期末大作业解题报告

本次报告撰写人：张晨阳

题目：202012-4、202303-4、Codeforces 1786F

题目 1：202012-4 食材运输

题目描述：

N 个酒店，由 $N-1$ 条双向道路构成一棵树，不同道路对应不同通行时间。

K 种食材对应 K 辆车，M 个检查点作为车辆起点。

求所有酒店的等待时间的最大值的最小值。

输入格式：

第一行 N, M, K；接下来 N 行，每行 K 个数（1 表示需要对应食材，0 表示不需要）；接下来 $N-1$ 行，每行 3 个整数 u, v, w（通行时间为 w 的双向道路连接 u 和 v）。

输出格式：

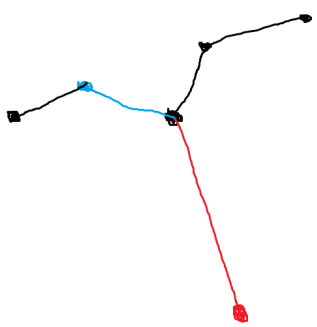
输出一个整数，表示在你的方案中，所有酒店的等待时间的最大值。

$N \leq 10^2, M \leq K \leq 10.$

题解：

主要思路：先预处理所有数据，然后进行动态规划，最后二分查找来尝试不同的时间（通过时间选择方案）

1. 存储所有数据：
   用位图表示来处理各个酒店所需要的食材，用 01 表示各食材的需求情况：第 i 位表示对第 i 号食材的需求（1 表示需要，0 则不需要），从而将所有食材压缩到一个整数上，该整数表示该点对所有食材的需求情况。
   对于每个节点存储的信息（指向酒店 v，与 v 之间道路时长 w），通过 $vector$ 结构进行添加信息，因为是双向道路，所以需要同时对 u，v 都进行添加。
2. $intitial$ 函数部分：
   计算从酒店 i 出发，配送食材 j 的最大等待时间。
   在进行 dfs 之前，用 $bool$ 数组标记出哪些酒店需要该食材；需要注意的是，在 dfs 中，某些必经点（下图的蓝色节点）也要标记出来。（每个标记循环前需要将该数组置零）
   dfs 部分：提供当前酒店号 u，和来源酒店号 pre。（当前节点和上一个节点）
   求解最大等待时间算法解释：
   其实只分为两种情况：



2.1. 该节点 → 不需要食材的节点（蓝色路径） → 需要食材的节点（黑色路径）

2.2. 该节点 $\rightarrow$ 需要食材的节点（黑色路径）

则总路径应为其中一条子路径走一次，其余子路径去并返回。即 $maxtime = onetime * 2 - maxn$，其中 $onetime$ 表示全部的单程路径，$maxn$ 表示途中红色的最长的路径。

3. $work$ 函数部分：

3.1. $M == K$

先遍历各食材最小解，从最小解里选出最大值。

3.2. $M < K$

备忘录 dp+二分查找

用位图标记某酒店是否适合配送每一个食材（时长<mid 表示适合）

dp 方程：

第 j 个检查点的配送情况=第 $j-1$ 个检查点情况叠加酒店 i 的配送情况

提交编号	用户名	姓名	试题名称	提交时间	代码长度	编程语言	评测结果	得分	时间使用	空间使用
3757273	张晨阳	张晨阳	食材运输	06-21 17:55	3.035KB	CPP14	正确	100	15ms	2.972MB

