图论

1、图的存储和遍历

类似迷宫类问题的深搜，这里从某个点出发能走到哪个点是从邻接矩阵中取数据判断。

```
#include <bits/stdc++.h>//1-2052      图的 dfs 遍历    javacn
using namespace std;
int n,e;
int a[20][20];// 邻接矩阵
bool f[20];// 节点是否访问
// 访问 x 节点
void dfs(int x)
{
      // 走过标记
      f[x] = true;
      cout<<x<<"   ";
    // 尝试访问 x 相邻的节点
      for(int i = 1;i <= n;i++)
      {
            if(a[x][i] == 1 && !f[i])
            {
                  dfs(i);
            }
      }
}
```

```
int main()
{
	cin>>n>>e;
	int x,y;
	// 存储邻接矩阵
	for(int i = 1;i <= e;i++)
	{
		cin>>x>>y;
		a[x][y] = 1;
		a[y][x] = 1;
	}

	// 从节点 1 开始访问
	dfs(1);
}
```

类似迷宫类问题的广搜，这里从某个点出发能走到哪些点，从邻接矩阵中取值判断。

```
#include <bits/stdc++.h>//2-2053  图的 bfs 遍历  javacn
using namespace std;
int n,e;
int a[20][20];// 邻接矩阵
bool f[20];// 节点是否访问
int q[20];// 存储走过的点
int head,tail;
// 访问 x 节点
void bfs()
{
	head = 1;
	tail = 1;
	q[1] = 1;// 从 1 的点开始访问
	f[1] = true;// 标记走过
	cout<<1<<"  ";
	while(head <= tail)
	{
		// 尝试 head 能访问到的所有的点
		for(int i = 1;i <= n;i++)
		{
			if(a[q[head]][i]==1&&!f[i])
			{
				// 入队
				tail++;
				f[i] = true;
				q[tail] = i;
				cout<<i<<"  ";
			}
		}

		head++;
	}
}
```

```
int main()
{
	cin>>n>>e;
	int x,y;
    // 存储邻接矩阵
	for(int i = 1;i <= e;i++)
	{
		cin>>x>>y;
		a[x][y] = 1;
		a[y][x] = 1;
	}

	// 从节点 1 开始访问
	bfs();
}
```

邻接表模板题，注意本题最终要求按照邻接点的编号从小到大打印，处理这个要求有 2 种思路：

1. 将邻接点存入数组或 vector 后排序，再打印；

2. 由于链式前向星在输出边时是按照存入的顺序反向输出的，因此我们可以将同一个顶点对应的邻接点，先按照邻接点的大小降序，再存入，这样输出时，就是按照从小到大的顺序输出的。

参考解法如下：

解法一：将待存储的边，先排序，再 add，从而使得输出时有序

```
#include <bits/stdc++.h>//3-2080-1    邻接点    javacn
using namespace std;

const int MAXN = 1e5;
// 采用链式前向星的方式来存储图（边）
struct Edge{
    // 从 u 出发，到 v 去，next 表示从 u 出发的
    // 上一条边在 edge 数组的哪个位置
      int u,v,next;
}edge[MAXN * 2];

// 存储每个点的最后一条边的起始位置
```

```
// 存储每个点的最后一条边的起始位置
int pre[1000100];
int k = 1;// 数组下标
int n;// 总共有多少条边

// 追加点
void add(int from,int to)
{
	edge[k].u = from;
	edge[k].v = to;
	edge[k].next = pre[from];// 前一条边的下标，形成边的链表
	pre[from] = k;// 更新 u 的上一条边的序号
	k++;
}

// 遍历函数
// 遍历函数
void print()
{
    // 遍历 n 个起点
	for(int i = 1;i <= n;i++)
	{
		if(pre[i] != 0) cout<<i<<endl;
		// 遍历以 i 为起点的边
		for(int j = pre[i];j != 0;j = edge[j].next)
		{
			cout<<edge[j].v;
			if(edge[j].next != 0)	cout<<" ";
		}
		if(pre[i] != 0) cout<<endl;
	}

}
```

```
struct node{
	int x,y;
};

node a[10010];

bool cmp(node n1,node n2)
{
	return n1.x<n2.x || n1.x==n2.x&&n1.y>n2.y;
}

int main()
{
    int u,v;
    int e;
    cin>>n>>e;
    for(int i = 1;i <= e;i++)
      {
	    cin>>a[i].x>>a[i].y;
      }

      sort(a+1,a+1+e,cmp);
      for(int i = 1;i <= e;i++)
      {
	    add(a[i].x,a[i].y);
      }
//  for(int i = 1;i <= n;i++){      cout<<pre[i]<<"   ";  }
//  cout<<endl;
//  for(int i = 1;i <= n;i++){
//      cout<<edge[i].u<<"   "<<edge[i].v<<"   "<<edge[i].next<<endl;
//  }
      print();
      return 0;
}
```

