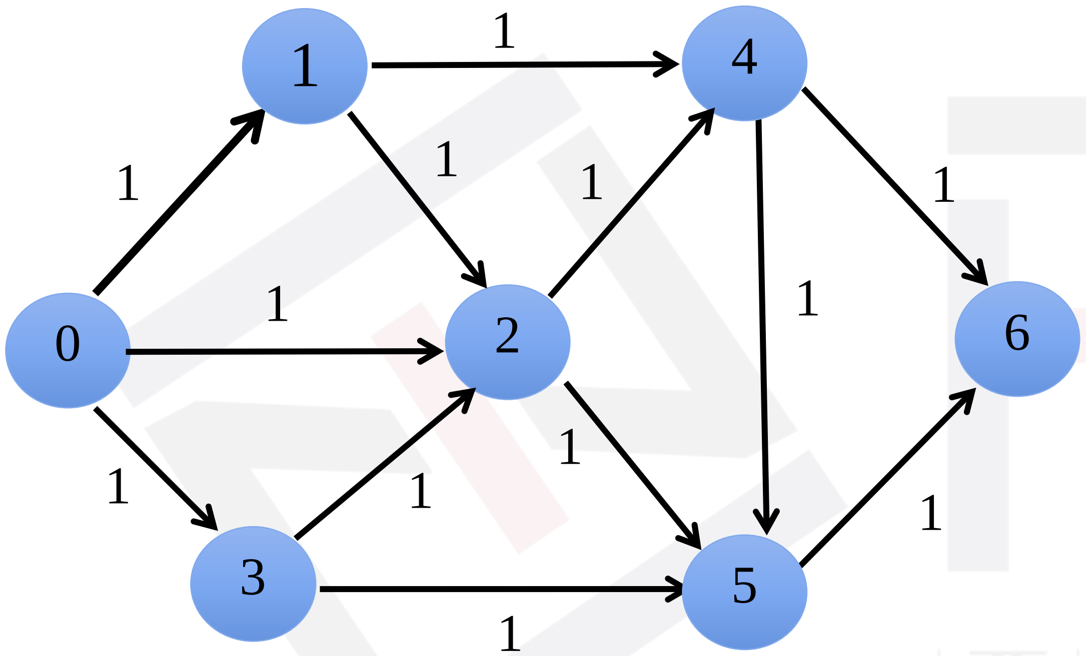
图论

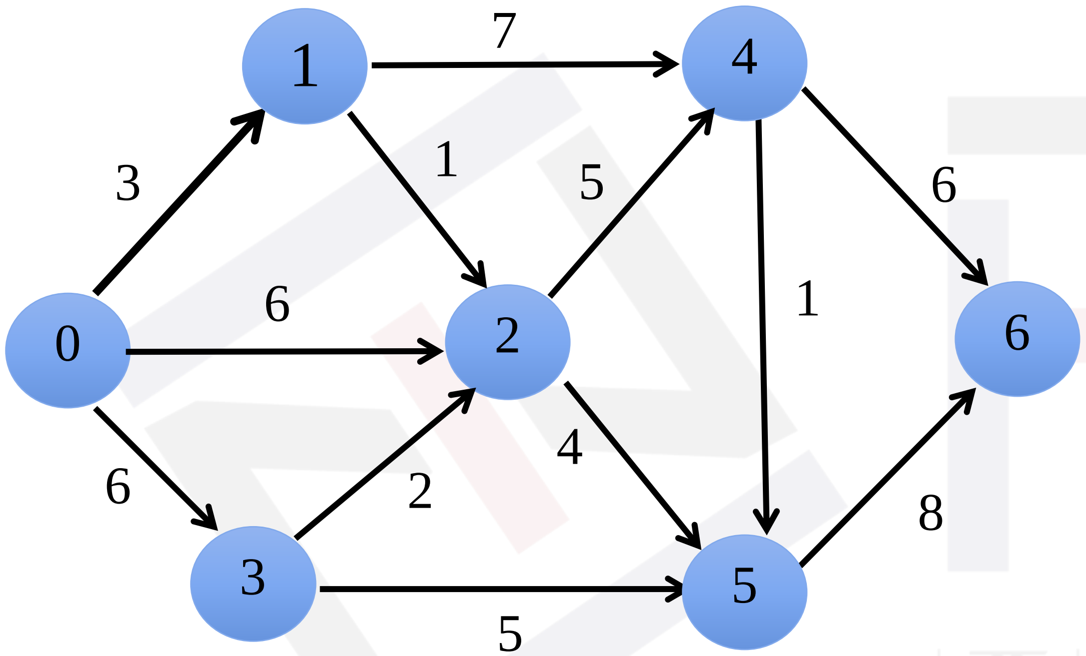
最短路问题



从节点0到节点6有很多条路径可以到达
但是哪一种方法**权值和**最小？即最短路径

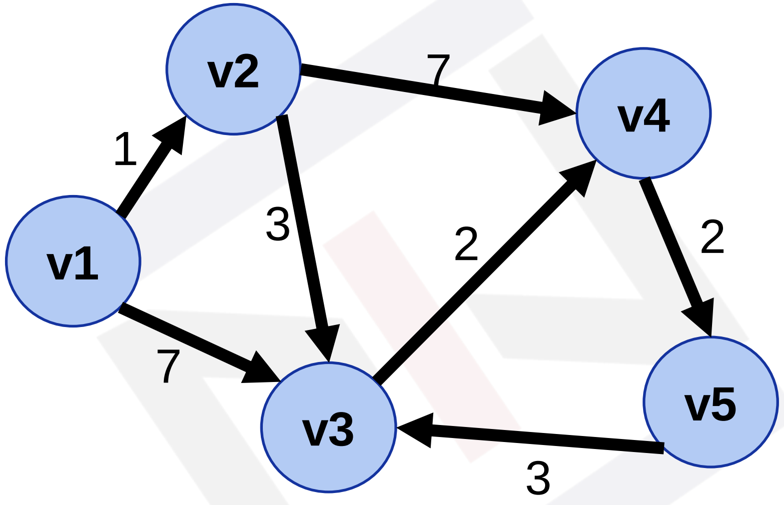
如何求?
BFS

相关概念



从节点0到节点6有很多条路径可以到达权值不为1，怎么办？

Floyed算法



Map[i][j]:i到j的距离

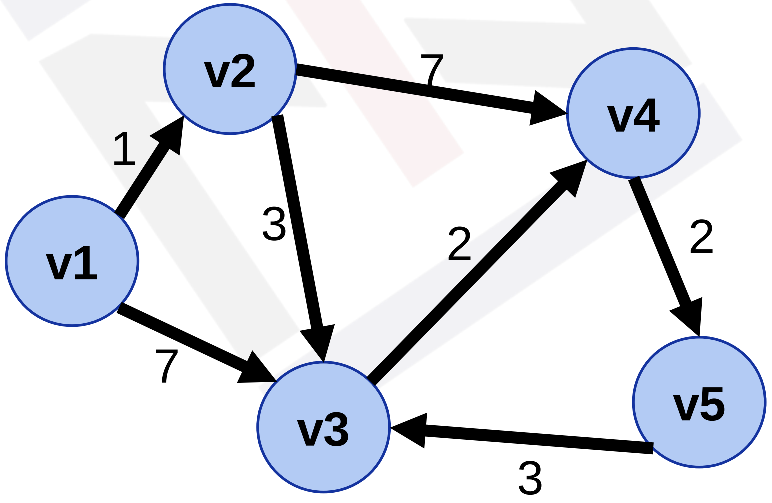
V3到v4的最短路就是Map[3][4];