C++

```cpp
#include<iostream>
#include<vector>
#include<algorithm>
#include<string.h>
using namespace std;
int N, M, K;
int foodbit[101];                       //位图表示
int waitmax[101][11];                   //记录从酒店i出发配送食材j的最大等待时间
bool passneed[101];                     //标记是否需要经过该酒店
int onetime;                            //配送所有酒店的单程时间
int maxnn = 0, minnn = 0;               //记录二分时的左右边界
bool dp[11][1 << 10];                   //表示是否可送达K种食材的配送组合
int flag[101];                          //位图标记某酒店是否适合配送每一个
食材
struct node {
        int v, w;
        node(int v, int w):v(v),w(w){}
};
vector<node> map[101];

int dfs(int u, int pre) {
        int ret = 0;
        //深度优先遍历u为顶点的所有路径
        for (const auto& x : map[u]) {
                int v = x.v;
                int w = x.w;
                if (v == pre)
                        continue;
                int re = dfs(v, u);
                //处理必经点（不需要食材但子节点需要且必经过该节点）
                if (passneed[v]) {
                        passneed[u] = true;
                        onetime += w;
                        ret = max(ret, re + w);
                }
        }
        return ret;
}

void initial() {
        for (int i = 1; i <= N; i++)
                for (int j = 0; j < K; j++) {
                        //事先标记
                        memset(passneed, 0, sizeof(passneed));
                        for (int k = 1; k <= N; k++)
```

```
                                if (foodbit[k] >> j & 1)
                                        passneed[k] = true;
                        //初始化单程时间
                        onetime = 0;
                        //从第i个酒店开始的情况下，找到最长的路径
                        int maxn = dfs(i, -1);
                        //其他路径去并返回，最长路径只走一次不返回
                        waitmax[i][j] = onetime * 2 - maxn;
                        //为二分查找的边界做准备
                        if (waitmax[i][j] > maxnn)
                                maxnn = waitmax[i][j];
                        else if (waitmax[i][j] < minnn)
                                minnn = waitmax[i][j];
                }
}

bool judge(int mid) {
        memset(flag, 0, sizeof(flag));
        memset(dp, 0, sizeof(dp));
        //位图标记酒店i是否适合配送食材j
        for (int i = 1; i <= N; i++)
                for (int j = 0; j < K; j++)
                        if (waitmax[i][j] <= mid)
                                flag[i] |= 1 << j;

        dp[0][0] = true;
        //遍历酒店，以酒店i为出发点
        for (int i = 1; i <= N; i++) {
                //遍历检查点，酒店i在j个检查点时的配送情况
                for (int j = 1; j <= M; j++)
                        for (int k = 0; k < (1 << K); k++)
                                dp[j][k | flag[i]] |= dp[j - 1][k];
                if (dp[M][(1 << K) - 1])
                        return true;
        }
        return dp[M][(1 << K) - 1];
}

void work() {
        if (M == K) {
                int ansmax = -1;
                for (int j = 0; j < K; j++) {
                        int foodmin = maxnn;
                        for (int i = 1; i <= N; i++)
                                if (waitmax[i][j] < foodmin)
                                        foodmin = waitmax[i][j];
```

```
                        ansmax = max(ansmax, foodmin);
                }
                cout << ansmax << "\n";
                return;
        }
        int l = minnn, r = maxnn, ans = 0;
        while (l <= r) {
                int mid = (l + r) / 2;
                if (judge(mid)) {
                        r = mid - 1;
                        ans = mid;
                }
                else
                        l = mid + 1;
        }
        cout << ans << "\n";
        return;
}

int main()
{
        ios::sync_with_stdio(false);
        //输入
        cin >> N >> M >> K;
        for (int i = 1; i <= N; i++)
                for (int j = 0; j < K; j++) {
                        int temp;
                        cin >> temp;
                        if (temp)
                                foodbit[i] |= (1 << j);
                }
        for (int i = 1; i < N; i++) {
                int u, v, w;
                cin >> u >> v >> w;
                map[u].emplace_back(v, w);
                map[v].emplace_back(u, w);
        }
        //预处理
        initial();
        //计算结果
        work();
        return 0;
}
```