解法二：使用 vector 存储每个点相邻的边，sort 排序后输出。

```
#include<bits/stdc++.h>//3-2080-2    邻接点    javacn
using namespace std;

// 邻接表：用数组模拟链表
int pre[1000010];// 存储以每个点为出发点的最后一条边的编号
struct node{
	// 边的起止点，以及上一条边的编号
	int from,to,next;
};
node a[10010];
int k = 0;//k 表示 a 数组的长度
int n,e;
vector<int> v;// 存储每个点相邻的点有哪些

// 加边函数
void add(int u,int v)
{
	k++;
	a[k].from = u;
	a[k].to = v;
	// 更新本条边的上一条边的编号
	a[k].next = pre[u];
	pre[u] = k;// 更新以 u 点为出发点的最后一条边的编号
}
```

```
int main()
{
	cin>>n>>e;//n 个点 e 条边
	int x,y;
	for(int i = 1;i <= e;i++)
	{
		cin>>x>>y;
		add(x,y);
	}

	// 输出以每个点为出发点的各个边
	for(int i = 1;i <= n;i++)
	{
		if(pre[i] == 0) continue;

		v.clear();// 将 vector 清空
		cout<<i<<endl;
		//j 是边的下标
		for(int j = pre[i];j != 0;j = a[j].next)
		{
			//cout<<a[j].from<<" "<<a[j].to<<endl;
			v.push_back(a[j].to);
		}
		// 对 vector 排序
		sort(v.begin(),v.end());

		// 输出
		for(int j = 0;j < v.size();j++)
		{
			cout<<v[j]<<" ";
		}
		cout<<endl;
	}
	return 0;
}
```

2、欧拉图

欧拉路问题，注意本题如果有两个奇点，要从值较大的奇点出发，因此要就每个结点的度。

```
#include <bits/stdc++.h>//1-2055  欧拉图     javacn
using namespace std;
int a[30][30];
int n,e;
int d[30];
int r[50];
int k;

void dfs(int s)
{
	for(int i = 1;i <= n;i++)
	{
		if(a[s][i] == 1)
		{
			a[s][i] = 0;
			a[i][s] = 0;
			dfs(i);
		}
	}

	k++;
	r[k] = s;
}
```

```
int main()
{
	cin>>n>>e;
	int x,y;
	for(int i = 1;i <= e;i++)
	{
		cin>>x>>y;
		a[x][y] = 1;
		a[y][x] = 1;
		d[x]++;
		d[y]++;
	}

	int s = 1;
	for(int i = n;i >= 1;i--)
	{
		if(d[i] % 2 == 1)
		{
			s = i;
			break;
		}
	}

	dfs(s);

	 //输出
	for(int i = k;i >= 1;i--)
	{
		cout<<r[i]<<" ";
	}

	return 0;
}
```

