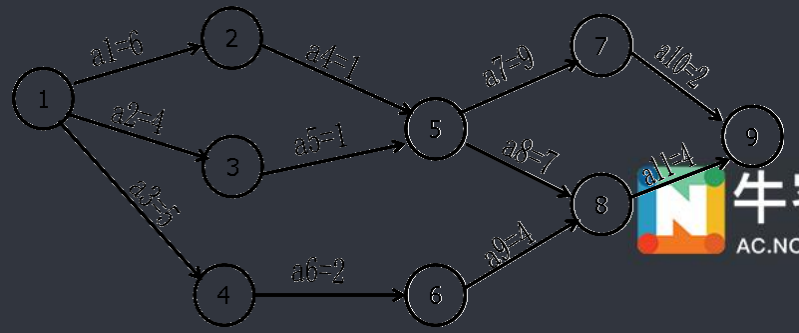
- AOE网(Activity On Edge network)，即边表示活动的网络，与AOV网相对应，它通常表示一个工程的计划或进度。
- AOE网是一个带权的有向无环图，图中的：
  边：表示活动(子工程)，
  边上的权：表示该活动的持续时间，即完成该活动所需要的时间；
- 顶点：表示事件，每个事件是活动之间的转接点，即表示它的所有入边活动到此完成，所有出边活动从此开始。其中有两个特殊的顶点(事件)，一个称做源点，它表示整个工程的开始，亦即最早活动的起点，显然它只有出边，没有入边；另一个称做汇点，它表示整个工程的结束，亦即最后活动的终点，显然它只有人边，没有出边。除这两个顶点外，其余顶点都既有人边，也有出边，是入边活动和出边活动的转接点。



- AOE网中有些活动可以并行进行，所以完成整个工程的最短时间是从源点到汇点的最长路径长度，路径长度为路径上各边的权值之和。把从源点到汇点的最长路径长度称为关键路径。
- 对于一个AOE网，待研究的问题是：(1)整个工程至少需要多长时间完成？(2)哪些活动是影响工程进度的关键？



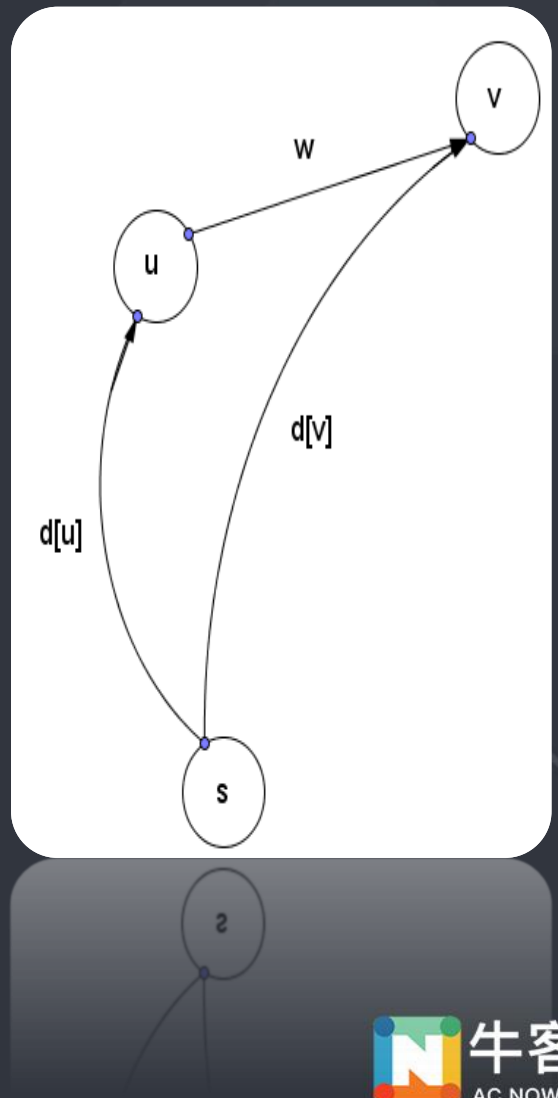
- 假设开始点是v1，从v1到vi的最长路径长度叫做事件vi的最早发生时间。这个时间决定了所有以vi为尾的弧所表示的活动的最早开始时间。我们用e(i)表示活动ai的最早开始时间。还可以定义一个活动的最迟开始时间l(i)，这是在不推迟整个工程的前提下，活动ai最迟必须开始进行的时间。两者之差l(i)-e(i)意味着完成活动ai的时间余量。我们把l(i)=e(i) 的活动叫做关键活动。
- 关键路径上的所有活动都是关键活动，因此提前完成非关键活动并不能加快工程的进度。



最短路

单源点最短路

- 从一个点出发，到达其他顶点的最短路径的长度。
- 基本操作：松弛
- d[u]+map[u, v]< d[v]这样的边(u,v)称为紧的(tense),可以对它进行松弛(relax):
- d[v] = d[u]+w, pred[v] = u
- 最开始给每一个点一个很大的d值从d[s]=0开始，不断的对可以松弛的点进行松弛，不能松弛的时候就已经求出了最短路了