题目 2：202303-4 星际网络 II

题目描述：

地址分配采用类似 IPv6的十六进制表示法，每 4 位用":"隔开。地址长度 n 是 16 的倍数，每个地址由 n 位二进制位组成。
三种操作：

- `1 id l r`：表示用户 id 申请地址在 $l \sim r$ 范围内（包含 l 和 r）的一段连续地址块。
  若申请的地址全部未被分配，则检查通过；若地址中存在已分配给其他用户的地址，则检查失败。
  特殊情况：申请地址中没有已分配给其他用户的地址，但含有已分配给该用户的地址。此时认为检查通过，但若申请的地址先前已全部分配给该用户则检查失败。
  如果检查通过，则返回 `YES`，并将申请地址分配给该用户；若不通过，返回 `NO`，不改变现有地址分配。
- `2 s`：检查地址 s 被分配给了哪个用户。若未被分配，则结果为 0。
- `3 l r`：检查 $l \sim r$ 范围内的所有地址是否完整分配给了某用户。若是，回答用户编号；若否，回答 0。
  输入格式：
  第一行，2 个正整数 n, q。接下来 q 行，每行一个操作，格式如上所述。
  输出格式：
  输出 q 行，每行一个非负整数或字符串，表示此次操作的结果。
  $n \leq 512, id \leq q \leq 5 \times 10^4$

题解：

线段树。

1. 存储数据：
   定义一个结构体 `node`，用来处理每个操作的信息：`op` 表示操作类型，`id` 表示用户编号，`l, r` 表示左边界和右边界。
   1.1. 对于操作 1：
   将左右边界分别存到一个数组里，并计算右边界+1 后的地址，也存到数组里（将地址范围变成一个左闭右开的区间，方便进行地址分配和范围的比较）。
   1.2. 对于操作 2：仅读取地址并将其添加到数组。
   1.3. 对于操作 3：读取左边界和右边界，并将它们和右边界的下一个地址添加到数组。
2. 处理数据：
   最大值和最小值存储地址分配 `id` 的最大值和最小值，和值存储当前地址块已经有多少地址被分配。
   2.1. 对数组进行排序并删去重复的元素。
   2.2. 构造线段树：`build` 函数使用递归方式构建线段树。初始化每个节点的属性，将范围分为两半，并在每一半上调用自身以构建子节点。最后，调用 `pushup` 函数更新当前节点的值。
   `build` 函数三个参数：`id` 表示当前节点，`L` 表示当前节点区间的左边界，`R` 表示当前节点区间的右边界。首先为当前节点设置左边界 `l[id]`、右边界 `r[id]`、最小值 `mn[id]` 和最大值 `mx[id]` 初始值。如果当前节点的左边界大于等于右边界（即区间只有一个元素），则直接返回，当前节点为叶子节点。否则，计算当前区间的中间位置 `mid`，然后递归地构建左子树和右子树。左子树的索引为 `id << 1`；右子树的索引为 `id << 1 | 1`。
   `pushup` 函数利用左子节点和右子节点的索引访问对应的子节点的值。然后，通过 `max` 函数更新当前节点的最大值，通过 `min` 函数更新当前节点的最小值，通过加法操作更新当前节点的和值。

3. 进行操作 1：
   先查询该区间内地址编号的最小值，如果最小值是初始值，说明没有进行分配，直接全部赋值为 `id`。如果发现最小值不是初始值，查最大值，如果最小值和最大值不同，说明这块地址多人分配，无需操作；如果最小值和最大值相同，那么地址分配给了一个人，然后用区间和来查询是否全部地址都分配给了这个人，如果区间和等于区间长度，说明这段地址空间全部分配给了一个人，否则判断所查询值与待分配编号是否相同。
   3.1. 先通过 `find` 函数查询左右边界在数组中的位置。**(使用 `lower_bound` 函数查找，时间复杂度 $O(log\ n)$，若直接遍历查找会超时)**
   3.2. 查询当前区间地址是否被分配：
      3.2.1. `judgemin(int id, int L, int R)` 函数：如果当前节点的区间完全包含在目标区间内，则直接返回当前节点的最小值。否则，调用 `pushdown` 函数将延迟标记向下传递，并通过递归调用查询左右子节点的最小值，并返回其中的较小值。
      3.2.2. `pushdown(int id)`：如果当前节点的延迟标记 `lazy[id]` 不等于预设值，则将节点的子节点进行更新。最后将当前节点的延迟标记重新设置为预设值。
   3.3. 如果未被分配，输出 `YES` 并调用 `update` 函数，将该区间内的节点信息更新。
   3.4. 如果只分配给了一个人
      3.4.1. 通过函数 `judgesum` 判断是否全部分配给了当前 `id`。
      首先，检查当前节点是否完全位于查询区间内，如果是，则直接返回该节点的和值 `sum[id]`。否则，调用 `pushdown` 函数将当前节点的懒惰标记进行推送，然后计算当前节点的中点 `mid`。接着，根据 `mid` 和查询区间的关系，递归调用 `judgesum` 函数在左子节点或右子节点上进行查询，并将两者的结果累加到 `ans` 变量中。最后，返回 `ans`。
      3.4.2. 如果全部分配，则输出 `NO`，否则输出 `YES` 并更新信息。
   3.5. 如果分配给其他人，输出 `NO`
4. 进行操作 2：
   单点查询。
5. 进行操作 3；
   与操作 1 类似。