如图中v1到v3的最短路，要经过v2

从i到j的最短路只有两种情况

1、最短路就是Map[i][j]

2、经过某些节点k，使得i到j最短路最小



我们可以**枚举**中间经过的节点来求最短路

Floyd算法

d[k][i][j] ：

表示路径除了i和j，中间只允许经过**1到k节点**的情况下，i到j的最小距离。

转移方程是:

有两种情况：加入k点之后

1、最短路经过k点： d[k][i][j]=d[k-1][i][k]+d[k-1][k][j]

2、最短路不经过k点：d[i][j][k]=d[k][i][j]

状态转移方程:

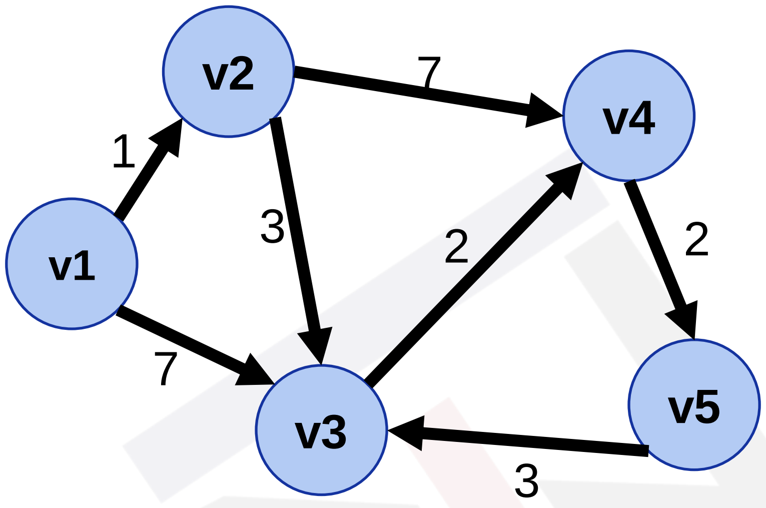
d[k][i][j]=min{ d[k-1][i][k]+d[k-1][k][j], d[k-1][i][j]}

边界条件:

联通点：dp[0][i][j]=mp[i][j] (mp[i][j]表示i到j的权值)；

不联通点： dp[0][i][j]= ∞；

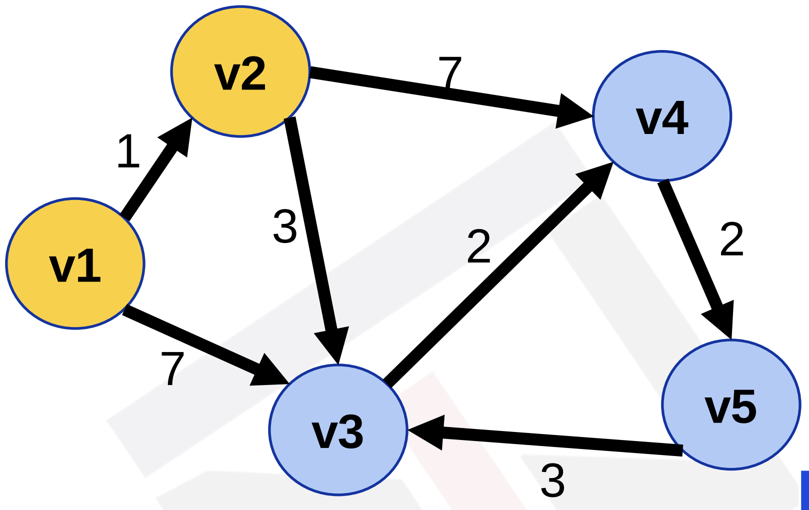
Floyd算法



初始化

dp	v1	v2	v3	v4	v5
v1	∞	1	7	∞	∞
v2	∞	∞	3	7	∞
v3	∞	∞	∞	2	∞
v4	∞	∞	∞	∞	2
v5	∞	∞	3	∞	∞

Floyd算法

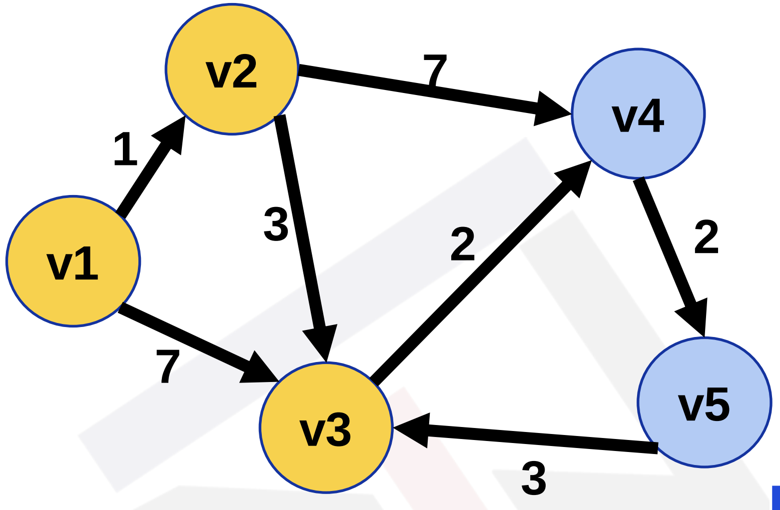


以**v1**位中转，更新矩阵，无变化

以**v2**位中转，更新矩阵

dp	v1	v2	v3	v4	v5
v1	∞	1	**4**	**8**	∞
v2	∞	∞	3	7	∞
v3	∞	∞	∞	2	∞
v4	∞	∞	∞	∞	2
v5	∞	∞	3	∞	∞

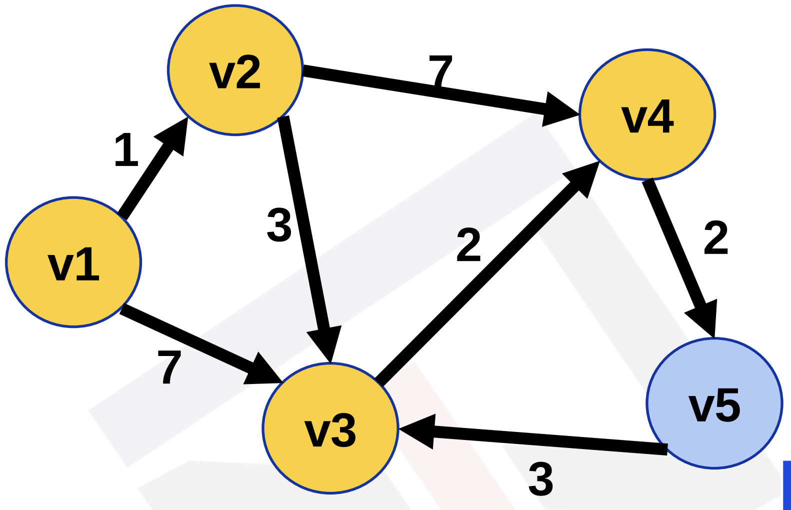
Floyd算法



以**v3**位中转，更新矩阵

dp	v1	v2	v3	v4	v5
v1	∞	1	4	**6**	∞
v2	∞	∞	3	**5**	∞
v3	∞	∞	∞	2	∞
v4	∞	∞	∞	∞	2
v5	∞	∞	3	∞	∞

Floyd算法



以v4位中转，更新矩阵

dp	v1	v2	v3	v4	v5
v1	∞	1	4	6	8
v2	∞	∞	3	5	7
v3	∞	∞	∞	2	4
v4	∞	∞	∞	∞	2
v5	∞	∞	3	∞	∞

Floyd算法—空间降维

算法描述：

在d[k][i][j] 中，因为**k是递增的**，处理经过k点时，**d[k][i][j]** 保存的状态就是上一个状态**d[k-1][i][j]** ，所以可以减少一维，使用二维数组。

（类似背包问题降维）

```
for(int k=1;k<=n;k++)   //阶段一定放在最外层
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            if(d[i][j]>d[i][k]+d[k][j])
                d[i][j]=d[i][k]+d[k][j];
```

最后d[i][j]表示什么？

i到j的最短路径

时间复杂度：

O(n^3)

Floyd算法小结

本质：

动态规划

一般使用什么方式储存图？

时间复杂度：o(n^3)

邻接矩阵

适用范围：

Floyd是解决多源最短路径的算法

边权可正可负，但是不能存在负权环

如果我们要输出最短路路径怎么办?

