## A 图上最短路II

```
vector<pii> edge[N];

i64 dijkstra(int n,int m,int s,int t)
{
    vector<i64> dist(n + 1, 1e18);
    vector<int> vis(n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    heap.push({0, s});
    dist[s] = 0;
    while(heap.size())
    {
        auto[d, u] = heap.top();
        heap.pop();
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto [v, w] : edge[u])
        {
            if(dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
    return dist[t] == 1e18 ? -1 : dist[t];
}


signed main()
{
    IOS;
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    cout << dijkstra(n, m, s, t) << '\n';
    return 0;
}
```

## B. 图上最短路III

spfa算法

```
vector<pii> edge[N];

i64 spfa(int n,int m,int s,int t)
{
    vector<i64> dist(n + 1, 1e18);
    vector<int> vis(n + 1);
    queue<int> q;
    dist[s] = 0;
    q.push(s);
    vis[s] = 1;
    while(q.size())
    {
        auto u = q.front();
        q.pop();
        vis[u] = 0;
        for(auto [v, w] : edge[u])
        {
            if(dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                if(!vis[v])
                {
                    q.push(v);
                    vis[v] = 1;
                }
            }
        }
    }
    return dist[t];
}


signed main()
{
    IOS;
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
    }
    auto ans = spfa(n,m,s,t);
    if(ans >= 1e17) cout << "unreachable\n";
    else cout << ans << '\n';
    return 0;
}
```

bellman-ford算法

```
vector<array<int, 3>> edge;

i64 bellman_ford(int n,int m,int s, int t)
{
    vector<i64> dist(n + 1, 1e18);
    dist[s] = 0;
    for(int i = 1; i <= n - 1; i ++)
    {
        auto last = dist;
        for(auto[u, v, w] : edge)
        {
            if(dist[v] > last[u] + w)
            {
                dist[v] = last[u] + w;
            }
        }
    }
    return dist[t];
}

signed main()
{
    IOS;
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge.push_back({u, v, w});
    }
    auto ans = bellman_ford(n,m,s,t);
    if(ans >= 1e17) cout << "unreachable\n";
    else cout << ans << '\n';
    return 0;
}
```

## C. 图上最短路IV

```
vector<array<int, 3>> edge;

i64 bellman_ford(int n,int m,int s, int t,int k)
{
    vector<i64> dist(n + 1, 1e18);
    dist[s] = 0;
    for(int i = 1; i <= k; i ++)
    {
        auto last = dist;
        for(auto[u, v, w] : edge)
        {
            if(dist[v] > last[u] + w)
            {
                dist[v] = last[u] + w;
            }
        }
    }
    return dist[t];
}

signed main()
{
    IOS;
    int n, m, s, t, k;
    cin >> n >> m >> s >> t >> k;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge.push_back({u, v, w});
    }
    auto ans = bellman_ford(n,m,s,t,k);
    if(ans >= 1e17) cout << "unreachable\n";
    else cout << ans << '\n';
    return 0;
}
```

## D. 图上最短路V

```
signed main()
{
    IOS;
    int n, m, t;
    cin >> n >> m >> t;
    vector<vector<i64>> dist(n + 1, vector<i64>(n + 1, 1e18));
    for(int i = 1; i <= n; i ++) dist[i][i] = 0;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        dist[u][v] = dist[v][u] = min(dist[u][v], (i64)w);
    }
    for(int k = 1; k <= n; k ++)
        for(int i = 1; i <= n; i ++)
            for(int j =1; j <= n; j ++)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
    while(t--)
    {
        int u, v;
        cin >> u >> v;
        if(dist[u][v] == 1e18) cout << -1 << '\n';
        else cout << dist[u][v] << '\n';
    }
    return 0;
}
```

## E. 热浪

```
vector<pii> edge[N];

i64 dijkstra(int n,int m,int s,int t)
{
    vector<i64> dist(n + 1, 1e18);
    vector<int> vis(n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    heap.push({0, s});
    dist[s] = 0;
    while(heap.size())
    {
        auto[d, u] = heap.top();
        heap.pop();
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto [v, w] : edge[u])
        {
            if(dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
    return dist[t] == 1e18 ? -1 : dist[t];
}


signed main()
{
    IOS;
    int n, m, s, t;
    cin >> n >> m >> s >> t;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    cout << dijkstra(n, m, s, t) << '\n';
    return 0;
}
```