提交编号	用户名	姓名	试题名称	提交时间	代码长度	编程语言	评测结果	得分	时间使用	空间使用
3757405	张晨阳	张晨阳	星际网络II	06-22 02:58	5.476KB	CPP14	正确	100	437ms	162.4MB

C++

```cpp
#include<iostream>
#include<map>
#include<vector>
#include<string>
#include<algorithm>
#define ls(o) (o << 1)
#define rs(o) (ls(o) | 1)
#define ini 0x3f3f3f3f
using namespace std;

const int N = 1e6 + 10;
int n, q;
vector<string> saveadds;
struct node {
        int op, id;
        string l, r;
}act[N];
int l[N], r[N], mx[N], mn[N], lazy[N], sum[N];

void pushup(int id) {
    mx[id] = max(mx[ls(id)], mx[rs(id)]);
    mn[id] = min(mn[ls(id)], mn[rs(id)]);
    sum[id] = sum[ls(id)] + sum[rs(id)];
}
//建树
void build(int id, int L, int R) {
    l[id] = L;
    r[id] = R;
    mn[id] = ini;
    mx[id] = -ini;
    lazy[id] = ini;
    if (L >= R)
        return;
    int mid = L + R >> 1;
    build(ls(id), L, mid);
    build(rs(id), mid + 1, R);
    pushup(id);
}

void pushdown(int id) {
    if (lazy[id] != ini) {
        sum[ls(id)] = r[ls(id)] - l[ls(id)] + 1;
        sum[rs(id)] = r[rs(id)] - l[rs(id)] + 1;
        mx[ls(id)] = lazy[id];
        mx[rs(id)] = lazy[id];
```

```
            mn[ls(id)] = lazy[id];
            mn[rs(id)] = lazy[id];
            lazy[ls(id)] = lazy[id];
            lazy[rs(id)] = lazy[id];
            lazy[id] = ini;
        }
    }
    //查找数组中出现s的位置
    int find(string s) {
        return lower_bound(saveadds.begin(), saveadds.end(), s) -
    saveadds.begin() + 1;
    }

    int judgemin(int id, int L, int R) {
        if (l[id] >= L && r[id] <= R)
            return mn[id];
        pushdown(id);
        int mid = l[id] + r[id] >> 1;
        int ans = ini;
        if (mid >= L)
            ans = judgemin(ls(id), L, R);
        if (mid + 1 <= R)
            ans = min(ans, judgemin(rs(id), L, R));
        return ans;
    }
    //更新节点信息
    void update(int id, int L, int R, int val) {
        if (l[id] >= L && r[id] <= R) {
            sum[id] = r[id] - l[id] + 1;
            mx[id] = val;
            mn[id] = val;
            lazy[id] = val;
            return;
        }
        pushdown(id);
        int mid = l[id] + r[id] >> 1;
        if (mid >= L)
            update(ls(id), L, R, val);
        if (mid + 1 <= R)
            update(rs(id), L, R, val);
        pushup(id);
    }
    //判断该地址是否已经全部分配给id
    int judgesum(int id, int L, int R) {
        if (l[id] >= L && r[id] <= R)
            return sum[id];
```

```
    pushdown(id);
    int mid = l[id] + r[id] >> 1;
    long long ans = 0;
    if (mid >= L)
        ans += judgesum(ls(id), L, R);
    if (mid + 1 <= R)
        ans += judgesum(rs(id), L, R);
    return ans;
}

int judgemax(int id, int L, int R) {
    if (l[id] >= L && r[id] <= R)
        return mx[id];
    pushdown(id);
    int mid = l[id] + r[id] >> 1;
    int ans = -ini;
    if (mid >= L)
        ans = judgemax(ls(id), L, R);
    if (mid + 1 <= R)
        ans = max(ans, judgemax(rs(id), L, R));
    return ans;
}
//计算右边界的下一个地址