添加数组pre[][],pre[i][j]为i到j的的中转点

pre[i][j]=i; 初始化

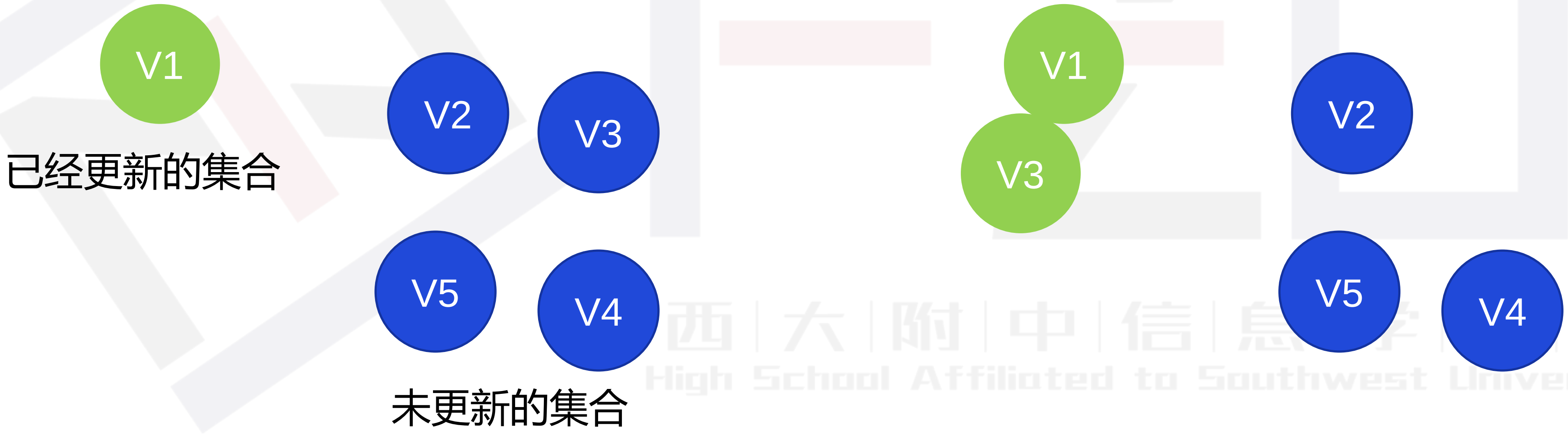
```
if(d[i][j]>d[i][k]+d[k][j])
{
    d[i][j]=d[i][k]+d[k][j];
    pre[i][j]=k;
}
```