欧拉图问题，本题要注意，两点之间可能有多个边，因此用邻接矩阵建边的时候，多个边要都建上，删边的时候，也要注意，一次只能删除一条边。

```
#include<bits/stdc++.h>//2-2054    骑马修栅栏    javacn
using namespace std;

const int N = 510;
int a[N][N],x,y,n;
int path[1030];// 存储走过的点
int k;// 路径数组下标
int d[N];// 存放每个点的度
int mi = INT_MAX,ma = INT_MIN;// 存放 x 和 y 的最小和最大值

// 深搜求欧拉路
void dfs(int s)
{
	// 从小到大尝试邻接点
	for(int i = mi;i <= ma;i++)
	{
		// 如果有路径可达
		if(a[s][i] > 0)
		{
			// 删除该边
			a[s][i]--;
			a[i][s]--;
			dfs(i);
		}
	}

	// 回溯时，该点没有任何其他的边可以尝试了，存储点
	// 第 1 次走过的路径不一定是正确的欧拉路径，回来时的路径是正确的
	path[k] = s;
	k++;
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>x>>y;
		// 邻接矩阵存储
		a[x][y]++;
		a[y][x]++;
		d[x]++;// 点的度增加
		d[y]++;
		mi = min(mi,min(x,y));
		ma = max(ma,max(x,y));
	}

	int s = mi;// 没有奇点从第 1 个点开始
	// 求第 1 个奇点
	for(int i = mi;i <= ma;i++)
	{
		if(d[i]%2==1)
		{
			s = i;// 从编号最小的点开始遍历
			break;
		}
	}

	dfs(s);// 从 s 点开始深搜

	for(int i = k - 1;i >= 0;i--)
	{
		cout<<path[i]<<endl;
	}
}
```

```cpp
#include <bits/stdc++.h>//3-2056  铲雪车 snow  hasome
using namespace std;

int main()
{
	long long x1,y1,x2,y2,s,e;
	long long h,m;
	double r=0;
	cin>>s>>e;
	while(cin>>x1>>y1>>x2>>y2)
	{
		r=r+sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
	}
	r=r*2/1000/20;
	h=(long long)(r);
	m=(long long)((r-h)*60+0.5);
	printf("%lld:%02lld\n",h,m);
    return 0;
}
```

3、最短路

迪杰斯特拉求最短路模板题：

```
#include <bits/stdc++.h>//1-2048-1  最短路径  javacn
using namespace std;
const int N = 60;
const int INF = 0x3f3f3f3f;
int n,s;
int d[N],f[N];
int a[N][N];


int main()
{
	cin>>n>>s;
	for(int i = 1;i <= n;i++)
	{
		for(int j = 1;j <= n;j++)
		{
			cin>>a[i][j];
		}
	}

    //dijkstra 求最短路
```

```
    //dijkstra 求最短路
    memset(d,0x3f,sizeof(d));
    d[s] = 0;
    // 求 n 次，每次确定一个点的最短路
    for(int i = 1;i <= n;i++)
    {
        int mi = -1;
        // 在没有确定的点中找最小
        for(int j = 1;j <= n;j++)
        {
            if(!f[j] && (mi == -1 || d[j] < d[mi]))
            {
                mi = j;
            }
        }
        // 确定该点的最短路
        f[mi] = true;
        // 更新 mi 到每个点的最短路
        for(int j = 1;j <= n;j++)
        {
            if(!f[j] && a[mi][j] != 0 && d[mi]+a[mi][j] < d[j])
            {
                d[j] = d[mi] + a[mi][j];
            }
        }
    }
    // 输出 s 点到每个点的距离
    for(int i = 1;i <= n;i++)
    {
        if(i != s)        {     cout<<(d[i]==INF?-1:d[i])<<" ";     }
    }
    return 0;
}
```

```cpp
#include<bits/stdc++.h>//1-2048-2  最短路径  w2016010182
using namespace std;
int W[115][115];
int D[115];
int pre[115];
bool V[115];
int n,s,x;
//void print(int i){
//    if(pre[i]!=0)     {     print(pre[i]);    }
//    printf("%d ",i);
//}
int main()
{
      scanf("%d%d",&n,&s);
      memset(W,0x3f,sizeof(W));
      memset(D,0x3f,sizeof(D));
      D[s]=0;
      for(int i=1;i<=n;i++)
      {
            for(int j=1;j<=n;j++)
            {
                  scanf("%d",&x);
                  if(x!=0)
                  {
                        W[i][j]=x;
                  }
                  if(i==j)
                  {
                        W[i][j]=0;
                  }
            }
      }
```

```
	for(int i=1;i<=n;i++)
	{
		int minn=0x3f3f3f3f;
		int k=0;
		for(int j=1;j<=n;j++)
		{
			if(!V[j]&&D[j]<minn)
			{
				minn=D[j];
				k=j;
			}
		}
		V[k]=true;
		for(int j=1;j<=n;j++)
		{
			if(D[k]+W[k][j]<D[j])
			{
				D[j]=D[k]+W[k][j];
//				pre[j]=D[j];
			}
		}
//		printf( “%d  “,pre[k]);
	}
//	print(n);
	for(int i=1;i<=n;i++)
	{
		if(i!=s)
		{
			if(D[i]!=0x3f3f3f3f)	{	printf("%d ",D[i]);	}
			else			{	printf("-1 ");		}
		}
	}
	return 0;
}
```

