# 拆点

拆点是一种图论建模思想，常用于网络流，用来处理 **点权或者点的流量限制** 的问题，也常用于 **分层图**

## 结点有流量限制的最大流

如果把结点转化成边，那么这个问题就可以套板子解决了。

我们考虑把有流量限制的结点转化成这样一种形式：由两个结点 $u,v$ 和一条边 $<u,v>$ 组成的部分。其中，结点 $u$ 承接所有从原图上其他点的出发到原图上该点的边，结点 $v$ 引出所有从原图上该点出发到达原图上其他点的边。边 $<u,v>$ 的流量限制为原图该点的流量限制，再套板子就可以解决本题。这就是拆点的基本思想。

如果原图是这样：

拆点之后的图是这个样子：

## 分层图最短路

分层图最短路，如：有 $k$ 次零代价通过一条路径，求总的最小花费。对于这种题目，我们可以采用 DP 相关的思想，设 $\text{dis}_{i,j}$ 表示当前从起点到 $i$ 号结点，使用了 $j$ 次免费通行权限后的最短路径。显然 $\text{dis}$ 数组可以这么转移 $$ \text{dis}_{i,j}=\min\{\min\{\text{dis}_{from,j-1}\},\min\{\text{dis}_{from,j}+w\}\} $$ 其中 $from$ 表示 $i$ 的父亲结点 $w$ 表示当前所走的边的边权。当 $j-1\ge k$ 时，$\text{dis}_{from,j}=\infin$

事实上，这个 DP 就相当于把每个结点拆分成了 $k+1$ 个结点，每个新结点代表使用不同多次免费通行后到达的原图结点。换句话说，就是每个结点 $u_i$ 表示使用 $i$ 次免费通行权限后到达 $u$ 结点。

> LOI2011 飞行路线
>
> 题意：有一个 $n$ 个点 $m$ 条边的无向图，你可以选择 $k$ 条道路以零代价通行，求 $s$ 到 $t$ 的最小花费。

参考核心代码：

```
struct State {     //优先队列的结点结构体
  int v, w, cnt;  // cnt 表示已经使用多少次免费通行权限
  State() {}
  State(int v, int w, int cnt) : v(v), w(w), cnt(cnt) {}
  bool operator<(const State &rhs) const { return w > rhs.w; }
};

void dijkstra() {
```

```
  memset(dis, 0x3f, sizeof dis);
  dis[s][0] = 0;
  pq.push(State(s, 0, 0));  //到起点不需要使用免费通行权，距离为零
  while (!pq.empty()) {
    const State top = pq.top();
    pq.pop();
    int u = top.v, nowCnt = top.cnt;
    if (done[u][nowCnt]) continue;
    done[u][nowCnt] = true;
    for (int i = head[u]; i; i = edge[i].next) {
      int v = edge[i].v, w = edge[i].w;
      if (nowCnt < k && dis[v][nowCnt + 1] > dis[u][nowCnt]) {  //可以免费通行
        dis[v][nowCnt + 1] = dis[u][nowCnt];
        pq.push(State(v, dis[v][nowCnt + 1], nowCnt + 1));
      }
      if (dis[v][nowCnt] > dis[u][nowCnt] + w) {  //不可以免费通行
        dis[v][nowCnt] = dis[u][nowCnt] + w;
        pq.push(State(v, dis[v][nowCnt], nowCnt));
      }
    }
  }
}

int main() {
  n = read(), m = read(), k = read();
  s = read() + 1, t = read() + 1;
  while (m--) {
    int u = read() + 1, v = read() + 1, w = read();
    add(u, v, w), add(v, u, w);  //双向边
  }
  dijkstra();
  int ans = std::numeric_limits<int>::max();  // ans 取int 最大值为初值
  for (int i = 0; i <= k; ++i)
    ans = std::min(ans, dis[t][i]);  //对到达终点的所有情况取最优值
  printf(ans);
}
```

# 参考链接

OI Wiki