## F. 负环

spfa算法

```
vector<pii> edge[N];

i64 spfa(int n,int m)
{
    vector<i64> dist(n + 1), cnt(n + 1);
    vector<int> vis(n + 1);
    queue<int> q;
    for(int i = 1; i <= n; i ++)
        q.push(i), vis[i] = 1;
    while(q.size())
    {
        auto u = q.front();
        q.pop();
        vis[u] = 0;
        for(auto [v, w] : edge[u])
        {
            if(dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                cnt[v] = cnt[u] + 1;
                if(cnt[v] >= n) return 1;
                if(!vis[v])
                {
                    q.push(v);
                    vis[v] = 1;
                }
            }
        }
    }
    return 0;
}


signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
    }
    if(spfa(n, m)) cout << "Yes\n";
    else cout << "No\n";
    return 0;
}
```

bellman-ford

```
vector<array<int, 3>> edge;

bool bellman_ford(int n,int m)
{
    vector<i64> dist(n + 1, 0);
    for(int i = 1; i <= n - 1; i ++)
    {
        auto last = dist;
        for(auto[u, v, w] : edge)
        {
            if(dist[v] > last[u] + w)
            {
                dist[v] = last[u] + w;
            }
        }
    }
    auto last = dist;
    for(auto[u, v, w] : edge)
    {
        if(dist[v] > last[u] + w)
        {
            return 1;
        }
    }
    return 0;
}

signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge.push_back({u, v, w});
    }
    if(bellman_ford(n, m)) cout << "Yes\n";
    else cout << "No\n";
    return 0;
}
```

## G. 信使

```
vector<pii> edge[N];

i64 dijkstra(int n,int m,int s)
{
    vector<i64> dist(n + 1, 1e18);
    vector<int> vis(n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    heap.push({0, s});
    dist[s] = 0;
    while(heap.size())
    {
        auto[d, u] = heap.top();
        heap.pop();
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto [v, w] : edge[u])
        {
            if(dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
    i64 ans = 0;
    for(int i = 1; i <= n; i ++) ans = max(ans, dist[i]);
    return (ans == 1e18 ? -1 : ans);
}


signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    cout << dijkstra(n, m, 1) << '\n';
    return 0;
}
```

## H. 开会

```
vector<pii> edge[N];

i64 dijkstra(int n,int m,int s)
{
    vector<i64> dist(n + 1, 1e18);
    vector<int> vis(n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    heap.push({0, s});
    dist[s] = 0;
    while(heap.size())
    {
        auto[d, u] = heap.top();
        heap.pop();
        if(vis[u]) continue;
        vis[u] = 1;
        for(auto [v, w] : edge[u])
        {
            if(dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
    i64 res = 0;
    for(int i = 1; i <= n; i ++) res += dist[i];
    return res;
}

signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    i64 ans = 1e18;
    for(int i = 1; i <= n; i ++)
        ans = min(ans, dijkstra(n, m, i));
    cout << ans << '\n';
    return 0;
}
```

## I. 最小花费

$x$ 向 $y$ 转账要扣除 $w\%$ 的手续费，可以以此建立一条边，从 $x$ 到 $y$ 长度为 $w$，那么在更新距离的时候，就有：

$dist[y] = \frac{dist[x]}{(1-w)}$

因此只需要从 $b$ 作为起点，设置 $dist[b] = 100$，反向跑一遍最短路，答案即为 $dist[a]$

```
vector<pair<int, double>> edge[N];

void solve()
{
    int n, m;
    cin >> n >> m;
    while (m--)
    {
        int u, v;
        double w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w / 100.0});
        edge[v].push_back({u, w / 100.0});
    }
    int a, b;
    cin >> a >> b;
    auto dijkstra = [&]() -> double
    {
        vector<double> dist(n + 10, 1e9), vis(n + 10);
        dist[b] = 1;
        priority_queue<pdi, vector<pdi>, greater<pdi>> heap;
        heap.push({1, b});
        while (heap.size())
        {
            auto u = heap.top().second;
            heap.pop();
            if (vis[u])
                continue;
            vis[u] = 1;
            for (auto [v, w] : edge[u])
            {
                if (dist[v] > dist[u] / (1 - w))
                    dist[v] = dist[u] / (1 - w);
                heap.push({dist[v], v});
            }
        }
        return dist[a];
    };
    cout << fixed << setprecision(8) << dijkstra() * 100.0;
}
```