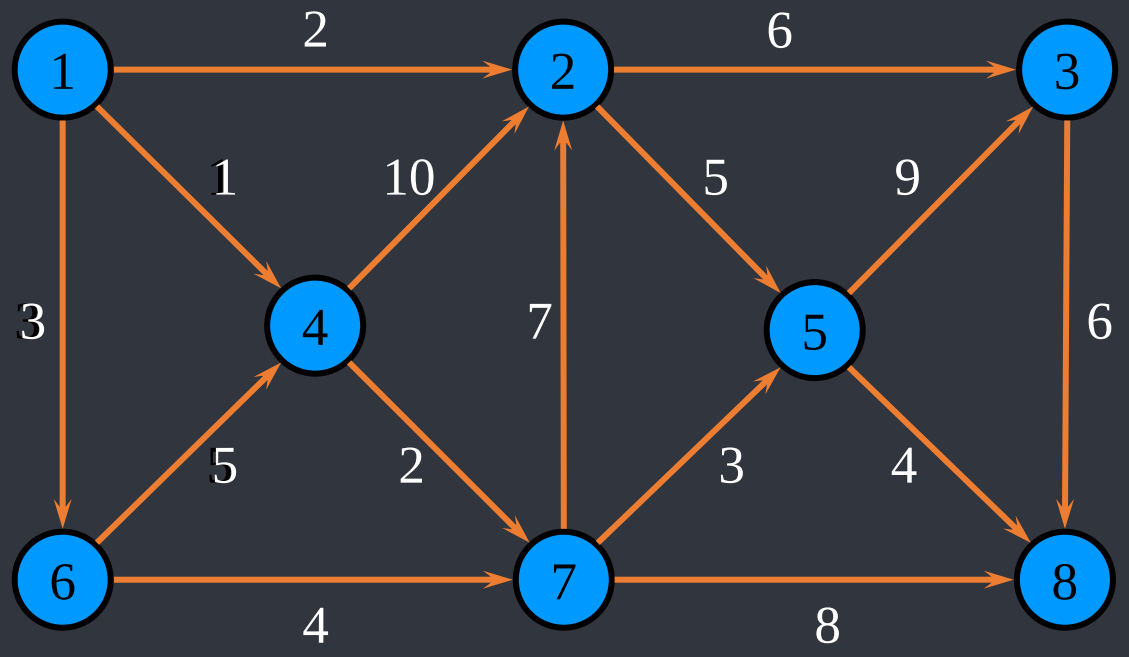


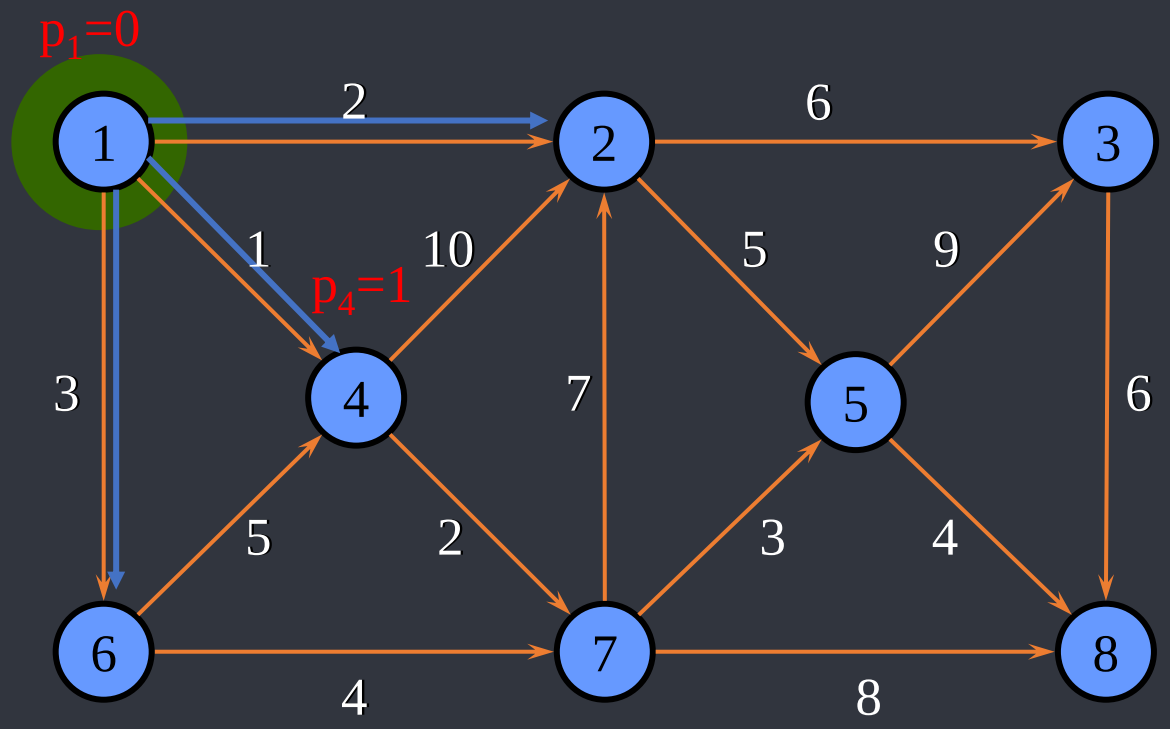
Dijkstra算法

- Dijkstra(迪杰斯特拉)算法是典型的单源最短路径算法，用于计算一个节点到其他所有节点的最短路径。主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止
- 注意该算法要求图中不存在负权边。
- 可以证明，具有最小的d[i]（临时最短路）值的（还没加入最短路）点在此以后无法松弛
- 所以每次找最近的点进行松弛操作

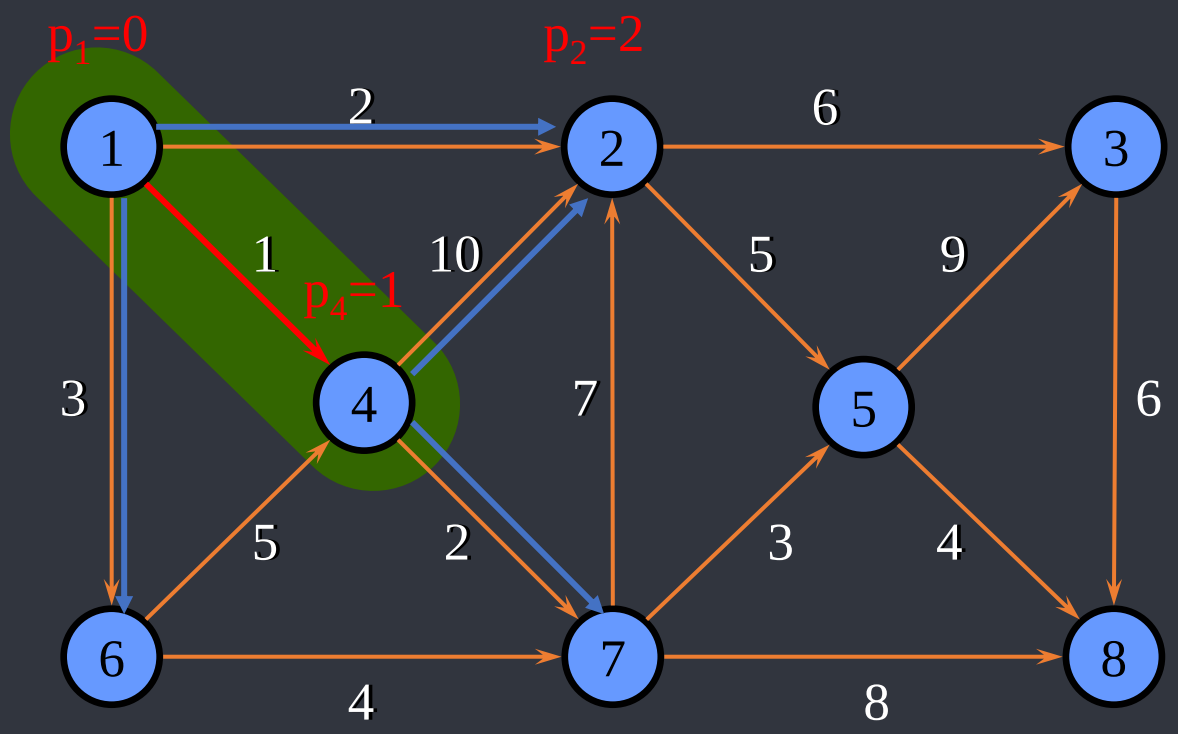
dijstra算法

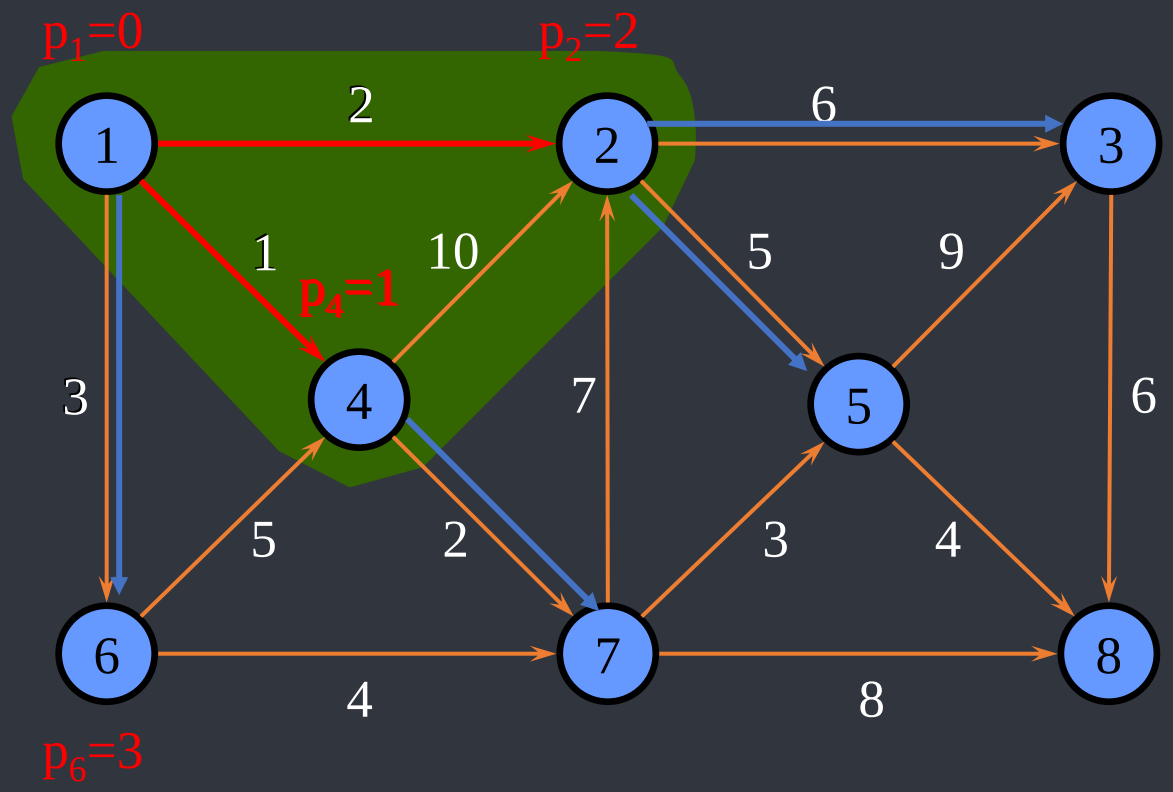
- 1、在开始之前，认为所有的点都没有进行过计算，dis[]全部赋值为极大值(dis[]表示各点当前到源点的最短距离)
- 2、源点的dis值明显为0
- 3、计算与s相邻的所有点的dis值 —— dis[v] = map[s][v]
- 4、还没算出最短路的点中dis[]最小的一个点u， 其最短路就是当前的dis[u]
- 5、对于与u相连的所有点v，若dis[u]+map[u][v] 比当前的dis[v]小, 更新dis[v]
- 6、重复4,5直到源点到所有点的最短路都已求出