邻接矩阵 + dijkstra 求最短路，回溯打印路径。

```
#include <bits/stdc++.h>//2-2047    最短距离和路径问题    javacn
using namespace std;

int n,m,s,t;
int a[1010][1010];// 邻接矩阵
bool v[1010];// 是否确定有最短路
int dis[1010];// 最短路的值
int p[1010];// 每个点的来源点
int inf = 0x3f3f3f3f;

// 求最短路
```

```
// 求最短路
void dijkstra()
{
    // 初始化
      memset(dis,0x3f,sizeof(dis));
      dis[s] = 0;// 出发点的最短路确定

      for(int i = 1;i <= n;i++)
      {
            // 求没有确定过的点的最短路的下标
            int mi = -1;
            for(int j = 1;j <= n;j++)
            {
                  if(!v[j] && (mi == -1 || dis[j] < dis[mi]))
                  {
                        mi = j;
                  }
            }

            v[mi] = true;// 该点可以确认是最短

            // 从该点出发查找可更新的点
            for(int j = 1;j <= n;j++)
            {
                  // 没有确定是最短，且有路径、且路径更短
                  if(!v[j] && a[mi][j]!=0 && dis[mi] + a[mi][j] < dis[j])
                  {
                        dis[j] = dis[mi] + a[mi][j];
                        p[j] = mi;//j 点来自 mi 点
                  }
            }
      }
}

// 递归输出路线
```

```
// 递归输出路线
void print(int k)
{
	if(p[k] != 0)
	{
	  print(p[k]);
	}
    cout<<k<<" ";// 输出 k 点来自哪里
}
int main()
{
	cin>>n>>m>>s>>t;
	int x,y,l;
	// 读入，并初始化邻接矩阵
	for(int i = 1;i <= m;i++)
	{
		cin>>x>>y>>l;
		if(a[x][y]==0||l<a[x][y])
		{
			a[x][y] = l;
			a[y][x] = l;
		}
	}

	dijkstra();// 迪杰斯特拉求最短路
	if(dis[t] == inf) cout<<"can't arrive";
	else
	{
		cout<<dis[t]<<endl;
		// 输出路线
		print(t);
	}
	return 0;
}
```

邻接矩阵 + dijkstra 求最短路：

```
#include <bits/stdc++.h>//3-2044    城市之间的最短路  javacn
using namespace std;
int n,m;
int a[20][20];// 邻接矩阵
bool v[20];// 是否找到最短路
int dis[20];// 最短路的值
int s,t;// 起止点
// 迪杰斯特拉算法
void dijkstra()
{
	// 初始化
	memset(dis,0x3f,sizeof(dis));
	// 出发点到自己的最短路径是确定的
	dis[s] = 0;
	// 求解 n 次
	for(int i = 1;i <= n;i++)
	{
		int mi = -1;
		for(int j = 1;j <= n;j++)
		{
			// 如果该点的最短路没有确定
			if(!v[j] && (mi == -1 || dis[j] < dis[mi]))	{mi = j;	}
		}
		//mi 点的最短路确定
		v[mi] = true;
		//mi 能够到达的点中，更新更短的路
		for(int j = 1;j <= n;j++)
		{
			if(!v[j]&&a[mi][j]!=0&&dis[mi]+a[mi][j]<dis[j])
			{	dis[j] = dis[mi]+a[mi][j];	}
		}
	}
}
```

```
int main()
{
	cin>>n>>m;
	// 读入路线及长度
	int x,y,l;
	for(int i = 1;i <= m;i++)
	{
		cin>>x>>y>>l;
		// 存储邻接矩阵
		a[x][y] = l;
		a[y][x] = l;
	}

	cin>>s>>t;

	// 求从 s 到每个点的最短路
	dijkstra();

	if(dis[t] == 0x3f3f3f3f) cout<<" No path" ;
	else cout<<dis[t];
	return 0;
}
```