## J. 昂贵的聘礼

我们可以把每一种用物品 v 去优惠 u 的操作看作是一条 v→u 的长度为优化价格的遍，同样的，我们假设我们现在手上什么物品都没有的状态位于第 0 号点，则对于每个物品其原价 p，我们都可以看作是原点向该物品连接了一条长度为 p 的边，那么问题最后就转化成从 0 点走到 1 点的最短路了。同时，题目中还存在交易等级的限制，我们注意到等级最多只有100个，且交易的等级范围限制 m 也只有1到100,那么我们可以考虑枚举1-m之间所有的等级范围，每次跑最短路的时候加上一个只有对方位于我们都交易等级限制内我们才选择去更新它的操作即可。

```
int level[N];

vector<pii> edge[N];

int dijkstra(int n,int m,int s,int t, int minlv,int maxlv)
{
    vector<int> dist(n + 1, 1e9), vis (n + 1);
    dist[s] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    heap.push({0, s});
    while(heap.size())
    {
        auto [d, u] = heap.top();
        heap.pop();
        if(vis[u]) continue;
        else vis[u] = 1;
        for(auto [v, w] : edge[u])
        {
            // 只有对方在等级限制内，才更新
            if(dist[v] > dist[u] + w and level[v] <= maxlv and level[v] >= minlv)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
    return dist[t];
}

signed main()
{
    IOS;
    int n, m;
    cin >> m >> n;
    for(int i = 1; i <= n; i ++)
    {
        int p,l,x;
        cin >> p >> l >> x;
        level[i] = l;
        edge[0].push_back({i, p});
        while(x--)
        {
            int u, w;
            cin >> u >> w;
            edge[u].push_back({i, w});
        }
    }
    int ans = 1e9;
    // 枚举等级区间
    for(int i = level[1] - m; i <= level[1]; i ++)
        ans = min(ans,dijkstra(n, m, 0, 1, i,i+m));
    cout << ans << '\n';
    return 0;
}
```

## K. 新年好

我们知道，每次dijkstra算法可以求出一个点到其他所有点的最短距离，而我们加上起点 1，一共只有6个点，我们可以先预处理出这6个点到其他点的最短距离，我们记为 $dist[i][j]$，表示从 $i$ 出发到 $j$ 的最短距离，这样，答案就只跟我们处理点的顺序有关了，此时我们再去枚举访问 $a,b,c,d,e$ 的顺序，我们记为 $order_i$

，对于每个顺序，他的距离就是:

$$dist[1][order[1]] + dist[order[1]][order[2]] + dist[order[2]][order[3]] + dist[order[3]][order[4]] + dist[order[4]][order[5]]$$

我们只需要枚举 $order_i$，然后求出最小值即可，时间复杂度为$O_{(6nlogn+5!)}$

```
vector<pii> edge[N];

void dijkstra(int n, int m, int s, vector<i64> &dist)
{
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    // vector<i64> dist(n + 1, 1e18);
    dist.assign(n + 1, 1e18);
    vector<int> vis(n + 1);
    dist[s] = 0;
    heap.push({0, s});
    while (heap.size())
    {
        auto [d, u] = heap.top();
        heap.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto [v, w] : edge[u])
        {
            if (dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
}

signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    array<int, 5> ar;
    for(int i = 0; i < 5; i ++) cin >> ar[i];
    while (m--)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    vector<vector<i64>> dist(6, vector<i64>(n + 1));
    dijkstra(n, m, 1, dist[0]);
    for(int i = 0; i < 5; i ++)
        dijkstra(n, m, ar[i], dist[i + 1]);
    vector<int> order = {0,1,2,3,4};
    i64 ans = 1e18;
    do{
        i64 now = dist[0][ar[order[0]]];
        for(int i = 0; i < 4; i ++)
        {
            now += dist[order[i] + 1][ar[order[i + 1]]];
        }
        ans = min(ans, now);
    }while(next_permutation(order.begin(), order.end()));
    cout << ans << '\n';
    return 0;
}
```