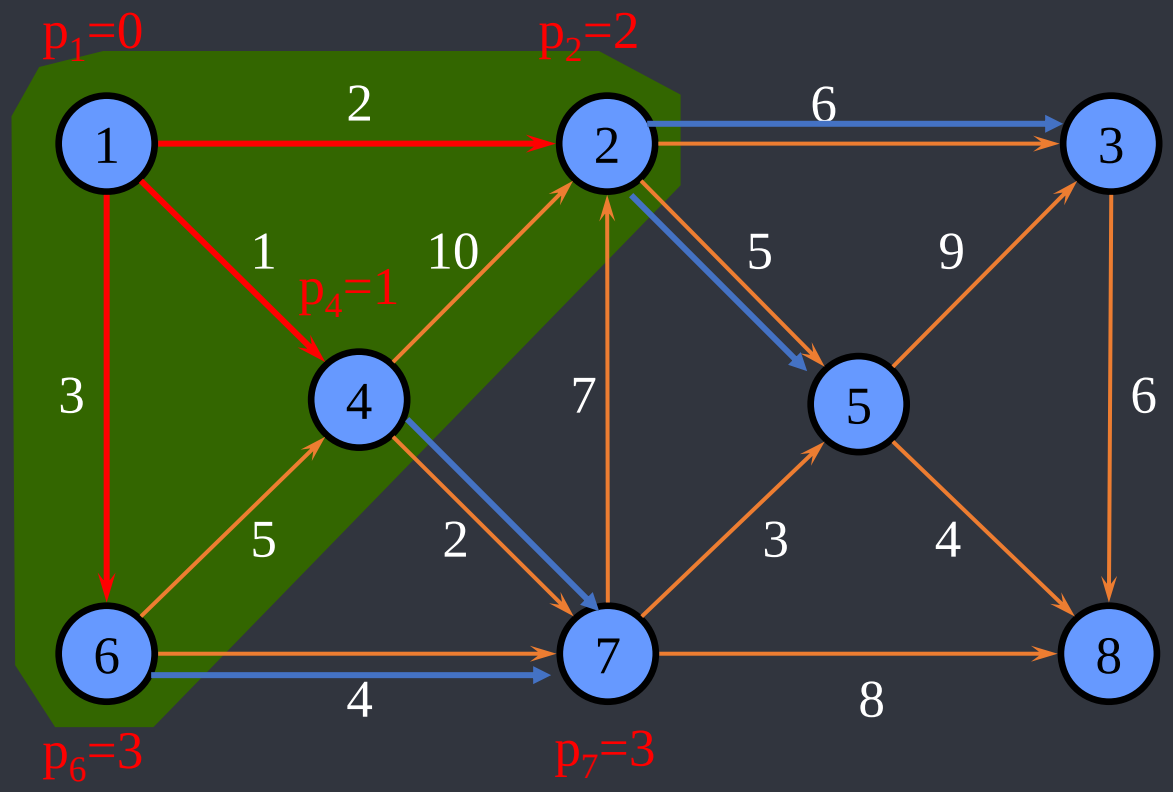


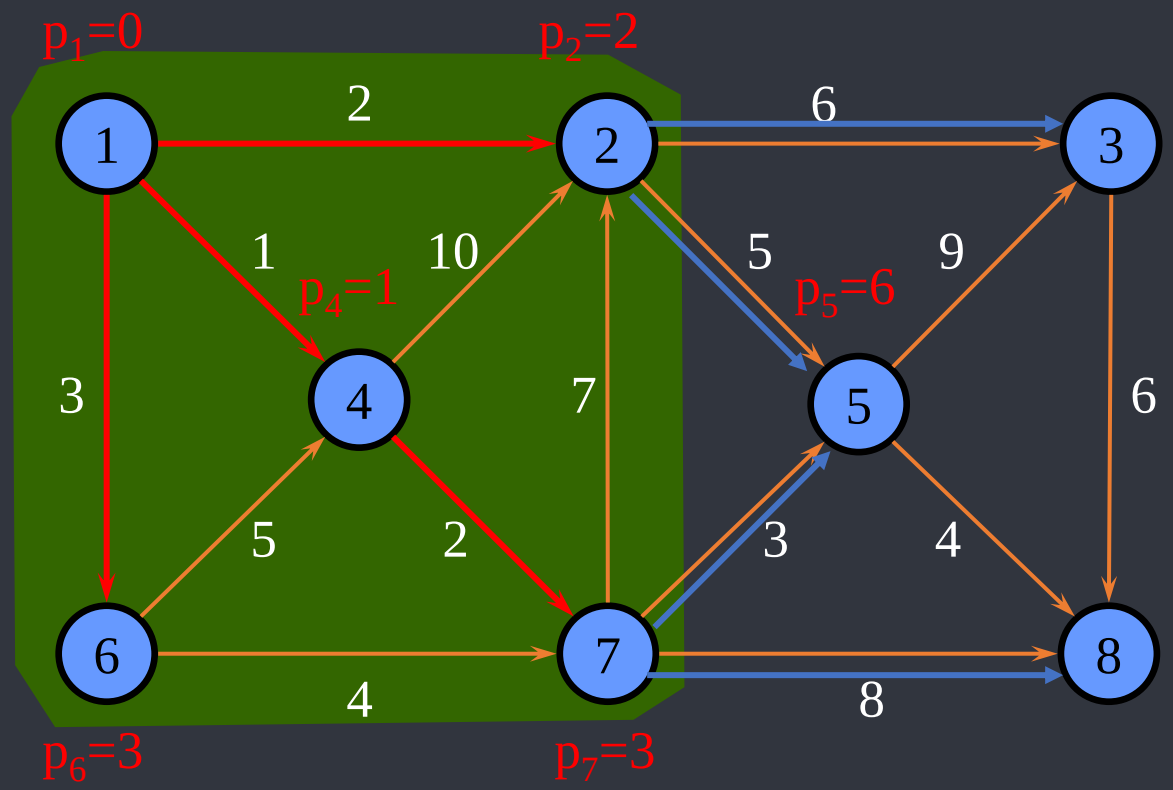


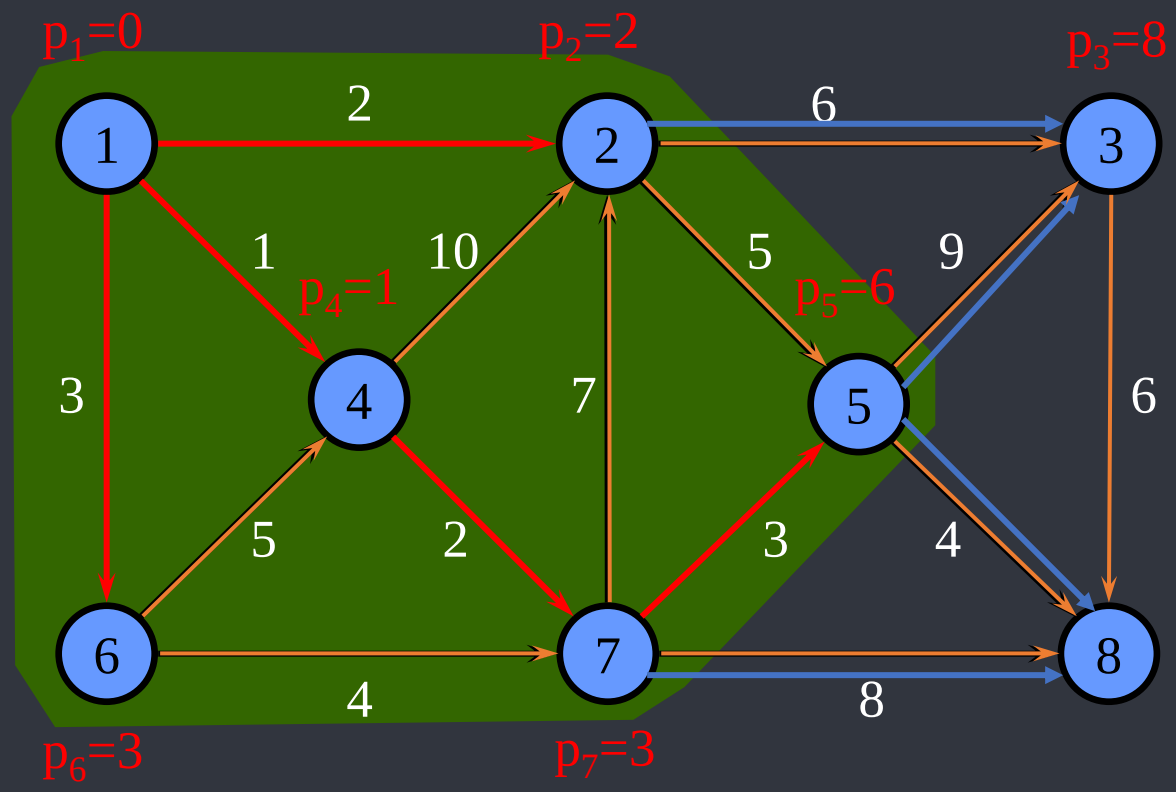
X={1,4}