Dijkstra

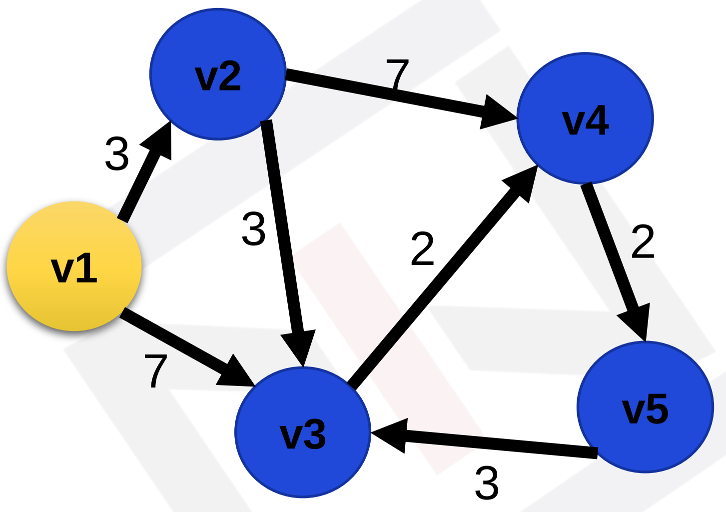
Dijkstra

Dijkstra本质上是贪心策略

每次从未更新的结点集合里，选择一个与当前集合的结点最近(边权最小)的结点加入



Dijkstra算法



结点1为**源点**

v1能直接到的节点为：

v2、v3

距离为：3、7

那么我们能确定最短路径的点为：

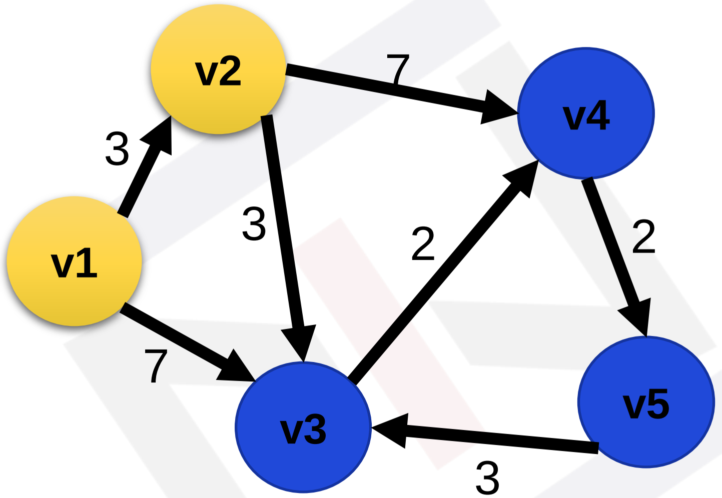
v2

使用算法的前提：

没有负权的边；

想一下原因：

因为所有边权为正数，不可能通过其它中转点，使得v1到v2的花费更短。

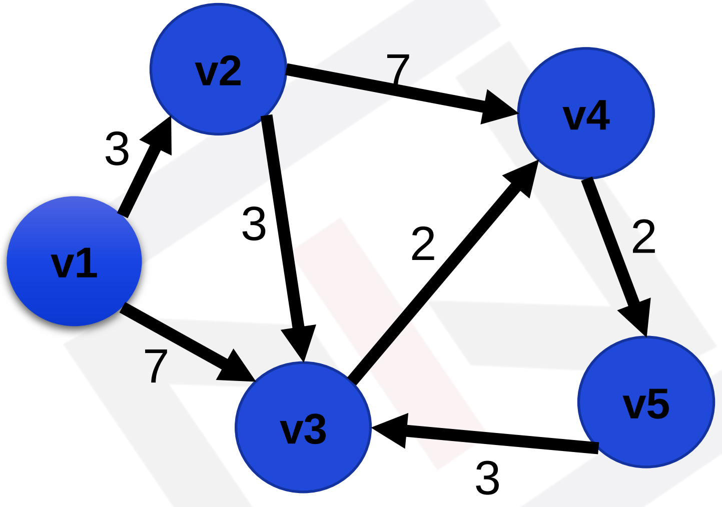


确定最短路径的点为：v1、v2

能到达的点为：v3、v4

分别的路径长度为：6、10

此次能确定的最短路径的点为：v3



每次选择**未确定**最短路径的点中，
到源点**路径长度最小的点**

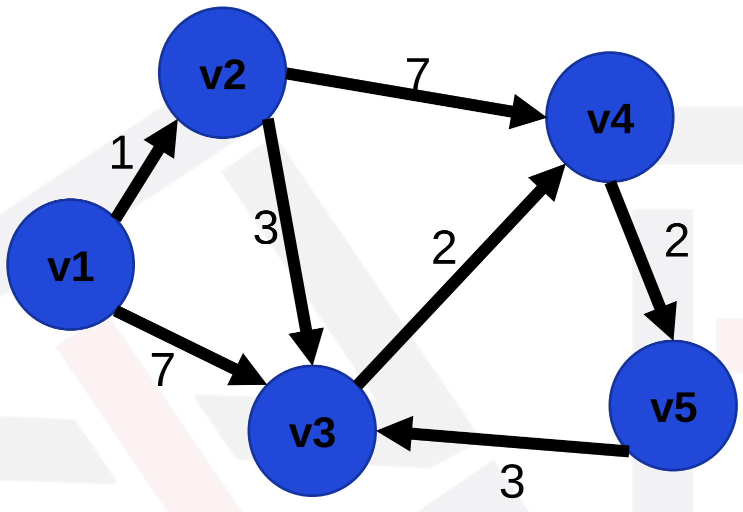
贪心的思想

那么Dijkstra算法需要**存储**的信息：

未确定最短路径的点：
vis[i]数组

某点i到源点的距离：
dis[i]数组表示当前距源点的距离

Dijkstra算法



dis[1]	0
dis[2]	∞
dis[3]	∞
dis[4]	∞
dis[5]	∞

vis[1]	0
vis[2]	0
vis[3]	0
vis[4]	0
vis[5]	0

1. 初始化：

dis[i]=∞, dis[1]=0。

vis[i]=0。//标记是否访问

1. 从dis中找到未访问的顶点且dis[]最小

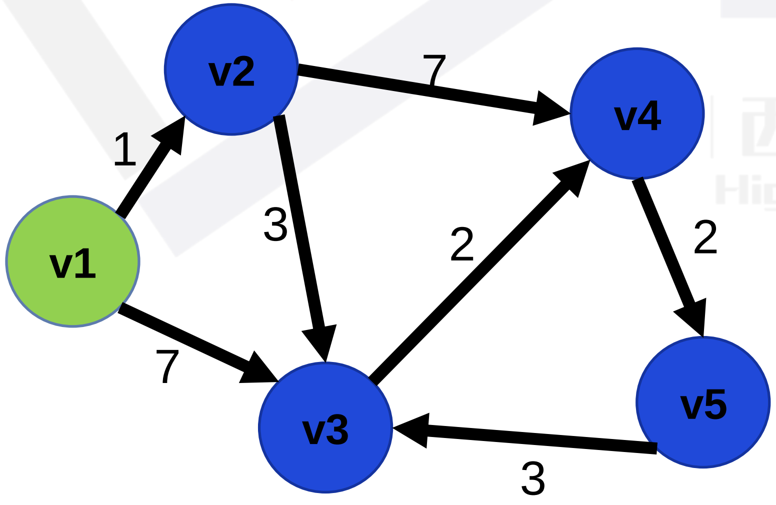
dis[1]最小，那么起点到1的最短距离就确定了，标记为找到。

更新**1能够到达**的点i，看是否能使dis[i]更短。

更新:

dis[2]=dis[1]+w[1][2]=1,

dis[3]=dis[1]+w[1][3]=7。



dis[1]	0
dis[2]	1
dis[3]	7
dis[4]	∞
dis[5]	∞

vis[1]	1
vis[2]	0
vis[3]	0
vis[4]	0
vis[5]	0

2. 从dis中找到未访问的顶点且dis[]最小

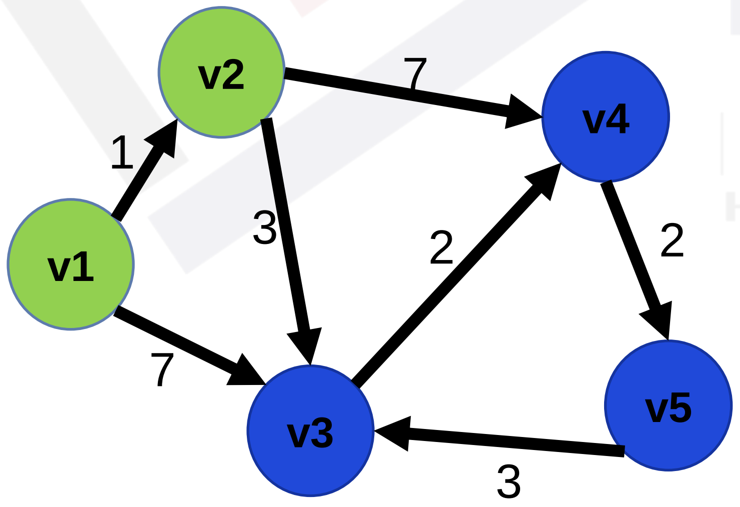
dis[2]最小，那么起点到2的最短距离就确定了，标记为找到。

更新**2能够到达**的点i，看是否能使dis[i]更短。

更新:

dis[3]=dis[2]+w[2][3]=4,

dis[4]=dis[2]+w[2][4]=8.



