图论

链式前向星

定义+初始化+加边

```
const int N=20005;
const int M=20005;
int cnt;
int head[N];
struct Edge
{
    int to,next,w;
}e[M<<1];
void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}
void add(int u,int v,int w){
    e[cnt].to=v;
    e[cnt].w=w;
    e[cnt].next=head[u];
    head[u]=cnt++;
}
```

遍历

```
for(int i=head[u];~i;i=e[i].next)
{

}
```

对所有节点的最短路

Floy-Warshall算法

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 1000007
typedef long long LL;

using namespace std;
inline LL read()
{
    LL X=0,w=1; char ch=0;
    while(ch<'0' || ch>'9') {if(ch=='-') w=-1;ch=getchar();}
    while(ch>='0' && ch<='9') X=(X<<3)+(X<<1)+ch-'0',ch=getchar();
    return X*w;
}
inline void write(LL x) {
    if(x < 0)putchar('-'),x=-x;
    if (x > 9)write(x / 10);
    putchar(x % 10 + 48);
}
const LL INF = 0x7fffffffffffffffll;
LL e[505][505], dis[505][505], m, n, p; // e邻接矩阵存边，dis[i][j]存i到j的最短路
int main()
{
    n = read(),m = read(),p = read(); //读入边数m，点数n
    for (LL i = 1; i <= n; i++)
        for (LL j = 1; j <= n; j++)
        {
            if (i == j)
                e[i][j] = dis[i][j] = 0;
            else
                e[i][j] = dis[i][j] = INF;
        }
    //初始化
    for (LL i = 1, x, y, u; i <= m; i++)
    {
        x = read(),y = read(),u = read();
        e[x][y] = min(u,e[x][y]);
        dis[x][y] = min(u,dis[x][y]);
    } //读入
    for (LL k = 1; k <= n; k++)
        for (LL i = 1; i <= n; i++)
            for (LL j = 1; j <= n; j++)
            {
                LL tmp;
                if(dis[i][k]==INF ||dis[k][j]==INF )
                    tmp = INF;
                else
                    tmp = dis[i][k] + dis[k][j];
                if(tmp<0)
                    tmp = INF;
                dis[i][j] = min(dis[i][j], tmp); //跑一遍Floyd

            }

    for(LL i = 0;i<p;i++)
    {
        LL a = read();
        LL b = read();
        if(dis[a][b]==INF)
        {
            write(-1);
            putchar('\n');
        }
        else
        {
            write(dis[a][b]);
            putchar('\n');
        }

    }
    return 0;
}
```

单源最短路

jhljx水水的最短路径

Dijkstra

- 时间复杂度：

  $O(|V|^2)$

- 采用传统的邻接矩阵的形式

  ```
  #define M 101
  #define INF 0x3f3f3f3f
  int prev[M];
  int dist[M];
  int weights[M][M];
  int path[M],top=0;
  void dijkstra(int weights[][M], int vs, int prev[], int dist[], int
  vertexNum)
  {
      int i,j,k;
      int min;
      int tmp;
      int flag[M];      // flag[i]=1表示"顶点vs"到"顶点i"的最短路径已成功获取。
  ```

```
    // 初始化
    for (i = 0; i < vertexNum; i++)
    {
        flag[i] = 0;              // 顶点i的最短路径还没获取到。
        prev[i] = vs;              // 顶点i的前驱顶点为0。
        dist[i] = weights[vs][i];// 顶点i的最短路径为"顶点vs"到"顶点i"的权。
    }

    // 对"顶点vs"自身进行初始化
    flag[vs] = 1;
    dist[vs] = 0;