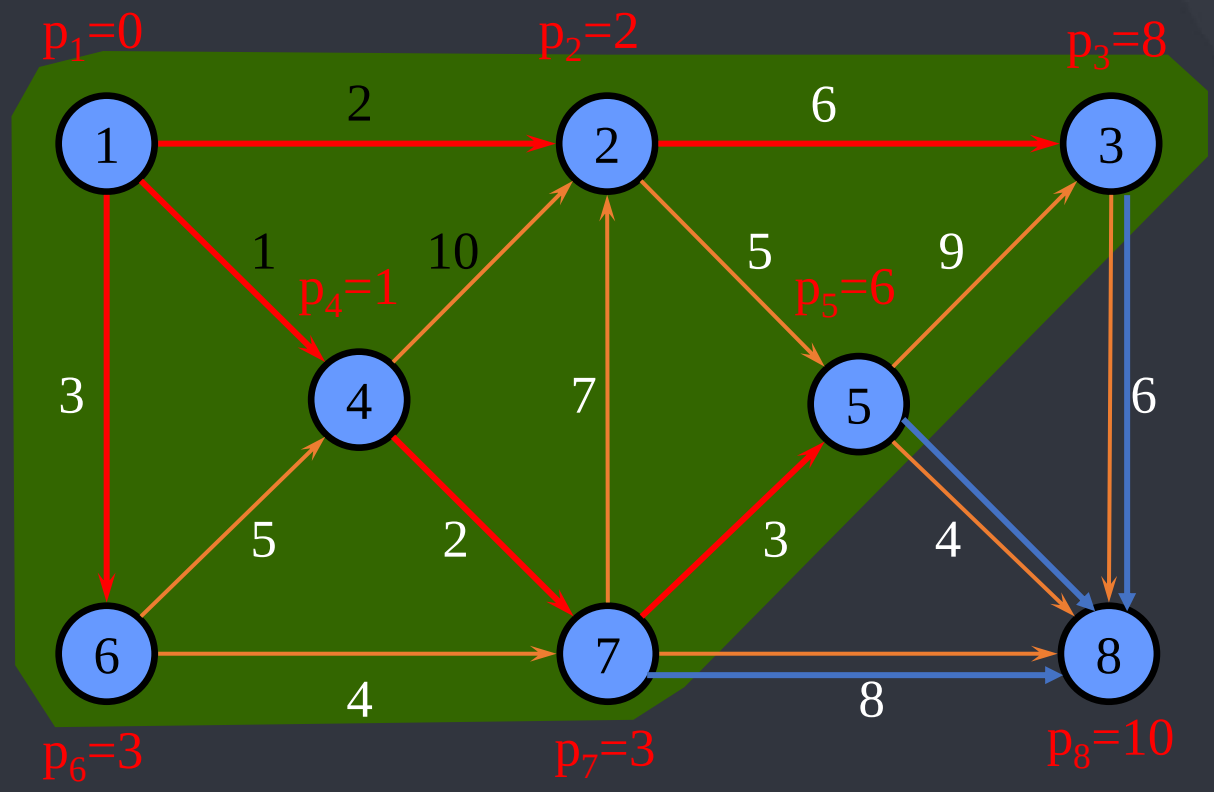


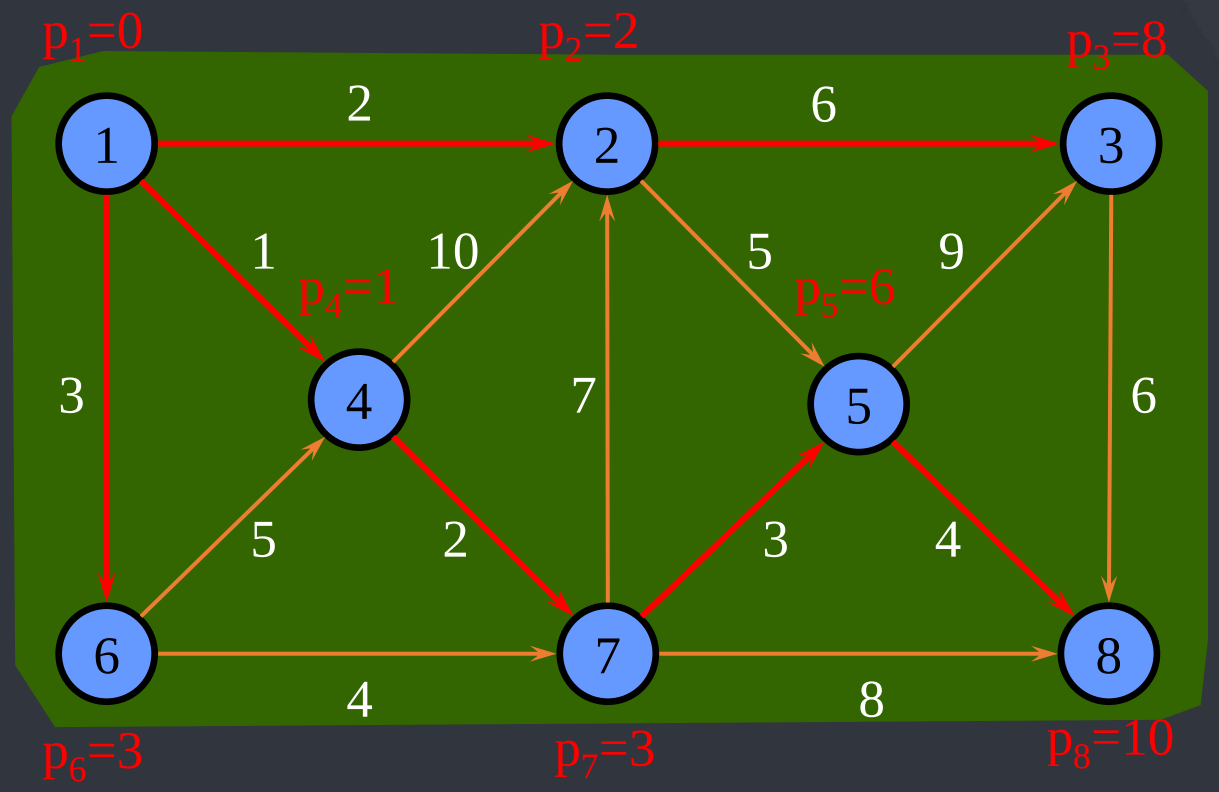






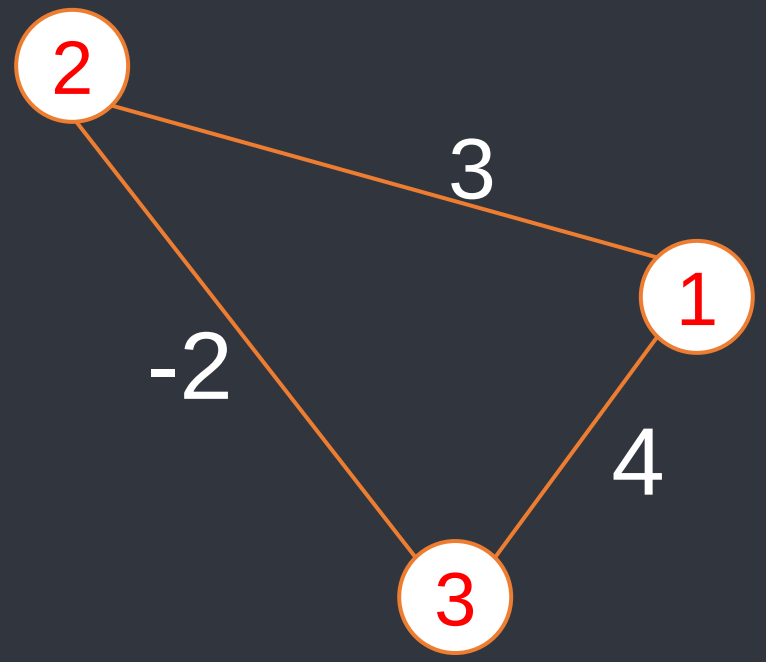