dis[1]	0
dis[2]	1
dis[3]	4
dis[4]	8
dis[5]	∞

vis[1]	1
vis[2]	1
vis[3]	0
vis[4]	0
vis[5]	0

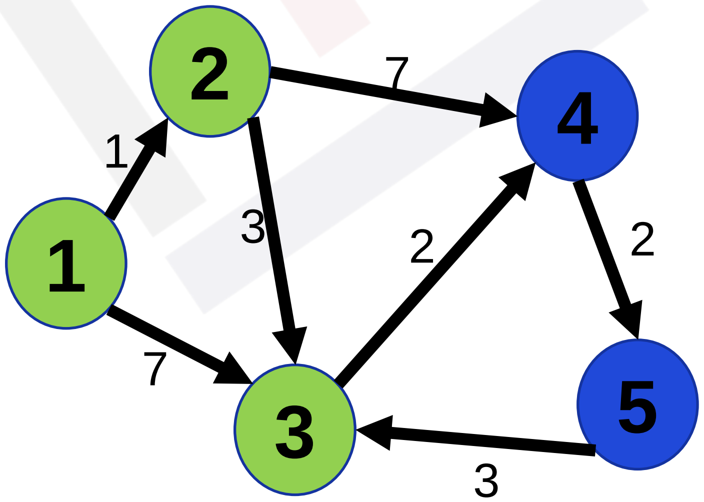
3. 从dis中找到未访问的顶点且dis[]最小

dis[3]最小，那么起点到3的最短距离就确定了，标记为找到。

更新**3能够到达**的点i，看是否能使dis[i]更短。

更新：

dis[4]=dis[3]+w[3][4]=6,



dis[1]	0
dis[2]	1
dis[3]	4
dis[4]	6
dis[5]	∞

vis[1]	1
vis[2]	1
vis[3]	1
vis[4]	0
vis[5]	0

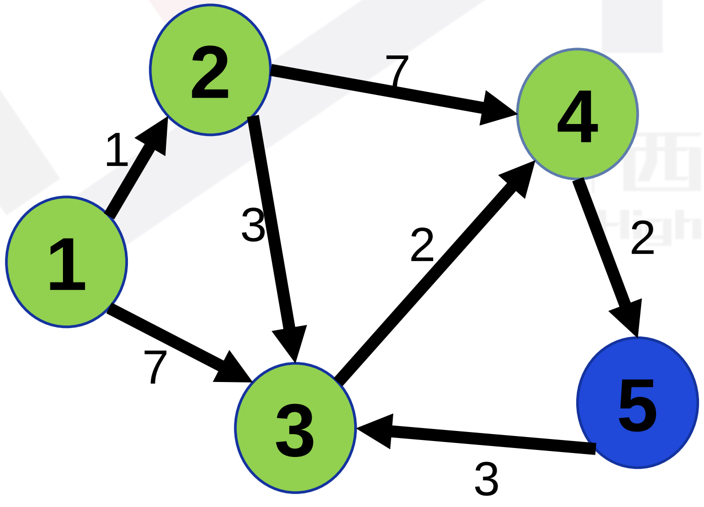
4. 从dis中找到未访问的顶点且dis[]最小

dis[4]最小，那么起点到4的最短距离就确定了，标记为找到。

更新**4能够到达**的点i，看是否能使dis[i]更短。

更新:

dis[5]=dis[4]+w[4][5]=8,



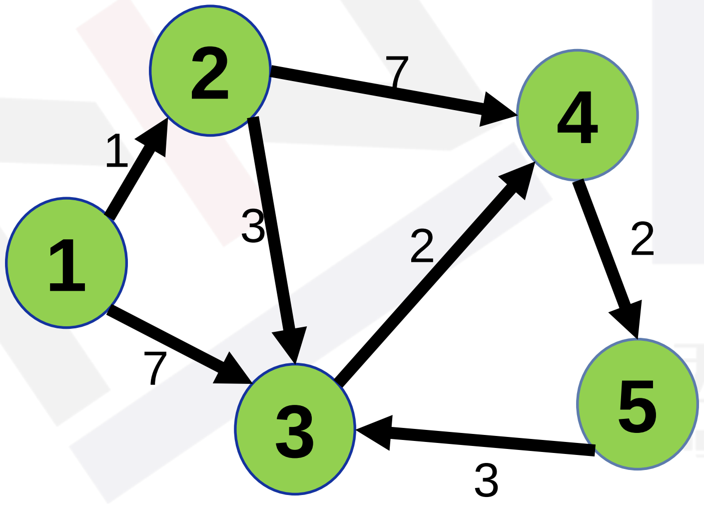
dis[1]	0
dis[2]	1
dis[3]	4
dis[4]	6
dis[5]	8

vis[1]	1
vis[2]	1
vis[3]	1
vis[4]	1
vis[5]	0

5. 从dis中找到未访问的顶点且dis[]最小

dis[5]最小，那么起点到5的最短距离就确定了，标记为找到。

更新**5能够到达**的点i，看是否能使dis[i]更短。

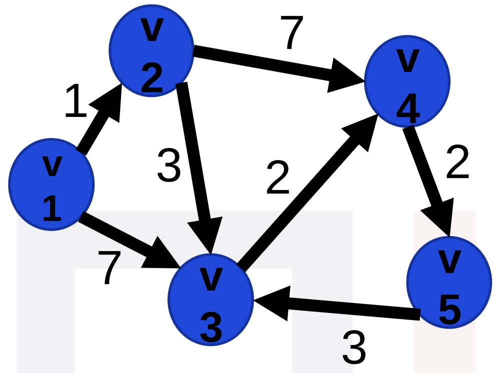


dis[1]	0
dis[2]	1
dis[3]	4
dis[4]	6
dis[5]	8

vis[1]	1
vis[2]	1
vis[3]	1
vis[4]	1
vis[5]	1

每一次寻找，找到一个点的最短路，n次寻找就找到了n个点的最短路。

Dijkstra算法过程



dis[i]表示从源点s到 i 的距离;
vis[i]表示是否得到从源点s到 i 的最小值;

① 初始化dis[i]=∞,dis[s]=0;
② 选择一个未标记的点k(vis[i]=0),并且dis[k]的值最小;
③ 标记点k(vis[k]=1);
④ 以k为中间点,修改dis[i]的值;
⑤ 重复步骤2到步骤4,n次过后求得s到各点的最小值

Dijkstra算法

算法描述：

设起点为s，dis[v]表示从s到v的最短路径。

1. **初始化**：dis[v]=∞；dis[s]=0；

2. for (i = 1; i <= n ; i++)

    a)在**没有被访问过的点**中找一个顶点u使得**dis[u]是最小**的。

    b)u标记为已确定最短路径

    for:判断是否能**更新与u相连的顶点**v的dis

```
if(dis[u]+w[u][v] < dis[v])
{
    dis[v] = dis[u] + w[u][v];
}
```

4. 算法结束：dis[v]为s到v的最短距离；

时间复杂度

O(n^2)

Dijkstra算法

西南大学附属中学 High School Affiliated to Southwest University

```
for(i=1;i<=n;i++)
{
     maxn=inf;
     for(j=1;j<=n;j++)                    //找出未访问最小的dis[j]
     {
          if( !vis[j] && dis[j]<maxn)
         {
                maxn=dis[j];k=j;
          }
     }
     vis[k]=1;
     for(j=1;j<=n;j++)          //k作为中间点，更新起点经过k到达其他点的dis[j]
          if(w[k][j])
                dis[j]=min{dis[k]+w[k][j],dis[j]};
}
```

如果我们要输出最短路路径

添加数组pre[],pre[v]为v的前驱节点

pre[s]=0; 初始化源点

for:判断是否能更新与u相连的顶点v的dis

```
if(dis[u]+w[u][v] < dis[v])
{
    dis[v] = dis[u] + w[u][v];
    pre[v] = u;
}
```

324. 最短路径问题

Dijkstra算法

题目描述

平面上有ｎ个点（ｎ＜＝10000），每个点的坐标均在－１００００～１００００之间，其中的一些点之间有连线。

若有连线，则表示可从一个点到达另一个点，即两点间有同路，同路的距离为两点间的直线距离。现在的任务是找出从一点到另一点之间的最短路径。

输入

共ｎ＋ｍ＋３行。

第一行：整数ｎ

第２行到第ｎ＋１行（共ｎ行），每行两个整数ｘ和ｙ，描述一个点的坐标。

第ｎ＋２行为一个整数m，m<=100000，表示图中连线的个数。

此后的m行，每行描述一条连线，由两个整数ｉ和ｊ组成，表示第ｉ个点和第ｊ个点之间有连线。

最后一行：两个整数ｓ和ｔ，分别表示源点和目标点。两个数之间用一个空格隔开。

输出

仅一行，一个实数（保留两位小数），表示从ｓ到ｔ的最短路径长度。

Dijkstra算法

与上一题有什么**不同**?

n<100 → n<10000

还能不能用二维数组存储???

知识链接： 图的存储

Dijkstra算法还能不能解决???

优化Dijkstra算法

Dijkstra+堆优化

Dijkstra算法

西南大学附属中学 High School Affiliated to Southwest University

算法描述：

设起点为s，dis[v]表示从s到v的最短路径。

1. **初始化**：dis[v]=∞；dis[s]=0；

2. for (i = 1; i <= n ; i++)

　　a) 在**没有被访问过的点**中找一个顶点u使得**dis[u]是最小**的。

　　b) u标记为已确定最短路径

　　　　for:判断是否能**更新与u相连的顶点**v的dis　**堆**

```
if(dis[u]+w[u][v] < dis[v])
{
    dis[v] = dis[u] + w[u][v];
}
```

4. 算法结束：dis[v]为s到v的最短距离；

哪些地方可以优化?