本题可以用邻接矩阵 +dijkstra 求解，也可以用邻接表 +dijkstra 求解。

参考解法一：邻接矩阵 +dijkstra

```
#include <bits/stdc++.h>//4-2049-1      两点之间的最短路径      javacn
using namespace std;

int a[110][110];// 邻接矩阵
int z[110][3];// 读入的 n 个点的坐标
int n,m;
double d[110];
bool f[110];
int s,t;

// 求第 x 个点和第 y 个点距离的平方
double fun(int x,int y){
    return sqrt((z[x][1] - z[y][1]) * (z[x][1] - z[y][1]) + (z[x][2] - z[y][2])
* (z[x][2] - z[y][2]));
}
```

```
//dijkstra 求最短路
void dijkstra()
{
	// 初始化
	for(int i = 1;i <= n;i++) d[i] = 0x3f3f3f3f;

	d[s] = 0;
	for(int i = 1;i <= n;i++)
	{
		int mi = -1;

		for(int j = 1;j <= n;j++)
		{
			if(!f[j]&&(mi==-1||d[j] < d[mi]))
			{
				mi = j;
			}
		}

		f[mi] = true;

		for(int j = 1;j <= n;j++)
		{
			if(!f[j] && a[mi][j]==1 && d[mi] + fun(mi,j) < d[j])
			{
				d[j] = d[mi] + fun(mi,j);
			}
		}
	}
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>z[i][1]>>z[i][2];
	}

	cin>>m;//m 对关系
	int x,y;
	for(int i = 1;i <= m;i++)
	{
	  cin>>x>>y;
	  // 无向图
	  a[x][y] = 1;
	  a[y][x] = 1;
	}
	cin>>s>>t;

	dijkstra();

	cout<<fixed<<setprecision(2)<<d[t];
	return 0;
}
```

参考解法二：邻接表 +dijkstra

```
#include <bits/stdc++.h>//4-2049-2    两点之间的最短路径    javacn
using namespace std;
int n,m;//n 个点，m 条边
// 点
struct node{
	int x,y;
}poi[110];
// 边
struct edge{
	int from,to,next;
	double len;
}a[10010];
int k = 0;//edge 数组的长度
int pre[110];// 每个点的最后一条边
double d[110]; // 走到每个点的最短路
bool f[110];// 走到哪些点的最短路确定了
int s,t;
// 求两点之间的距离
double dis(node n1,node n2)
{
	return sqrt((n1.x-n2.x)*(n1.x-n2.x)+(n1.y-n2.y)*(n1.y-n2.y));
}

// 建边
void add(int u,int v)
{
	k++;
	a[k].from = u;
	a[k].to = v;
	a[k].len = dis(poi[u],poi[v]);
	a[k].next = pre[u];
	pre[u] = k;
}
```

```
int main()
{
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>poi[i].x>>poi[i].y;
	}

	cin>>m;
	int x,y;
	for(int i = 1;i <= m;i++)
	{
		cin>>x>>y;
		// 无向图
		add(x,y);
		add(y,x);
	}

	cin>>s>>t;

	//dijkstra 求最短路
```

```
	//dijkstra 求最短路
	int mi;

	for(int i = 1;i <= n;i++)
	{
		d[i] = 0x3f3f3f3f;
	}

	d[s] = 0;
	for(int i = 1;i <= n;i++)
	{
		// 没有确定点中的最小值
		mi = -1;
		for(int j = 1;j <= n;j++)
		{
			if(!f[j] && (mi==-1||d[j]<d[mi]))
			{
				mi = j;
			}
		}

		f[mi] = true;// 确定这个点的值
		// 讨论 mi 可以去的点
		for(int j = pre[mi];j != 0;j = a[j].next)
		{
			int to = a[j].to;
			if(!f[to] && d[mi] + a[j].len < d[to])
			{
				d[to] = d[mi] + a[j].len;
			}
		}
	}
	cout<<fixed<<setprecision(2)<<d[t];
	return 0;
}
```