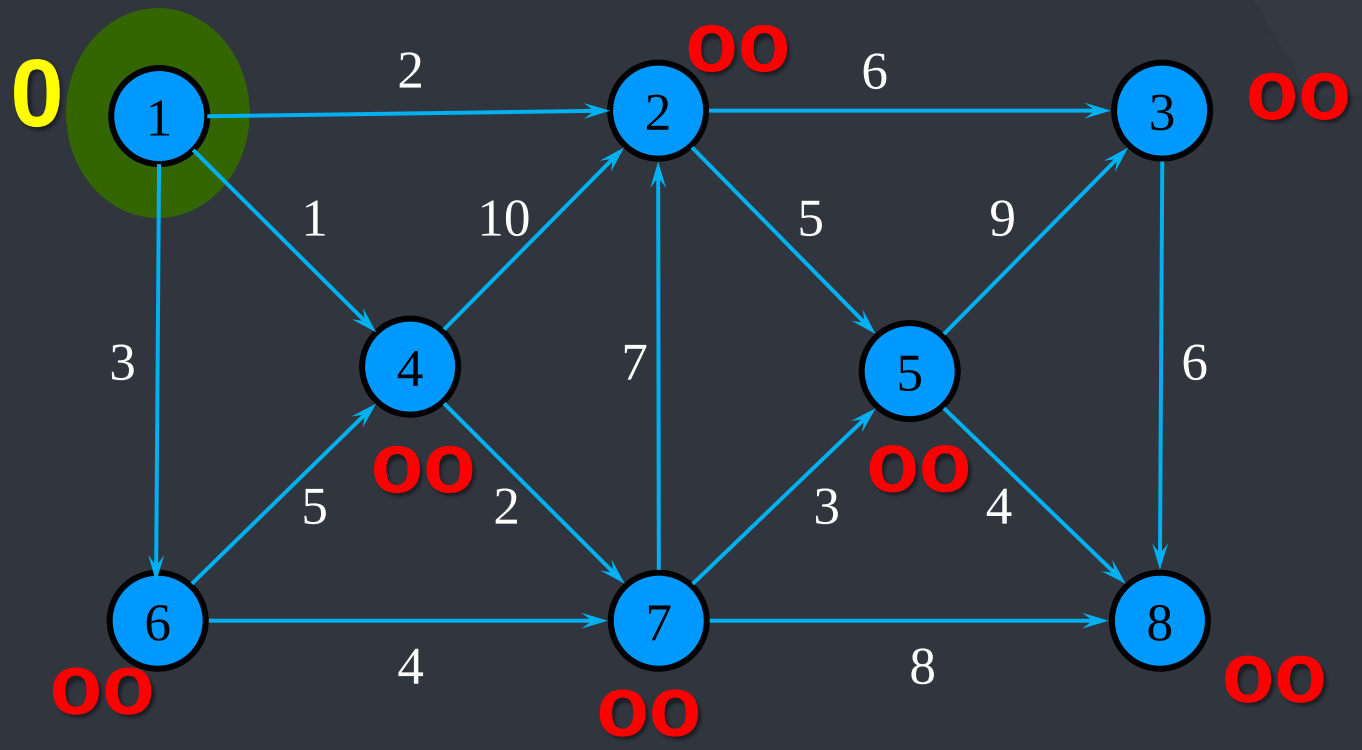
Dijkstra

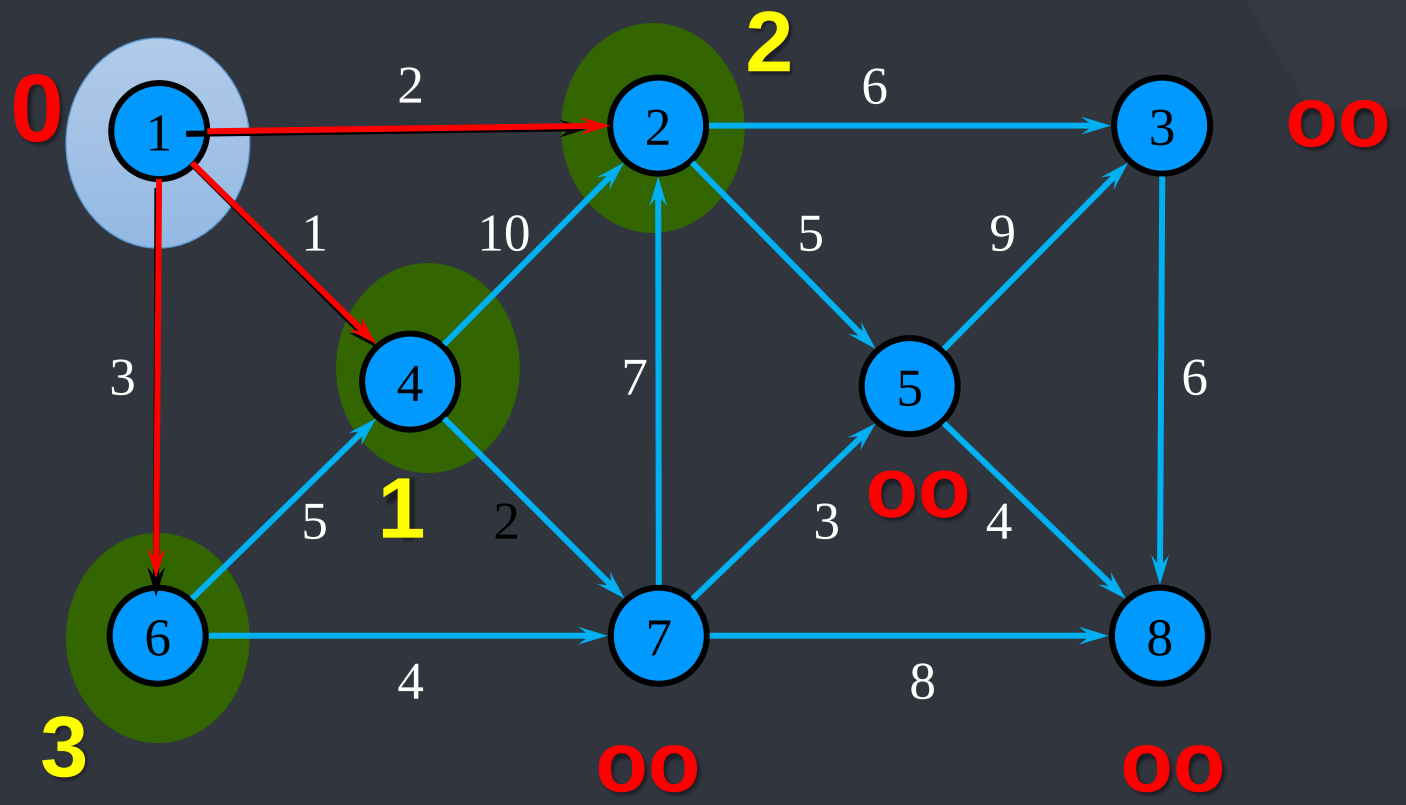
- Dijkstra算法也适用于无向图。但不适用于有负权边的图。
- d[1,2] = 2
- 但用Dijkstra算法求得 d[1,2] = 3