Dijkstra的堆优化

算法描述：

设起点为s，dis[v]表示从s到v的最短路径。

1. **初始化**：dis[v]=∞；dis[s]=0;

2. for (i = 1; i <= n ; i++)

    a) 在**没有被访问过的点**中找一个顶点u使得**dis[u]是最小**的。 堆优化O(log(n))

    b) u标记为已确定最短路径

        for:判断是否能**更新与u相连的顶点**v的dis 邻接表存储优化

            if(dis[u]+w[u][v] < dis[v])

            {

                dis[v] = dis[u] + w[u][v]; → **松弛操作 更新堆**

            }

4. 算法结束：dis[v]为s到v的最短距离；

堆结点信息：

dis[i]以及点的编号i

Dijkstra的堆优化

算法描述：

设起点为s，dis[v]表示从s到v的最短路径。

1. **初始化**：dis[v]=∞；dis[s]=0；

2. for （i = 1；i <= n ；i++)

a) 在**没有被访问过的点**中找一个顶点u使得**dis[u]是最小**的。

b) u标记为已确定最短路径

for:判断是否能**更新与u相连的顶点**v的dis

```
if(dis[u]+w[u][v] < dis[v])
{
    dis[v] = dis[u] + w[u][v];
}
```

4. 算法结束：dis[v]为s到v的最短距离；

如何更新？

松弛操作，在堆中找到需要更新的点

实现难度大，复杂度也未减小

堆的特性：**堆顶最小**(小根堆)

将松弛后的点加入堆中,不管之前是否在堆中

如何防止**重复访问**一个点

在pop()时判断是否已经访问过

算法描述：
设起点为s，dis[v]表示从s到v的最短路径。
1. **初始化**：dis[v]=∞；dis[s]=0，(**put(s和dis[s])**)；
2. while(堆不为空)
    a) get()得到结点u；
    b) **如果u访问过，continue；**
    c) u标记为已确定最短路径
        for:判断是否能**更新与u相连的顶点**v的dis
```
if(dis[u]+w[u][v] < dis[v])
{
    dis[v] = dis[u] + w[u][v];
    put(u和dis[u]);
}
```
4. 算法结束：dis[v]为s到v的最短距离；

堆优化代码实现

西南大学附中
High School Affiliated to Southwest University

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=10010,M=100010;
double xx[N],yy[N];
int n,m,s,t;
int fir[N],to[M],nex[M],tot;
double val[M];
double dis[N];
bool vis[N];
void add(int a,int b,double c){
            val[++tot]=c;
            to[tot]=b;
            nex[tot]=fir[a];
            fir[a]=tot;
}
struct node{
            double d;
            int num;
            bool operator < (const node& t) const
            {
                        return d>t.d;           //小根堆
            }
};
```

```cpp
priority_queue <node> q;
void dijkstra(){
                memset(dis,127,sizeof(dis));
                dis[s]=0.0;
                node t1;
                t1.d=0.0; t1.num=s;
                q.push(t1);
                while(!q.empty())
                {
                   node u=q.top(); q.pop();
                   if(vis[u.num]) continue;
                   vis[u.num]=1;
                   for(int i=fir[u.num];i;i=nex[i]){
                      int v=to[i];
                      double w=val[i];
                      if(u.d+w<dis[v]){
                                dis[v]=u.d+w;
                                node x;
                                x.d=dis[v];
                                x.num=v;
                                q.push(x);
                      }
                   }
                }
                printf("%.2lf\n",dis[t]);
}
int main(){
                cin>>n;
                for(int i=1;i<=n;i++)   cin>>xx[i]>>yy[i];
                cin>>m;
                int t1,t2; double t3,t4;
                for(int i=1;i<=m;i++){
                                t3=xx[t1]-xx[t2];
                                cin>>t1>>t2;
                                t4=yy[t1]-yy[t2];
                                add(t1,t2,sqrt(t3*t3+t4*t4));
                                add(t2,t1,sqrt(t3*t3+t4*t4));
                }
                cin>>s>>t;
                dijkstra();
                return 0;
}
```

Dijkstra的堆优化

STL pair<>版

```
int main()
{
        cin>>n>>m;
        for(int i=1;i<=m;i++){
                int x,y,z;
                cin>>x>>y>>z;
                ver[++tot]=y,edge[tot]=z;
                next[tot]=head[x],head[x]=tot;
        } //构建邻接矩阵
        dijkstra();
        for(int i=1;i<=n;i++)
        cout<<d[i]<<endl;
        return 0;
}
```

```
const int N=10010,M=1000010;
int head[N],ver[M],edge[M],next[M],d[N];
bool v[N];
int n,m,tot;
priority_queue< pair<int,int> > q;
//大根堆 优先队列 pair第一维为dist相反数（变成小根堆） 第二维为节点编号

void dijkstra(){
        memset(d,0x3f,sizeof(d));
        d[1]=0;//dist初始化 起点为0，其余为正无穷
        memset(v,0,sizeof(v));//节点标记
        q.push(make_pair(0,1));
        while(q.size()){
                int x=q.top().second;q.pop();//取堆顶
                if(v[x]) continue;v[x]=1;
                for(int i=head[x];i;i=next[i]){//扫描所有出边
                        int y=ver[i],z=edge[i];
                        if(d[y]>d[x]+z) {
                                d[y]=d[x]+z;
                                q.push(make_pair(-d[y],y));
                        }
                }
        }
}
```

Dijkstra算法

本质：贪心策略

时间复杂度：o(n^2)

优化：

每一次查找最小值o(n)

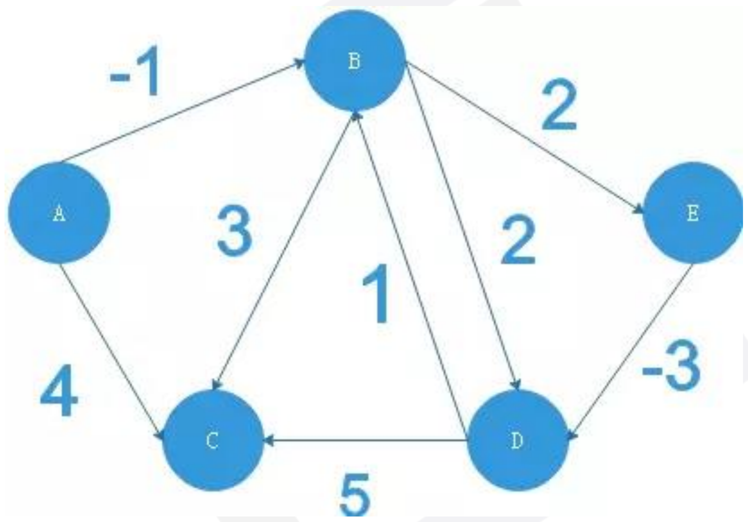
使用堆或者优先队列优化到o(log(n))

限制： Dijkstra 是解决单源最短路径的算法

不能解决边权为负的情况

分界线

Bellman-Ford



松弛操作：

dis[i]表示源点s到i的距离

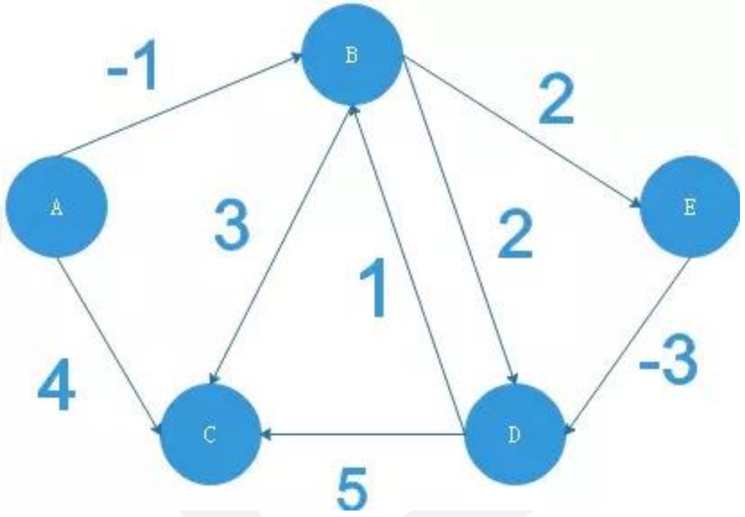
针对每条边<a,b>：
如果dis[a]+w[a][b]<dis[b]
则 dis[b]=dis[a]+w[a][b]