    // 遍历G.vexnum-1次；每次找出一个顶点的最短路径。
    for (i = 1; i < vertexNum; i++)
    {
        // 寻找当前最小的路径；
        // 即，在未获取最短路径的顶点中，找到离vs最近的顶点(k)。
        min = INF;
        for (j = 0; j < vertexNum; j++)
        {
            if (flag[j]==0 && dist[j]<min)
            {
                min = dist[j];
                k = j;
            }
        }
        // 标记"顶点k"为已经获取到最短路径
        flag[k] = 1;
        // 修正当前最短路径和前驱顶点
        // 即，当已经"顶点k的最短路径"之后，更新"未获取最短路径的顶点的最短路径和前驱顶
点"。
        for (j = 0; j < vertexNum; j++)
        {
            tmp = (weights[k][j]==INF ? INF : (min + weights[k][j])); // 防止
溢出
            if (flag[j] == 0 && (tmp  < dist[j]) )
            {
                dist[j] = tmp;
                prev[j] = k;
            }
        }
    }
}
int n,m,u,v,w;
int main()
{
    scanf("%d %d",&n,&m);
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            weights[i][j]=INF;
    for(int i=0;i<m;i++)
    {
        scanf("%d %d %d",&u,&v,&w);
        weights[u][v] = w;
        weights[v][u] = w;
    }
    dijkstra(weights,0,prev,dist,n);
    for(int t=4;t!=0;t=prev[t]) //路径追溯
    {
        path[top]=t;
        top++;
    }
    for(int i=0;i<=top;i++)
        printf("%d ",path[i]);
    return 0;
}
```

Dijkstra+链式前向星+堆优化

- **其中edge[i].to表示第i条边的终点,edge[i].next表示与第i条边同起点的下一条边的存储位置,edge[i].w为边权值.**

  另外还有一个数组heap[],它是用来表示**以i为起点的第一条边存储的位置,**实际上你会发现这里的第一条边存储的位置其实在**以i为起点的所有边的最后输入的那个编号.**

```
#include<cstdio>
#include<queue>
#include<cstring>
#include<algorithm>
#include<climits>
using namespace std;
const int maxn = 210;
const int maxm = 1010;
const int INF = 0x3f3f3f3f;
inline void write(int x) {
    if (x < 0)putchar_unlocked('-'), x = -x;
    if (x > 9)write(x / 10);
    putchar_unlocked(x % 10 + 48);
}
inline int read() {
    int k = 0, f = 1;
    char c = getchar_unlocked();
    while (c < '0' || c>'9') {
        if (c == '-')f = -1;
        c = getchar_unlocked();
    }
    while (c >= '0' && c <= '9') {
        k = (k << 1) + (k << 3) + c - 48;
        c = getchar_unlocked();
    }
    return k * f;
}
int t;
int n, m;
int u, v, w;
struct edge {
    int next, to, w;
}edges[maxm];
int head[maxn], cnt;
inline void addedge(int u, int v, int w) {
    edges[cnt].next = head[u];
    edges[cnt].to = v;
    edges[cnt].w = w;
    head[u] = cnt++;
}
int dis[maxn][maxn];
int max_dis;
bool vis[maxn];
struct node {
    int v, w;
    node(int _v = 0, int _w = 0) {v = _v, w = _w;}
    bool operator < (const node & o) const {
        return o.w < w;
    }
};
priority_queue<node> q;
inline void init() {
    memset(dis, 0x3f, sizeof(dis));
    memset(head, 0xff, sizeof(head));
    cnt = 0;
    max_dis = INT_MIN;
}
inline void dijkstra(int s, int* d) {
    while(!q.empty()) q.pop();
    memset(vis, 0, sizeof(vis));
    d[s] = 0;
    q.push(node(s, 0));
    while(!q.empty()) {
        node tmp = q.top(); q.pop();
        int u = tmp.v;
        if(vis[u])continue;
        vis[u] = true;
        for(int i = head[u]; ~i; i = edges[i].next) {
            int v = edges[i].to, w = edges[i].w;
            if(!vis[v] && d[v] > d[u] + w) {
                d[v] = d[u] + w;
                q.push(node(v, d[v]));
            }
        }
    }
    for(int i = 1; i <= n; ++i) {
        if(i == s)continue;
        if(d[i] != INF && d[i] > max_dis) max_dis = d[i];
    }
}
inline void print() {
    for(int i = 1; i <= n; ++i)
        for(int j = 1; j <= n; ++j)
            if(dis[i][j] != INF && dis[i][j] == max_dis)
                write(i), putchar_unlocked(' '), write(j),
putchar_unlocked('\n');
}

int main() {
    t = read();
    while(t--) {
        init();
        n = read(), m = read();
        while(m--) {
            u = read(), v = read(), w = read();
            if(u != v)addedge(u, v, w);
        }
        for(int i = 1; i <= n; ++i) dijkstra(i, dis[i]);
        print();
    }
}
```

Dijkstra+可以重复调用的链式前向星堆优化版本