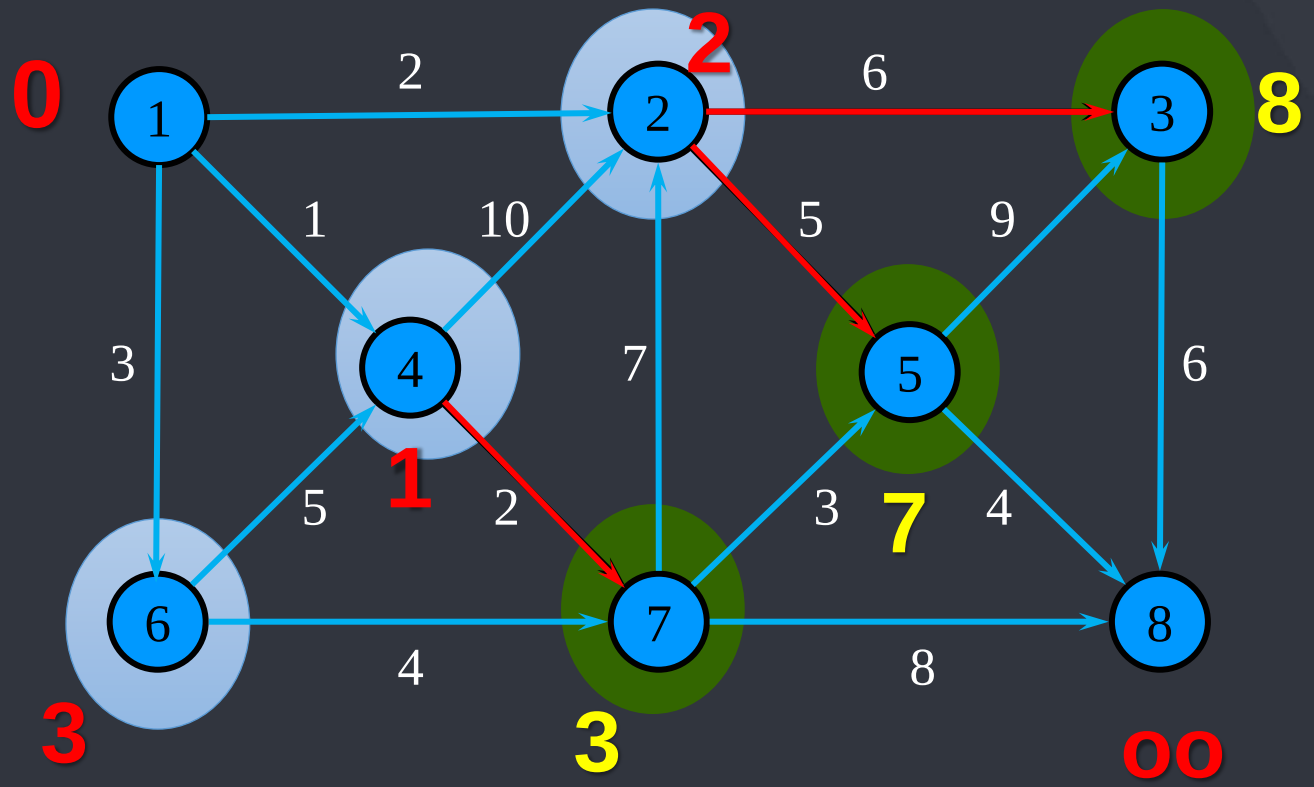


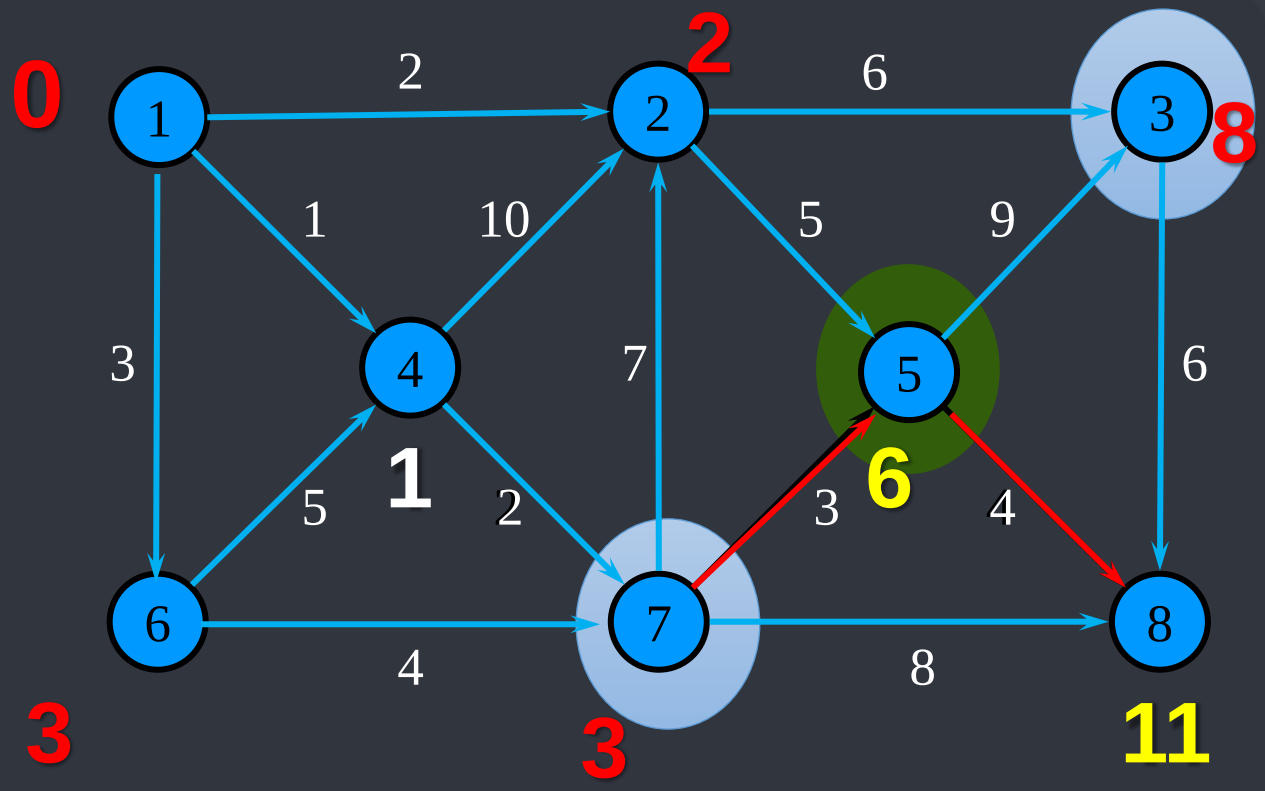
Bellman-Ford算法

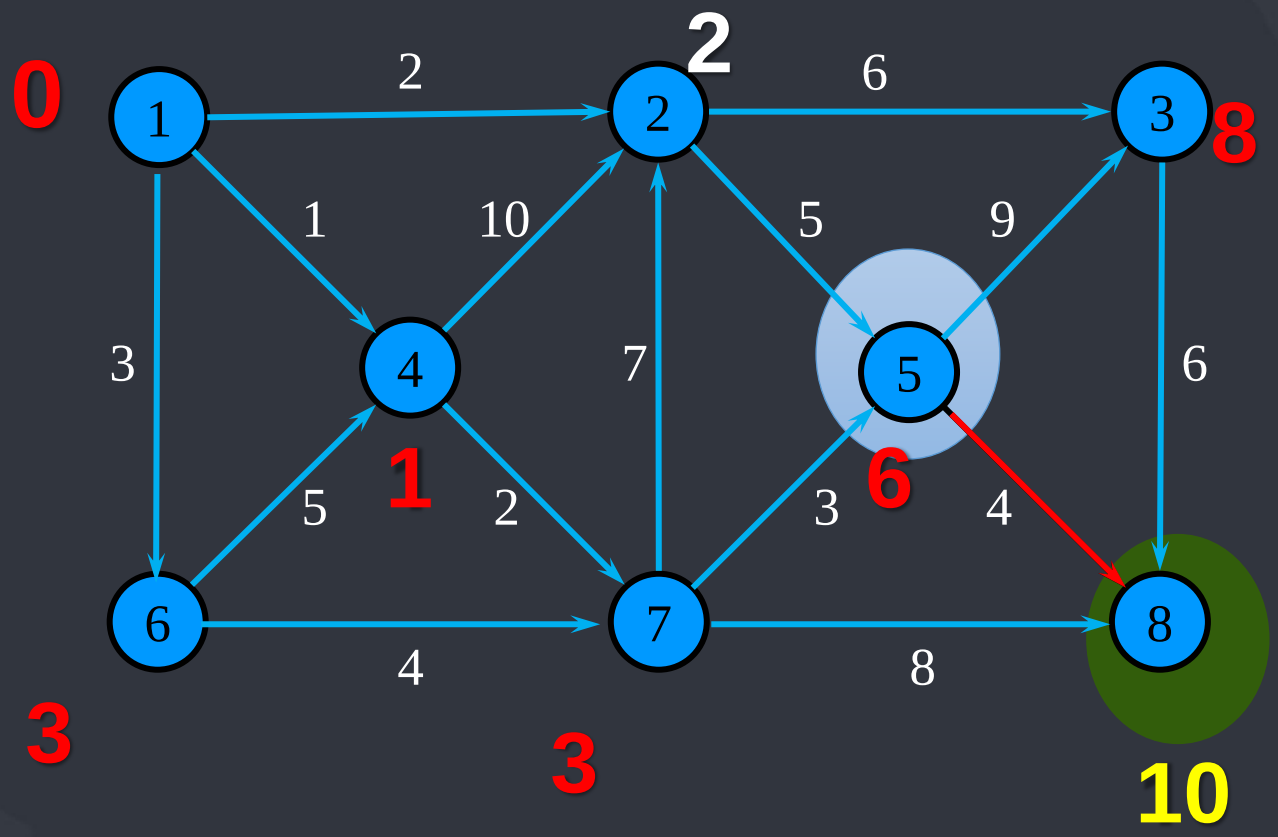
- Bellman-Ford算法：为了能够求解含负权边的带权有向图的单源最短路径问题，Bellman(贝尔曼)和Ford(福特)提出了从源点逐次绕过其他顶点，以缩短到达终点的最短路径长度的方法。
- 枚举所有的点，能松弛就进行松弛操作，直到所有点都不能松弛了





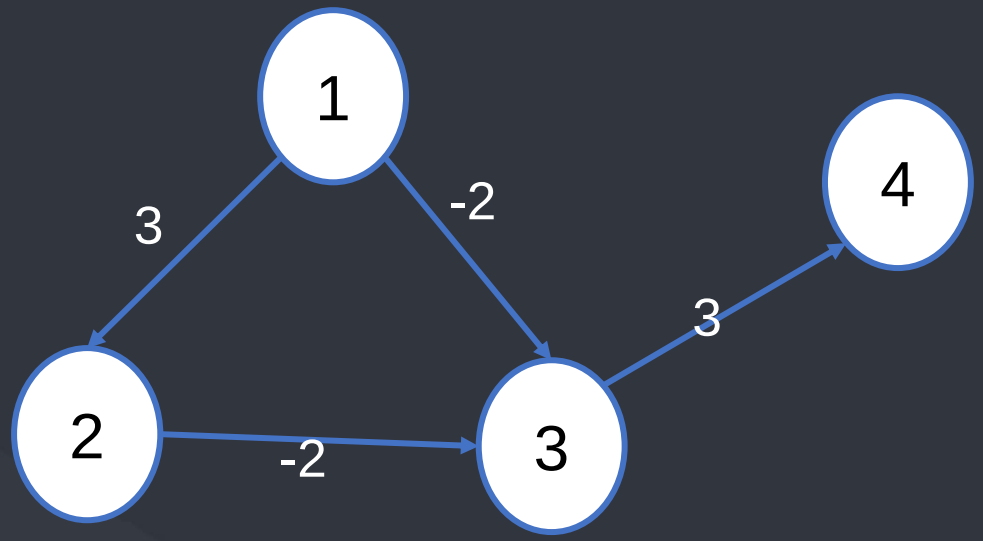






Bellman-Ford算法

- Bellman-Ford算法的限制条件：要求图中不能包含权值总和为负值回路(负权值回路)，如下图所示。



```
14    While (b)
15    {
16        b =false;
17        for (i = 1; i <= n; i++)
18        for (j = 1; j <= n; j++)
19            if (d[j] + map[j][i] < d[i])
20            {
21                d[i]  =  d[j] + map[j][i];
22                b = true;
23            }
24    }
```