假设 x 点向 y 点汇款，汇率为 z% ，也就是 0.0z ，那么如果要使得 y 点收到 100 元，也就是 x 汇出金额*(1-0.0z)=100，那么 x 点汇出金额 =100/(1-0.0z) 。

利用这个原理，从终点 B ，反推如果终点 B 要收到 100 元，各个点要汇出的最少费用是多少钱。

```
#include<bits/stdc++.h>//5-2050      最少的手续费     javacn
using namespace std;

int a[2010][2010];// 邻接矩阵
int n,m,s,e; //s,e 表示起止点 start, end.
// 反推要使得 B 得到 100 元，那么每个点至少要汇出多少钱
double d[2010];
bool f[2010];
```

```
int main()
{
	cin>>n>>m;
	int x,y,z;
	for(int i = 1;i <= m;i++)
	{
		cin>>x>>y>>z;
		a[x][y] = z;
		a[y][x] = z;
	}
	cin>>s>>e;
	//dijkstra 求解
	for(int i = 1;i <= n;i++) d[i] = 0x3f3f3f3f;
	d[e] = 100;

	for(int i = 1;i <= n;i++)	// 反推走到每个点最少需要汇出的钱数
	{
		int mi = -1;
		for(int j = 1;j <= n;j++)
		{
			if(!f[j]&&(mi==-1||d[j]<d[mi]))	{	mi = j;	}
		}
		f[mi] = true;
		// 更新更短的路径
		for(int j = 1;j <= n;j++)
		{
			if(!f[j]&&a[mi][j]&&d[mi]/(1-a[mi][j]*0.01)<d[j])
			{
				d[j] = d[mi] / (1 - a[mi][j]*0.01);
			}
		}
	}
	cout<<fixed<<setprecision(8)<<d[s];
	return 0;
}
```

问题本质：求编号为 Z 的点到每个大写字母的最短路。

```
#include<bits/stdc++.h>//6-2091    回家 Bessie Come Home  javacn
using namespace std;

/*
求：哪头牛最快到谷仓。
1. 点与点之间通过无向边连接，点与点之间可能有重边，也可能有自环。
2. 普通的牧场标记为：a~z，A~Y，A~Y 中有牛。
3. Z 表示目的地：谷仓。
读入：有 P 条边。

分析：
1. 由于本题是无向图：因此 A~Y -> Z 的最短路，等价于 Z -> A~Y 的最短路。
2. 本题点的总数 N = 52 个。
3. 本题点的编号可以映射到 1~52，也可以直接用点的 ascii 码作为点的编号。
ascii 的范围：65~90，97~122，因此认为点的编号最大是：122。
*/
const int N = 130;
int a[N][N];// 邻接矩阵
int d[N];// 走到每个点的最短路
bool f[N];// 标记走到哪些点的最短路不用修改了
int m, n;
char x, y;
int len;
```

```
int main()
{
	cin >> m;
	for(int i = 1; i <= m; i++)
	{
		cin >> x >> y >> len;
		// 如果 xy 之间没有变：直接赋值
		// 如果有边：打擂台求最短
		if(a[x][y] == 0 || len < a[x][y])
		{
			a[x][y] = a[y][x] = len;
		}
	}

	//dijkstra
	// 初始化
	memset(d, 0x3f, sizeof(d));
	d['Z'] = 0;
	n = 52;// 点的数量

	// 将所有的字母拼成一个字符串
	string t = " ";
	for(char i ='A'; i <= 'Z'; i++) t += i;
	for(char i ='a'; i <= 'z'; i++) t += i;

    // 更新的次数
```

```
    // 更新的次数
      for(int i = 1; i <= n; i++)
      {
            // 求没有确定最短路的点中，最短路的最小值对应的点
            int mi = -1;
            // 循环所有的点
            for(int j = 1; j <= n; j++)
            {
                  int p = t[j];// 点的编号是 ascii
                  if(!f[p] && (mi == -1 || d[p] < d[mi]))
                  {
                        mi = p;
                  }
             }
            f[mi] = true;
            // 从 mi 出发看能更新到哪些其他点的最短路
            for(int j = 1; j <= n; j++)
            {
                  int p = t[j];
                  if(!f[p] && a[mi][p] && d[mi] + a[mi][p] < d[p])
                  {
                        d[p] = d[mi] + a[mi][p];
                  }
            }
      }
    // 最终求：Z 到 A~Y 哪个点的最短路的值最小
    // 相当于求：最小数下标
      char mi = 'A';
      for(char i = 'B'; i <= 'Y'; i++)
      {
            if(d[i] < d[mi]) mi = i;
      }
    cout << mi << " " << d[mi];
    return 0;
}
```