string next(string s) {
    string result = s;
    bool carry = true;
    for (int i = s.size() - 1; i >= 0; i--) {
        if (s[i] == ':')
            continue;
        else if (s[i] == 'f' && carry)
            result[i] = '0';
        else if (s[i] == '9') {
            result[i] = 'a';
            carry = false;
            break;
        }
        else {
            result[i] = s[i] + 1;
            carry = false;
            break;
        }
    }
    return result;
}

int main()
```

```
{
        ios::sync_with_stdio(false);
        cin >> n >> q;
        for (int i = 1; i <= q; i++) {
                cin >> act[i].op;
                if (act[i].op == 1) {
                        cin >> act[i].id >> act[i].l >> act[i].r;
                        saveadds.push_back(act[i].l);
                        saveadds.push_back(act[i].r);
                        saveadds.push_back(next(act[i].r));
                }
        else if (act[i].op == 2) {
            cin >> act[i].l;
            saveadds.push_back(act[i].l);
        }
        else {
            cin >> act[i].l >> act[i].r;
            saveadds.push_back(act[i].l);
            saveadds.push_back(act[i].r);
            saveadds.push_back(next(act[i].r));
        }
        }
    sort(saveadds.begin(), saveadds.end());
    //去除数组中的重复元素
    vector<string> unique_values;  // 用于存储去重后的唯一元素
    for (int i = 0; i < saveadds.size(); i++)
        if (i == 0 || saveadds[i] != saveadds[i - 1])
            unique_values.push_back(saveadds[i]);
    saveadds = unique_values;  // 将原容器替换为去重后的结果
    //建树
    build(1, 1, saveadds.size());
    for (int i = 1; i <= q; i++) {
        if (act[i].op == 1) {
            int ll = find(act[i].l), rr = find(act[i].r);
            //该地址全部未被分配
            if (judgemin(1, ll, rr) == ini) {
                cout << "YES\n";
                update(1, ll, rr, act[i].id);
            }
            //该地址是否只分配给了一个人
            else if (judgemin(1, ll, rr) == act[i].id && judgemax(1, ll,
rr) == act[i].id) {
                //该地址本来就已经全部分配给了id
                if (judgesum(1, ll, rr) == (rr - ll + 1))
                    cout << "NO\n";
                else {
```

```
                    cout << "YES\n";
                    update(1, ll, rr, act[i].id);
                }
            }
            //该地址分配给了其他人
            else
                cout << "NO\n";
        }
        else if (act[i].op == 2) {
            int ll = find(act[i].l);
            int t = judgemax(1, ll, ll);
            if (t != -ini)
                cout << t << "\n";
            else
                cout << "0\n";
        }
        else {
            int ll = find(act[i].l), rr = find(act[i].r);
            int id = judgemin(1, ll, rr);
            //该地址是否只分配给了一个人
            if (id == judgemax(1, ll, rr) && judgesum(1, ll, rr) == (rr
- ll + 1))
                cout << id << "\n";
            else
                cout << "0\n";
        }
    }
    return 0;
}
```