Bellman-ford的队列优化——SPFA

- 每一次松弛的时候bellman-ford都要枚举所有的点，而其实有很多点都是不需要枚举的，所以又很多的无效枚举，于是效率显得略低
- 怎么改进呢?
- 其实每次松弛的时候只需要枚举与上次被松弛的点相连的点就可以了
- 于是有了Shortest Path Faster Algorithm

SPFA算法的实现

- 设Dist代表S到I点的当前最短距离，Fa代表S到I的当前最短路径中I点之前的一个点的编号。开始时Dist全部为+∞，只有Dist[S]=0，Fa全部为0。
- 维护一个队列，里面存放所有需要进行迭代的点。初始时队列中只有一个点S。用一个布尔数组记录每个点是否处在队列中。

SPFA算法的实现

- 每次迭代，取出队头的点v，依次枚举从v出发的边v->u，设边的长度为len，判断Dist[v]+len是否小于Dist[u]，若小于则改进Dist[u]，将Fa[u]记为v，并且由于S到u的最短距离变小了，有可能u可以改进其它的点，所以若u不在队列中，就将它放入队尾。这样一直迭代下去直到队列变空，也就是S到所有节点的最短距离都确定下来，结束算法。若一个点入队次数超过n，则有负权环。

SPFA算法的实现

- SPFA 在形式上和广度优先搜索非常类似，不同的是广度优先搜索中一个点出了队列就不可能重新进入队列，但是SPFA中一个点可能在出队列之后再次被放入队列，也就是一个点改进过其它的点之后，过了一段时间可能本身被改进，于是再次用来改进其它的点，这样反复迭代下去。

SPFA算法的实现

- 在Bellman-Ford算法中，要是某个点的最短路径估计值更新了，那么我们必须对所有边的终点再做一次松弛操作；在SPFA算法中，某个点的最短路径估计值更新，只有以该点为起点的边指向的终点需要再做一次松弛操作。在极端情况下，后者的效率将是前者的n倍。
- 在平均情况下，SPFA算法的期望时间复杂度为O(KM)。M为边数，k是每个点平均入队次数。

Floyd算法

- 假设求从顶点vi到vj的最短路径。如果从vi到vj有弧，则从vi到vj存在一条长度为cost[i,j]的路径，该路径不一定是最短路径，尚需进行n次试探。
- 考虑路径（ vi， v1 ， vj）是否存在（即判别弧（ vi， v1 ）和（ v1 ， vj ）是否存在）。如果存在，则比较cost[i,j]和（ vi， v1 ， vj）的路径长度，取长度较短者为从vi到vj的中间顶点的序号不大于1的最短路径，记为新的cost[i,j] 。
- 假如在路径上再增加一个顶点v2 ，如果（ vi，…， v2 ）和（ v2 ，…，vj ）分别是当前找到的中间顶点的序号不大于2的最短路径，那么（ vi，…， v2 ，… ， vj ）就有可能是从vi到 vj的中间顶点的序号不大于2的最短路径。将它和已经得到的从vi到 vj的中间顶点的序号不大于1的最短路径相比较，从中选出中间顶点的序号不大于2的最短路径之后，再增加一个顶点v3 ，继续进行试探。依次类推。

Floyd算法

- 在一般情况下，若（vi，...，vk）和（vk，...，vj）分别是从vi到vk和从vk到vj的中间顶点的序号不大于k-1的最短路径，则将（vi，...，vk，...，vj）和已经得到的从vi到vj且中间顶点的序号不大于k-1的最短路径相比较，其长度较短者便是从vi到vj的中间顶点的序号不大于k的最短路径。这样，在经过n次比较后，最后求得的必是从vi到vj的最短路径。按此方法，可以同时求得各对顶点间的最短路径。

Floyd求最短路径

- 动态规划思想
- F[i,j]表示i到j的经过小于k的点所能得到的临时最短路
- 枚举中转点k
- if (f[i][k] + f[k][j] <= f[i][j]) f[i][j] = f[i][k] + f[k][j];

```
14    for (int k = 1; k <= n; k++)
15    for (int i = 1; i <= n; i++)
16    for (int j = 1; j <= n; j++)
17        if ((i != j)&&(j != k)&&(k != i))
18        {
19            if (f[i][k] + f[k][j] <= f[i][j])
20                f[i][j] = f[i][k] + f[k][j];
21        }
22
```

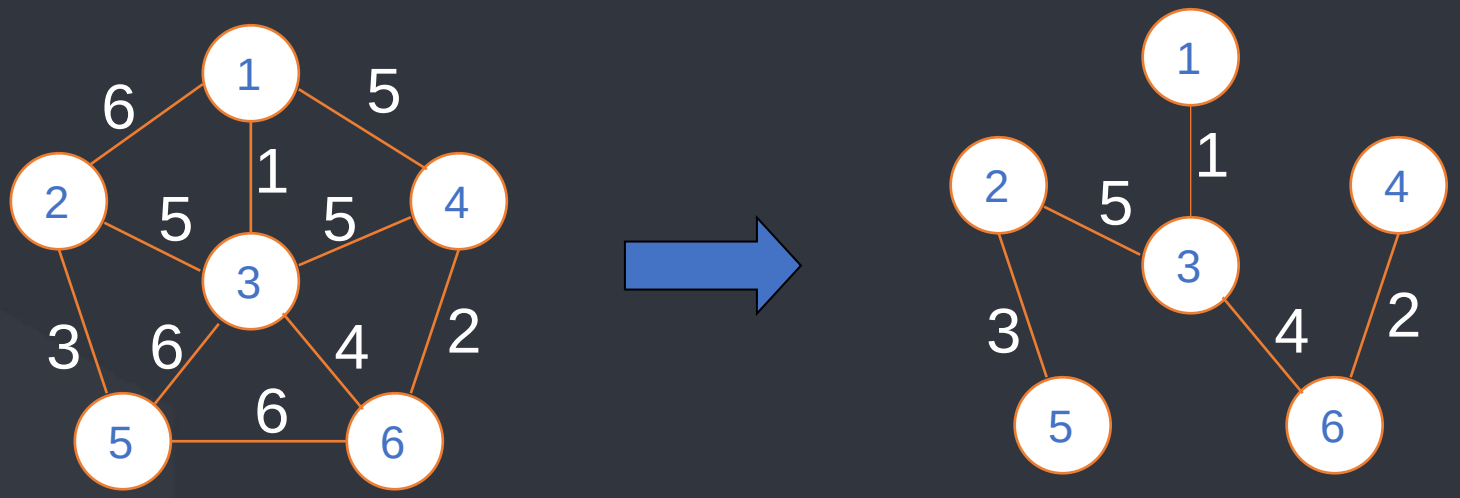
Dijkstral、SPFA、floyd比较

- dijkstral算法（复杂度O(n^2))——不能有负权边
- spfa算法　　(复杂度O(km))——不能有负权环
- Floyd（复杂度O(n^3))——可以求多源点最短路