```
#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>
using namespace std;
const int INF = 0x3f3f3f3f;
const int maxn = 1000010;
const int maxm = 50000010;
inline void write(int x)
{
    if (x < 0)
        putchar('-'), x = -x;
    if (x > 9)
        write(x / 10);
    putchar(x % 10 + 48);
}
inline int read()
{
    int k = 0, f = 1;
    char c = getchar();
    while (c < '0' || c > '9')
    {
        if (c == '-')
            f = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9')
    {
        k = (k << 1) + (k << 3) + c - 48;
        c = getchar();
    }
    return k * f;
}
struct edge
{
    int v, w, next;
} edges[maxm];
int head[maxn], cnt;
inline void addEdge(int u, int v, int w)
{
    edges[++cnt].v = v;
    edges[cnt].w = w;
    edges[cnt].next = head[u];
```

```cpp
    head[u] = cnt;
}
bool vis[maxn];
int dis[maxn], diss[maxn];
struct node
{
    int v, w;
    node(int _v = 0, int _w = 0) { v = _v, w = _w; }
    bool operator<(const node &o) const
    {
        return o.w < w;
    }
};
priority_queue<node> pq;
void dijkstra(int s)
{
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    dis[s] = 0;
    while (pq.size())
        pq.pop();
    pq.push(node(s, 0));
    while (!pq.empty())
    {
        node tmp = pq.top();
        pq.pop();
        int u = tmp.v;
        if (vis[u])
            continue;
        vis[u] = true;
        for (int i = head[u]; i; i = edges[i].next)
        {
            int v = edges[i].v;
            int w = edges[i].w;
            if (!vis[v] && dis[v] > w + dis[u])
            {
                dis[v] = w + dis[u];

                pq.push(node(v, dis[v]));
            }
        }
    }
}
void dijkstra_2(int s)
{
    memset(diss, 0x3f, sizeof(diss));
    memset(vis, 0, sizeof(vis));
    diss[s] = 0;
    while (pq.size())
        pq.pop();
    pq.push(node(s, 0));
    while (!pq.empty())
    {
        node tmp = pq.top();
        pq.pop();
        int u = tmp.v;
        if (vis[u])
            continue;
        vis[u] = true;
        for (int i = head[u]; i; i = edges[i].next)
        {
            int v = edges[i].v;
            int w = edges[i].w;
            if (!vis[v] && diss[v] > w + diss[u])
            {
                diss[v] = w + diss[u];

                pq.push(node(v, diss[v]));
            }
        }
    }
}
int n, m, k;
int u, v, w;
int main()
{
    n = read(), m = read(), k = read();
    while (m--)
    {
        u = read(), v = read(), w = read();
        addEdge(u, v, w);
        addEdge(v, u, w);
    }

    int ans = 0x7fffffff;
    dijkstra(1);
    dijkstra_2(n);
    for (int i = 0; i < k; i++)
    {
        int tmp = read();
        ans = min(ans, dis[tmp] + diss[tmp]);
    }
    write(ans);
}
```

Dijkstra+链式前向星+堆优化+vector(最强优化)

jhljx水水的最短路径

治安点

- **在更改判断条件的时候注意顺便改变初始化的条件！！！！！！！！**

```
#include<cstdio>
#include<cstring>
#include<queue>
#include<vector>
#include<algorithm>
#define maxn 100010
#define INF 0x3f3f3f3f
using namespace std;
bool occur[maxn];
```

```cpp
int d[maxn];
int n, m, k;
int x, y, t;
struct node {
    int v, w;
    node(int _v = 0, int _w = 0) { v = _v; w = _w; }
    bool operator < (const node& o) const {     //判断优先级
        return o.w < w;
    }
};
vector<node>g[maxn];
inline void init() {
    for (int i = 0; i < maxn; ++i) {
        g[i].clear();
        occur[i] = false;
        d[i] = INF;     //与源点相反
    }
}
void dijkstra(int s) {
    priority_queue<node>q;
    q.push(node(s, 0)); //与初始化相反
    d[s] = 0;
    while (!q.empty()) {
        node tmp = q.top();
        q.pop();
        int v = tmp.v;
        if (occur[v])continue;
        occur[v] = true;
        for (int i = 0; i < g[v].size(); ++i) {
            int v2 = g[v][i].v;
            int w = g[v][i].w;
            if (!occur[v2] && d[v2] > w + d[v]) {   //修改条件之处