## L. 乘车路线

本题在基础单源最短路的题设下，终点变为唯一，但起点有多个，这种题我们一般有两个处理方法：

1.将所有起点连到同一个虚拟点，建一条长度为0的边，然后跑单源最短路即可

2.将所有的有向边反向，再从重点跑单源最短路，最后取所有起点dist的最小值即可

这里采用第一种方法：

```
vector<pii> edge[N];

void solve(int n, int m, int s)
{
    // 初始化图,因为有多测，所以每次都要清空
    for(int i = 1; i <= n; i ++) edge[i].clear();
    while (m--)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
    }
    int t;
    cin >> t;
    // 0 点作为超级原点，连接所有的起点，边权为0
    for(int i = 0; i < t; i ++)
    {
        int st;
        cin >> st;
        edge[0].push_back({st, 0});
    }
    vector<int> dist(n + 1, 1e9);
    // 从 0 开始跑dijkstra即可
    auto dijkstra = [&]()
    {
        dist[0] = 0;
        priority_queue<pii, vector<pii>, greater<pii>> heap;
        heap.push({0, 0});
        vector<bool> vis(n + 1);
        while (heap.size())
        {
            auto u = heap.top().second;
            heap.pop();
            if (vis[u])
                continue;
            else
                vis[u] = 1;
            for (auto[v, w] : edge[u])
            {
                if (dist[v] > dist[u] + w)
                {
                    dist[v] = dist[u] + w;
                    heap.push({dist[v], v});
                }
            }
        }
    };
    dijkstra();
    if (dist[s] == 1e9)
        cout << -1 << endl;
    else
        cout << dist[s] << endl;
}

signed main()
{
    int T;
    cin >> T;
    int n, m, p;
    while (T--)
    {
        cin >> n >> m >> p;
        solve(n, m, p);
    }
    return 0;
```

```
}
```

## M. 最短路计数

```
vector<int> edge[N];

void dijkstra(int n, int m, int s, vector<i64> &cnt)
{
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    vector<i64> dist(n + 1, 1e18);
    vector<int> vis(n + 1);
    dist[s] = 0;
    cnt[s] = 1;
    heap.push({0, s});
    while (heap.size())
    {
        auto [d, u] = heap.top();
        heap.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto v : edge[u])
        {
            if (dist[v] > dist[u] + 1)
            {
                dist[v] = dist[u] + 1;
                cnt[v] = cnt[u];
                heap.push({dist[v], v});
            }
            else if(dist[v] == dist[u] + 1)
            {
                cnt[v] += cnt[u];
                cnt[v] %= 100003;
            }
        }
    }
}

signed main()
{
    IOS;
    int n, m;
    cin >> n >> m;
    while (m--)
    {
        int u, v;
        cin >> u >> v;
        edge[u].push_back(v);
        edge[v].push_back(u);
    }
    vector<i64> cnt(n + 1);
    dijkstra(n, m, 1, cnt);
    for(int i = 1; i <= n; i ++) cout << cnt[i] << '\n';
    return 0;
}
```