最小生成树

最小生成树

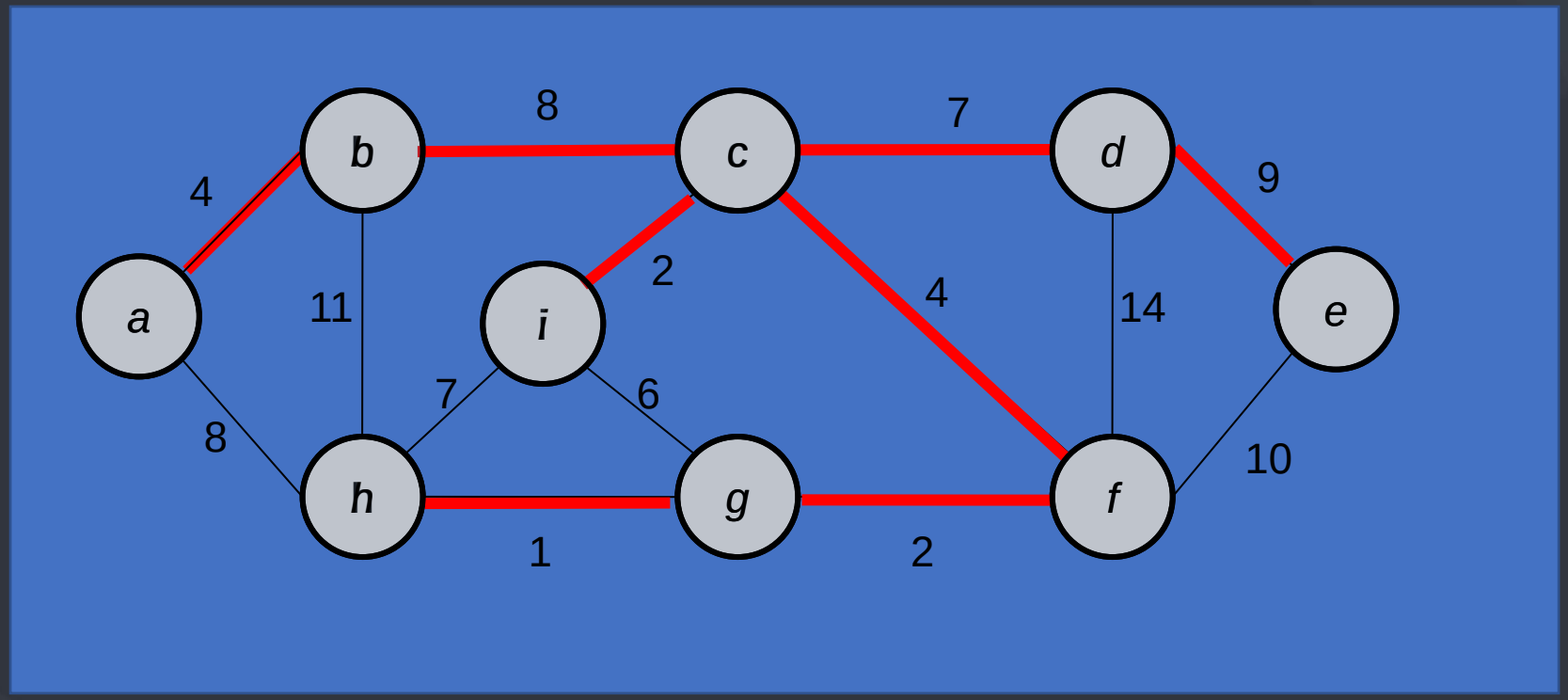
- G=（V，E），若G的一个生成子图是一棵树，则称之为G的一棵生成树（记为T）
- 最小生成树： 无向图G的所有生成树中，树枝的权值总和最小的称为G的最小生成树。



Prim算法

- 贪心准则
  - 加入后仍形成树，且耗费最小
- 算法过程
  - 从单一顶点的树T开始
  - 不断加入耗费最小的边(u, v)，使T∪{(u, v)}仍为树　——u、v中有一个已经在T中，另一个不在T中

Prim 算法

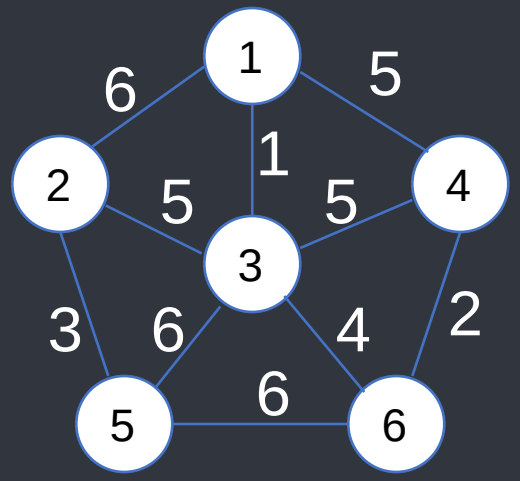


```
15  void prim()
16  {
17      long result=0;
18      ty t1, t2;
19      memset(dist, 127, sizeof(dist));
20      dist[1] = 0;
21      for (long i = head[1]; i != 0; i = edge[i].next)
22      {
23          if (edge[i].w < dist[edge[i].t])
24          {
25              dist[edge[i].t] = edge[i].w;
26              q.push(edge[i]);
27          }
28      }
```

```
29      while( !q.empty() )
30      {
31          t1 = q.top();
32          q.pop();
33          if (dist[t1.t] == 0) continue;
34          result += t1.w;
35          dist[t1.t] = 0;
36          for (long j = head[t1.t]; j != 0; j = edge[j].next)
37          {
38              long u = edge[j].t;
39              if ((dist[u] != 0) && (dist[u] > edge[j].w))
40              {
41                  dist[u] = edge[j].w;
42                  t2.t = u;
43                  t2.w = dist[u];
44                  q.push(t2);
45              }
46          }
47      }
48      cout << result << endl;
49  }
```

Kruskal算法

- kruskal 算法思想：贪心选取最短的边来组成一棵最小的生成树。
- 具体做法：先将所有的边做排序，然后利用并查集作判断来优先选择较小的边，直到建成一棵生成树。



- 实现方式——并查集！

```cpp
int kruskal()
{
    int cnt = 0;
    int int sum = 0;
    for (int i = 0; i < m; i++)
    {
        int fx = findfa(edge[i].x);
        int fy = findfa(edge[i].y);
        if (fx != fy)
        {
            merge(fx, fy);
            cnt++;
            sum += edge[i].l;
            if (cnt >= n - 1) break;
        }
    }
    cout << sum;
}
```

```cpp
int findfa(int x)
{
    return fa[x] == x ? x : (fa[x] = findfa(fa[x]));
}

int merge(int x, int y)
{
    fa[findfa(x)] = findfa(y);
}
```

Prim 和 Kruskal的比较

- Prim和Kruskal的贪心策略是一样的，都是选耗费最小的边：
- 对于Prim，其选取的边(u,v)必有一个顶点已经被覆盖，另一个顶点未被覆盖。
- 而对于Kruskal,其选取的边(u,v)任意，只要这个边的加入不能使被覆盖的顶点构成回路。

- prim 算法　　(复杂度O(n^2))——适用于稠密图
- kruskal算法　　(复杂度O(m*lg(m))——适用于稀疏图

建图~

例1：

- 一张N个点M条边的有向图中，求从指定k个点任意一个出发到点S的最短路

例2： NC208246胖胖的牛牛

- N*N（1≤N≤100）方格中，‘x’表示不能行走的格子，‘.’表示可以行走的格子。卡门很胖，故而不好转弯。现在要从A 到走到B 点，请问最少要转90度弯几次?

例3：

- 有n个点，分布在1~k层，一些点之间有边（n条），相邻层的任意两个点之间都有一条传送带，传动带将第i层的点传送到第i+1层的点花费的时间是Ti，求点1到点n的最小时间

例4：NC124167 UVALive7250　Meeting

- 给n个点，m个集合，每个集合里有si 个点，相同集合内部点的距离为fi，A在1号点，B在n号点，A B要见面，他们只能在节点见面不可以在路上见面，AB走单位距离花费1分钟，问最少花费几分钟可以见面。
- N<=10^5, M<=10^5
- Sigma(Si) <=10^6

例5：NC107872 poj 3275 Ranking the Cows

- 有N个数字，已经比较了M对(x,y),其中x>y, 问至少再需要比较多少对数字，就能把N个数按大小有序的排列起来
- N ≤ 1000

```
for(int k = 1;k <= n;k++)
        for(int i = 1;i <= n;i++)
            for(int j = 1;j <= n;j++)
                mp[i][j] = mp[i][j] || (mp[i][k] && mp[k][j]);
```

```
for (int k= 1; k <= n; k++)
    for (int j = 1; j <= n; j++)
        if (g[j][k]) g[j] |= g[k];
        //j可以到k，则k可以到的点j都可以到  或一下就可以了
```

例6：NC20568 [SCOI2012]滑雪与时间胶囊

- a180285非常喜欢滑雪。他来到一座雪山，这里分布着M条供滑行的轨道和N个轨道之间的交点（同时也是景点），而且每个景点都有一编号i（$1 \le i \le N$）和一高度H_i。a180285 能从景点i 滑到景点j 当且仅当存在一条i 和j 之间的边，且i 的高度不小于j。
- 与其他滑雪爱好者不同，a180285喜欢用最短的滑行路径去访问尽量多的景点。如果仅仅访问一条路径上的景点，他会觉得数量太少。于是a180285拿出了他随身携带的时间胶囊。
- 这是一种很神奇的药物，吃下之后可以立即回到上个经过的景点（不用移动也不被认为是 a180285 滑行的距离）。
- 请注意，这种神奇的药物是可以连续食用的，即能够回到较长时间 之前到过的景点（比如上上个经过的景点和上上上个经过的景点）。
- 现在，a180285站在1号景点望着山下的目标，心潮澎湃。他十分想知道在不考虑时间 胶囊消耗的情况下，以最短滑行距离滑到尽量多的景点的方案（即满足经过景点数最大的前 提下使得滑行总距离最小）。你能帮他求出最短距离和景点数吗?

作业

- https://ac.nowcoder.com/acm/problem/collection/1239