                d[v2] = w + d[v];
                q.push(node(v2, d[v2]));
            }
        }
    }
}
inline int read()
{
    int X=0,w=1; char ch=0;
    while(ch<'0' || ch>'9') {if(ch=='-') w=-1;ch=getchar();}
    while(ch>='0' && ch<='9') X=(X<<3)+(X<<1)+ch-'0',ch=getchar();
    return X*w;
}
inline void write(int x) {
    if(x < 0)putchar('-'),x=-x;
    if (x > 9)write(x / 10);
    putchar(x % 10 + 48);
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        init();
        for (int i = 0; i < m; ++i) {
            x = read(), y = read(), t = read();
            g[x].push_back(node(y, t));
            g[y].push_back(node(x, t));
        }
        dijkstra(1);
        bool flag = false;
        for (int i = 2; i <= n; ++i) {
            if (d[i] < INF)write(d[i]);
            else write(-1);
            putchar(' ');
        }
        putchar('\n');
    }
}
```

SPFA（解决负权边）

- **在更改判断条件的时候注意顺便改变初始化的条件！！！！！！！！**

```
#include<iostream>
#include<cstring>
#include<queue>
using namespace std;
const int maxn=505,maxe=100001;
int n,m,cnt;
int head[maxn],dis[maxn],sum[maxn];
bool vis[maxn];//标记是否在队列中
struct node{
    int to,next,w;
}e[maxe];

void add(int u,int v,int w){
    e[cnt].to=v;
    e[cnt].next=head[u];
    e[cnt].w=w;
    head[u]=cnt++;
}

bool spfa(int u){
    queue<int>q;
    memset(vis,0,sizeof(vis));//标记是否在队列中
    memset(sum,0,sizeof(sum));//统计入队的次数
    memset(dis,0x3f,sizeof(dis));
    vis[u]=1;
    dis[u]=0;
    sum[u]++;
    q.push(u);
    while(!q.empty()){
        int x=q.front();
        q.pop();
        vis[x]=0;
        for(int i=head[x];~i;i=e[i].next){
            int v=e[i].to;
            if(dis[v]>dis[x]+e[i].w){   //判断条件
                dis[v]=dis[x]+e[i].w;
                if(!vis[v]){
                    if(++sum[v]>=n)
                        return true;
                    vis[v]=1;
                    q.push(v);
                }
            }
        }
    }
    return false;
}

void print(){//输出源点到其它节点的最短距离
    cout<<"最短距离："<<endl;
    for(int i=1;i<=n;i++)
        cout<<dis[i]<<" ";
    cout<<endl;
}

int main(){
    cnt=0;
    cin>>n>>m;
    memset(head,-1,sizeof(head));
    int u,v,w;
    for(int i=1;i<=m;i++){
        cin>>u>>v>>w;
        add(u,v,w);
    }
    if(spfa(1))
        cout<<"有负环！"<<endl;
    else
        print();
    return 0;
}
```

Bellman Ford(一般用于判断正负环)

- **在更改判断条件的时候注意顺便改变初始化的条件！！！！！！！！**

```
#include<iostream>
#include<cstring>
using namespace std;
struct node{
    int a,b,w;
}e[210];
int dis[110];
int n,m,cnt=0;

void add(int a,int b,int w){
    e[cnt].a=a;
    e[cnt].b=b;
    e[cnt++].w=w;
}

bool bellman_ford(int u){//求源点u到其它顶点的最短路径长度，判负环
    memset(dis,0x3f,sizeof(dis));
    dis[u]=0;
    for(int i=1;i<n;i++){//执行n-1次
        bool flag=false;
        for(int j=0;j<m;j++)//边数m或cnt
            if(dis[e[j].b]>dis[e[j].a]+e[j].w){ //判断条件
```

```
                dis[e[j].b]=dis[e[j].a]+e[j].w;
                flag=true;
            }
        if(!flag)
            return false;
    }
    for(int j=0;j<m;j++)//再执行1次，还能松弛说明有环
        if(dis[e[j].b]>dis[e[j].a]+e[j].w)  //判断条件
            return true;
    return false;
}

void print(){//输出源点到其它节点的最短距离
    cout<<"最短距离："<<endl;
    for(int i=1;i<=n;i++)
        cout<<dis[i]<<" ";
    cout<<endl;
}