题目 3：Codeforces 1786F Wooden Spoon

题目描述：

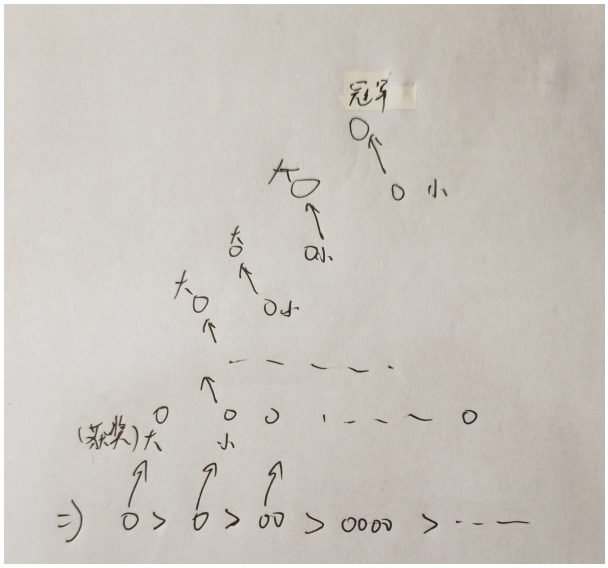
编号从 1 到 2^n 的参赛选手，以高为 n，叶子数为 2^n 的结构比赛。
当两个选手比赛时，编号较小的获胜。
"Wooden Spoon"颁给满足一下条件的选手（有且仅有一位）：

- 输了第一局比赛
- 第一局比赛的对手输了他的第二局
- 上一条中获胜的选手输了他的第三局
- ……
- 满足前面条件的选手输掉了决赛
  共 $(2^n)!$ 种选手对阵排列，计算出每位选手有多少种排列可以使其获得"Wooden Spoon"
  输入格式：

只有一个 n
输出格式：
2^n 行，第 i 行输出选手 i 可得奖的排列种数（取模 998244353）。
$1 \leq n \leq 20$

题解：

首先不考虑获奖选手的叶子位置（因为任何位置都可通过反转左右叶子得到，只需最后乘 2^n 即可），所以先假设获奖选手在最左边的节点。



对于最下面的每个集合，都是其中的编号最小者向上继续比赛。对于每两个相邻的集合，令左边最小值为 $min1$，右边最小值为 $min2$，则区间 $(min2, min1)$ 之间的数，只能放在左边集合（包括 $min2$ 集合）中。

排序方法：从小到大填入选手，从右集合向左集合填。

设 `dp[i][j]` 表示后 i 个集合中剩下 j 个空位的方案数。初始化 `dp[0][0]=1`。

则

$$dp[i][j] \times j = dp[i][j-1]$$

从右向左看，第一个集合有 2^{n-1} 个位置，第 i 个集合有 2^{n-i} 个位置，则

$$dp[i+1][j+2^{n-i-1}-1] = dp[i][j] \times 2^{n-i-1}$$

$dp[n][2^n-k]$ 表示排完了所有集合，其中选手 k 排在最左边（获奖者），那么还有 2^n-k 个大于他的选手排在后面剩余的空位中（因为序号大于 k，所有不影响他获奖，只需考虑他们的位置顺序）。

则最后第 k 行结果为 dp 结果×k 的位置×剩下选手排列顺序：

$$dp[n][2^n-k] \times 2^n \times (2^n-k)!$$

为便于计算，提前将阶乘结果存到数组 `fac` 里。

My Submissions

#	When	Who	Problem	Lang	Verdict	Time	Memory
210642027	Jun/22/2023 18:11 UTC+8	SevenGrasszcy	F - Wooden Spoon	GNU C++14	Accepted	967 ms	213400 KB

C++

```cpp
#include<bits/stdc++.h>
#define mod 998244353
using namespace std;
const int N = 1 << 21;
int n, dp[25][N], fac[N];
int main()
{
	ios::sync_with_stdio(false);
	cin >> n;
	fac[0] = 1;
	for (int i = 1; i <= (1 << n); i++)
		fac[i] = 1ll * fac[i - 1] * i% mod;
	dp[0][0] = 1;
	for(int i = 0; i <= n; i++)
		for (int j = (1 << n) - (1 << n - i); j >= 0; j--) {
			if (i < n)
				dp[i + 1][j + (1 << n - i - 1) - 1] +=
1ll * dp[i][j] % mod * (1 << n - i - 1) % mod;
			if (j)
				dp[i][j - 1] += 1ll * dp[i][j] % mod * j
% mod;
		}
	int times = 1 << n;
	for (int k = 1; k <= 1 << n; k++)
		cout << 1ll * dp[n][(1 << n) - k] * fac[(1 << n) - k] %
mod * times % mod << "\n";
	return 0;
}
```