```
void relax(a,b)
{
        if(dis[a]+w[a][b]<dis[b])
                dis[b]=dis[a]+w[a][b];
}
```

不难发现求最短路就是进行**若干次松弛操作**，使得dis[i]趋于更小

那我们可以怎样进行松弛？？？

Bellman-Ford(福特)算法



和迪杰斯特拉算法一样，会使用到dis[i]

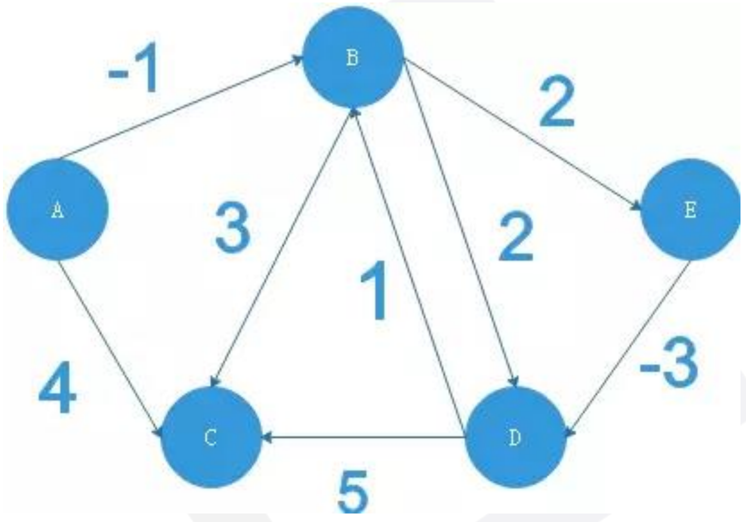
初始化：dis[s]=0,dis[i]=∞

因为只有源点的dis已知，所以第一轮松弛肯定要松弛与源点相连的边

怎么知道哪些边与源点相连?
第一轮松弛后怎么知道又有哪些边要松弛?

福特算法：**暴力枚举所有的边**

Bellman-Ford(福特)算法



那要松弛多少轮才能得到最小的路径？

极端情况：

最多松弛**n-1**(n为顶点数)次

福特算法：

对整个图进行**n-1**（n代表顶点数）次遍历操作；

每次遍历，对所有的边进行一次**松弛**操作

Bellman-Ford(福特)算法

核心代码：

```
for(int i=1;i<=n-1;i++)
      for(int j=1;j<=m;j++)   //m表示边的数量
            relax(u[j],v[j]);
```

时间复杂度：

O(ne)

n为顶点数，e为边集

Bellman-Ford(福特)算法

福特算法：

时间复杂度

O(ne) 时间复杂度高于Dijkstra

不但可求**负权边**，还能判断**是否存在负权环**

怎么判断？

进过n-1次循环后，最短路肯定可以确定

但是如果我们继续松弛，还能发现

dis[a]+w[a][b]<dis[b]

说明存在负权环

Bellman-Ford(福特)算法