int main(){
    int a,b,w;
    cin>>n>>m;
    for(int i=0;i<m;i++){
        cin>>a>>b>>w;
        add(a,b,w);
    }
    if(bellman_ford(1))//判断负环
        cout<<"有负环！"<<endl;
    else
        print();
    return 0;
}
```

对于必须经过某些（个）点的最短路

- 对于这类问题，需要拆点，把其变成两次单源最短路，必要的时候可以进行反向边操作。

拓扑排序

知识链接：拓扑排序

输入元素为排序元素

D.ly的排队问题 (zcmu.edu.cn)

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#define M 100007
using namespace std;
priority_queue<int, vector<int>, greater<int> > q;
vector<int> edge[M];
vector<int> ans;
int in[M];
set<int> appear;
char order[5];
```

```
int main()
{
    while (~scanf("%s", order))
    {
        appear.insert(order[0] - 'A');
        appear.insert(order[2] - 'A');
        if (order[1] == '>')
        {
            in[order[0] - 'A']++;
            edge[order[2] - 'A'].push_back(order[0] - 'A');
        }
        else
        {
            in[order[2] - 'A']++;
            edge[order[0] - 'A'].push_back(order[2] - 'A');
        }
    }
    for (int i = 0; i < 30; i++)
    {
        if (in[i] == 0 && appear.count(i) != 0)
            q.push(i);
    }
    while (!q.empty())
    {
        int p = q.top();
        q.pop();
        ans.push_back(p);
        for (int i = 0; i < edge[p].size(); i++)
        {
            int y = edge[p][i];
            in[y]--;
            if (in[y] == 0)
                q.push(y);
        }
    }
    if (ans.size() == appear.size())
    {
        for (int i = 0; i < ans.size(); i++)
            printf("%c", ans[i] + 'A');
        putchar('\n');
    }
    else
        puts("No Answer!");
    return 0;
}
```

编号小的尽量排在前面

生日宴会

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#define M 100007
using namespace std;
```

```
priority_queue<int, vector<int>> q;
vector<int> edge[M];
vector<int> ans;
int in[M], x, y, n, m;
set<int> appear;
int main()
{
    scanf("%d %d", &n, &m);
    while (m--)
    {
        scanf("%d %d", &x, &y);
        in[x]++;
        edge[y].push_back(x);
    }
    for (int i = 1; i <= n; i++)
    {
        if (in[i] == 0)
            q.push(i);
    }
    while (!q.empty())
    {
        int p = q.top();
        q.pop();
        ans.push_back(p);
        for (int i = 0; i < edge[p].size(); i++)
        {
            int y = edge[p][i];
            in[y]--;
            if (in[y] == 0)
                q.push(y);
        }
    }
    for (int i = n - 1; i >= 0; i--)
        printf("%d ", ans[i]);
    putchar('\n');

    return 0;
}
```

并查集的应用

判断通路

- 用并查集判断即可，不需要跑最短路算法。

最小生成树

```
#include <bits/stdc++.h>
#define mp make_pair
#define ms(a,b) memset(a,b,sizeof(a))
#define maxn 1000007
typedef long long LL;
using namespace std;

struct edge
```

```
{
    LL u,v;
    LL w;
};
struct edge edges[60000100];
LL i;
LL Father[1010000];
LL cnt;
long long res;
void initFather(LL vertexNum)
{
    LL i;
    for(i=1;i<=vertexNum;++i)
    {
        Father[i]=i;
    }
}
LL getFather(LL x)
{
    return Father[x]==x?x:(Father[x]=getFather(Father[x]));
}
void kruskal(LL vertexNum,LL edgeNum)
{
    LL p,q;
    cnt=0,res=0;


    for(i=0;i<edgeNum;++i)
    {
        p=getFather(edges[i].u);
        q=getFather(edges[i].v);
        if(p!=q)
        {
            Father[p]=q;
            res+=edges[i].w;
            cnt++;
        }
        if(cnt==vertexNum-1)
        {
            break;
        }
    }

}
int cmp(const void*p1,const void*p2)
{
    struct edge *a=(struct edge*)p1;
    struct edge *b=(struct edge*)p2;
    return a->w-b->w;
}
int main()
{
    LL n,m;
    scanf("%lld%lld",&n,&m);
    initFather(n);
    LL i;
    for(i=0;i<m;++i)
        scanf("%lld %lld %lld",&edges[i].u,&edges[i].v,&edges[i].w);
```

```c
    qsort(edges,m,sizeof(struct edge),cmp);
    kruskal(n,m);
    printf("%lld\n",res);
}
```