## N. Orange走迷宫II

我们注意到，在网格图上走是不花费代价的，因此更新最短路时，如果是走格子，直接 $dist[v] = dist[u]$，同时，对于使用钻头的操作，我们花费 $w[i]$，可以更新一整列的格子，用循环暴力将这列格子更新即可。

```
int dx[] = {0,1,0,-1};
int dy[] = {1,0,-1,0};

struct node
{
    int x,y,dist;
    bool operator<(const node & _n) const
    {
        return dist > _n.dist;
    }
};

signed main()
{
    int n,m;
    cin >> n >> m;
    vector<string> graph(n);
    for(int i = 0; i < n; i ++)
        cin >> graph[i];
    vector<int> c(m);
    for(int i = 0; i < m; i ++)
        cin >> c[i];
    vector<vector<int>> vis(n, vector<int>(m));
    vector<vector<int>> dist(n, vector<int>(m, 1e18));
    dist[0][0] = 0;
    priority_queue<node> heap;
    heap.push({0,0,0});
    while(heap.size())
    {
        auto [x,y,d] = heap.top();
        heap.pop();
        if(vis[x][y]) continue;
        vis[x][y] = 1;
        for(int i = 0; i < 4; i ++)
        {
            int px = x + dx[i], py = y + dy[i];
            if(px < 0 || py < 0 || px >= n || py >= m) continue;
            if(graph[px][py] == '.')
            {
                if(dist[px][py] > dist[x][y])
                {
                    dist[px][py] = dist[x][y];
                    heap.push({px, py, dist[px][py]});
                }
            }
        }
        for(int i = -1; i < 2; i ++)
        {
            int py = y + i;
            if(py < 0 || py >= m) continue;
            for(int j = 0; j < n; j ++)
            {
                int px = j;
                if(dist[px][py] > dist[x][y] + c[py])
                {
                    dist[px][py] = dist[x][y] + c[py];
                    heap.push({px, py, dist[px][py]});
                }
            }
        }
    }
    cout << dist[n-1][m-1];
    return 0;
}
```

## O. 飞行路线

dp做法

```
using piii = array<i64, 3>;

vector<pii> edge[N];

i64 dijkstra(int n,int m,int s,int t,int k)
{
    vector<vector<i64>> dist(k + 1, vector<i64>(n + 1,1e18));
    vector<vector<int>> vis(k + 1, vector<int>(n + 1));
    priority_queue<piii, vector<piii>, greater<piii>> heap;
    heap.push({0, s, 0});
    dist[0][s] = 0;
    while(heap.size())
    {
        auto[d, u, p] = heap.top();
        heap.pop();
        if(vis[p][u]) continue;
        vis[p][u] = 1;
        for(auto [v, w] : edge[u])
        {
            if(dist[p][v] > dist[p][u] + w)
            {
                dist[p][v] = dist[p][u] + w;
                heap.push({dist[p][v], v, p});
            }
            if(p < k and dist[p + 1][v] > dist[p][u])
            {
                dist[p + 1][v] = dist[p][u];
                heap.push({dist[p + 1][v], v, p + 1});
            }
        }
    }
    i64 ans = 1e18;
    for(int i = 0; i <= k; i ++)
        ans = min(ans, dist[i][t]);
    return ans;
}


signed main()
{
    IOS;
    int n, m, k, s, t;
    cin >> n >> m >> k >> s >> t;
    while(m --)
    {
        int u, v, w;
        cin >> u >> v >> w;
        edge[u].push_back({v, w});
        edge[v].push_back({u, w});
    }
    cout << dijkstra(n, m, s, t, k) << '\n';
    return 0;
}
```

分层图做法

```
vector<pii> edge[N];

i64 dijkstra(int n, int m, int s, int t, int k)
{
    vector<i64> dist((k + 1) * n + 1, 1e18);
    vector<int> vis((k + 1) * n + 1);
    priority_queue<pii, vector<pii>, greater<pii>> heap;
    heap.push({0, s});
    dist[s] = 0;
    while (heap.size())
    {
        auto [d, u] = heap.top();
        heap.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto [v, w] : edge[u])
        {
            if (dist[v] > dist[u] + w)
            {
                dist[v] = dist[u] + w;
                heap.push({dist[v], v});
            }
        }
    }
    i64 ans = 1e18;
    for (int i = 0; i <= k; i++)
        ans = min(ans, dist[t + i * n]);
    return ans;
}

signed main()
{
    IOS;
    int n, m, k, s, t;
    cin >> n >> m >> k >> s >> t;
    while (m--)
    {
        int u, v, w;
        cin >> u >> v >> w;
        for (int i = 0; i <= k; i++)
        {
            edge[u + i * n].push_back({v + i * n, w});
            edge[v + i * n].push_back({u + i * n, w});
            if (i < k)
            {
                edge[u + i * n].push_back({v + (i + 1) * n, 0});
                edge[v + i * n].push_back({u + (i + 1) * n, 0});
            }
        }
    }
    auto ans = dijkstra(n, m, s, t, k);
    cout << ans << '\n';
    return 0;
}
```