```
bool ford () {
    for(int i=1;i<=n-1;i++)
        for(int j=1;j<=m;j++)  //m表示边的数量
            if(dis[v[j]]>dis[u[j]]+w[j])
                dis[v[j]]=dis[u[j]]+w[j];
    for(int j=1;j<=m;j++)
        if(dis[v[j]]>dis[u[j]]+w[j])
            return false;
}
```

Bellman-Ford(福特)算法

Ford算法：每一轮松弛枚举所有边

能否优化？？？

每次能够松弛的边是哪些？

是前一次**被松弛**那些入点

SPFA算法

用队列来优化

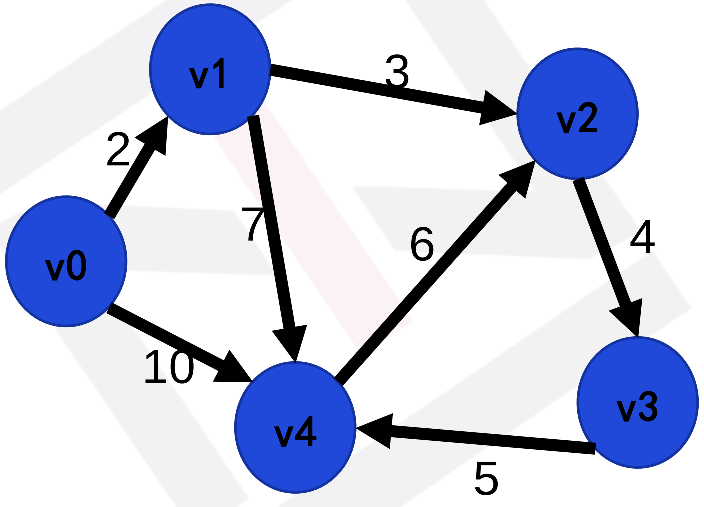
SPFA

①建立一个**队列**，初始化队列只有起点s；

②在建立一个数组**dis[i]**记录起始点s到i的最短路径

③ 将**队首**的点作为起始点去刷新所有的最短路，如果刷新成功且刷新点不在队列中就将该点入队；
（可以用**inq[i]**表示是否在队列中）

④ 重复执行，直至**队列为空**。



inq	1	0	0	0	0
q	v0				

	v0	v1	v2	v3	v4
dis	0	∞	∞	∞	∞

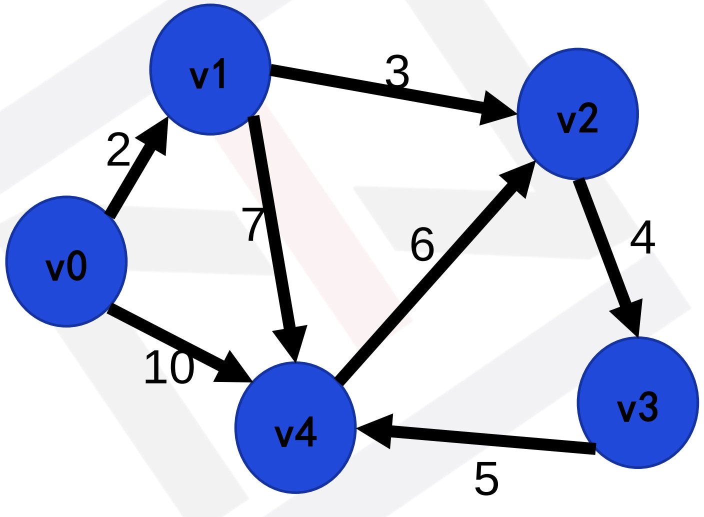
(1) 源点入队，dis[v0]=0,其余为∞

inq	0	1	0	0	1
q	v1	v4			

	v0	v1	v2	v3	v4
dis	0	2	∞	∞	10

(2) 源点v0出队，v1,v4入队；
dis[v1]=2,dis[v4]=10

SPFA



inq	0	0	1	0	1
q	v4	v2			

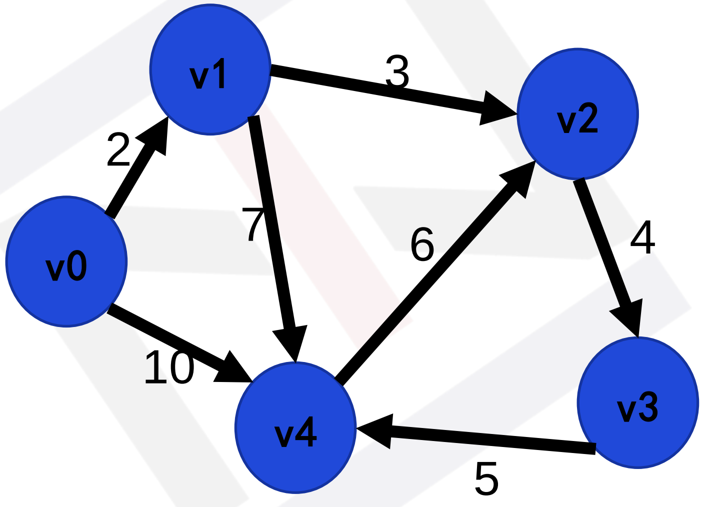
	v0	v1	v2	v3	v4
dis	0	2	5	∞	9

(3) v1出队，v2入队；
dis[v2]>dis[v1]+a[v1][v2]=5,更新
dis[v4]>dis[v1]+a[v1][v4]=9,更新

inq	0	0	1	0	0
q	v2				

	v0	v1	v2	v3	v4
dis	0	2	5	∞	9

(4) v4出队



inq	0	0	0	1	0
q	v3				

	v0	v1	v2	v3	v4
dis	0	2	5	9	9

(5) v2出队，v3入队
dis[v3]>dis[v2]+a[v2][v3]=9,更新

inq	0	0	0	0	0
q	v3				

	v0	v1	v2	v3	v4
dis	0	2	5	9	9

(6) v3出队，队列空
V0到各点的最短距离已求出

SPFA

西南大学附属中学 High School Affiliated to Southwest University

伪代码:

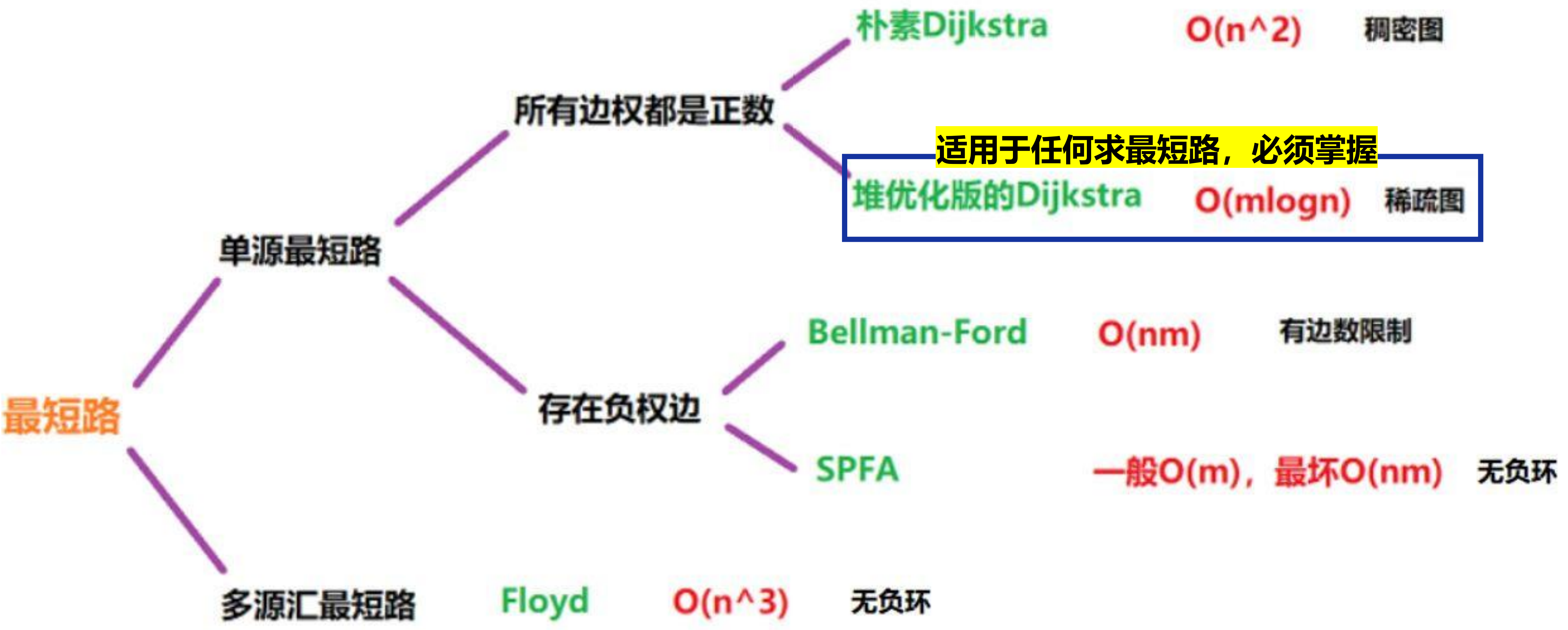
```
void  spfa(s) {//求单源点s到其它各顶点的最短距离
    for(int i=1;i<=n;i++){ dis[i]=∞; inq[i]=false; }   //初始化每点到s的距离，不在队列
    dis[s]=0;  //将dis[源点]设为0
    inq[s]=true; //源点s入队列
    head=0; tail=1; q[tail]=s; //源点s入队，头尾指针赋初值
    while(head<tail){
       head+=1;  //队首出队
       v=q[head];  //队首结点v
       inq[v]=false;  //释放对v的标记，可以重新入队
       for 每条边(v,i)  //对于与队首v相连的每一条边，建议用邻接表存储
          if (dis[i]>dis[v]+a[v][i])  //如果不满足三角形性质
               dis[i] = dis[v] + a[v][i]   //松弛dis[i]
               if (inq[i]=false) {tail+1; q[tail]=i; inq[i]=true;} //不在队列，则加入队列
       }
}
```

本质： 福特算法的队列优化

时间复杂度：o(kE) E是边的数量，k是很小常数

限制： SPFA 是解决单源最短路径的算法
可以解决边权为负的情况.
在某些题中可能存在一些数据卡SPFA(网格图+长链)

总结


最短路
单源最短路
所有边权都是正数
朴素Dijkstra
O(n^2)
稠密图
适用于任何求最短路，必须掌握
堆优化版的Dijkstra
O(mlogn)
稀疏图
存在负权边
Bellman-Ford
O(nm)
有边数限制
SPFA
一般O(m)，最坏O(nm)
无负环
多源汇最短路
Floyd
O(n^3)
无负环

Thanks

For Your Watching

FZOI

西|大|附|中|信|息|学|竞|赛|

High School Affiliated to Southwest University