SPFA 求最短路：

解法一：使用 STL 中的 queue 求解

```
#include <bits/stdc++.h>//7-2051-1    有负权边的最短路    javacn
using namespace std;

struct edge{
	int from,to,len,next;
};

int n,e;
edge a[200010];
int k;
int pre[20010];
int d[20010];
queue<int> q;
bool f[20010];

void add(int u,int v,int l){
	k++;
	a[k].from = u;
	a[k].to = v;
	a[k].len = l;
	a[k].next = pre[u];
	pre[u] = k;
}
```

```
void spfa()
{
	memset(d,0x3f,sizeof(d));
	d[1] = 0;
	q.push(1);
	f[1] = true;// 入队标记
	while(!q.empty())
	{
		int h = q.front();

		for(int i = pre[h];i != 0;i = a[i].next)
		{
			int to = a[i].to;
			if(d[h] + a[i].len < d[to])
			{
				d[to] = d[h] + a[i].len;
				if(!f[to])
				{
					q.push(to);
					f[to] = true;// 在队列中
				}
			}
		}

		f[h] = false;// 出队标记
		q.pop();
	}
}
```

```
int main()
{
	cin>>n>>e;
	int x,y,l;
	for(int i = 1;i <= e;i++)
	{
		cin>>x>>y>>l;
		add(x,y,l);
	}

	spfa();

	for(int i = 2;i <= n;i++)
	{
		cout<<d[i]<<endl;
	}
}
```

解法二：使用循环数组模拟队列求解

```
#include<bits/stdc++.h>//7-2051-2    有负权边的最短路    javacn
using namespace std;

const int N = 2e4 + 10;
const int M = 2e5 + 10;
struct node{
    int to,next,len;
}a[M];
int pre[N],k = 0;
int d[N];
bool f[N];
int q[N],h,t;
int n,m;

void add(int u,int v,int len)
{
      k++;
      a[k] = {v,pre[u],len};
      pre[u] = k;
}
```

```
void spfa()
{
	memset(d,0x3f,sizeof(d));
	h = 0,t = 1;
	q[0] = 1;
	d[1] = 0;
	f[1] = true;
	while(h != t)
	{
		int p = q[h];
		h++;
		if(h == n) h = 0;
		f[p] = false;

		for(int i = pre[p];i != 0;i = a[i].next)
		{
			int to = a[i].to;
			if(d[p] + a[i].len < d[to])
			{
				d[to] = d[p] + a[i].len;
				if(!f[to])
				{
					q[t] = to;
					f[to] = true;
					t++;
					if(t == n) t = 0;
				}
			}
		}
	}
	for(int i = 2;i <= n;i++)
	{
		printf( “%d\n” ,d[i]);
	 }
}
```

```
int main()
{
	scanf("%d%d",&n,&m);
	int x,y,len;
	while(m--)
	{
		scanf("%d%d%d",&x,&y,&len);
		add(x,y,len);
	}

	spfa();
	return 0;
}
```