ICPC Template Manual

作者: 贺梦杰

August 7, 2019

Contents

Chapter 1

基础

1.1 测试

Chapter 2

搜索

Chapter 3

动态规划

Chapter 4

字符串

4.1 KMP

```
int *Get_next(string str)
{
    int *ptr = new int[str.length()];
    //申请next数组
    ptr[0] = 0;             //首位next值为0
    int i = 1;              //初始化
    int j = 0;              //初始化
    int len = str.length(); //模式串长度
    while (i < len)
    {
        if (str[i] == str[j])
        {
            ptr[i] = j + 1;
            j++;
            i++; //确定前缀后缀相同的长度
        }
        else
        {
            //不同时
            if (j != 0)
                j = ptr[j - 1]; //j回到前一个字符的next值位置
            else
            {
                ptr[i] = 0; //回到模式串的第一个字符
                i++;
            }
        }
    }
    return ptr;
}
int KMP(string s, string p)
{
    int *next = Get_next(p);
    //获得next数组
    int i = 0;
    int j = 0;
    int len = s.length();
    while (i < len)
    {
        if (s[i] == p[j])
        {
            i++;
            j++; //匹配
            if (j >= p.length())
                return i - j;
        }
        else
        {
            //字符不相同回到前一个字符的next值位置
            if (j != 0)
                j = next[j - 1];
```

```
            else
                i++;
        }
    }
    return -1;
}
```

来源：*https://www.bilibili.com/video/av47471886?from=search&seid=4651914725266859344*

Chapter 5

数据结构

5.1 并查集

```
#define MAX 1010
struct node
{
    int par;
    //int rank;
    //路径压缩后 rank=1或2 rank失去了意义
    int data;
};
node ns[MAX];
void Init()
{
    for (int i = 1; i < MAX; i++)
    {
        ns[i].par = i;
    }
}
int Find(int i)
{
    if (ns[i].par == i)
    {
        //返回根结点
        return i;
    }
    ns[i].par = Find(ns[i].par);
    //路径压缩
    return ns[i].par;
}
void Union(int i, int j)
{
    int pi = Find(i);
    int pj = Find(j);
    if (pi != pj)
    {
        ns[pi].par = pj;
    }
}
```

5.2 线段树

5.2.1 基础操作

```
const int N = 1e5 + 10;
#define ls(a) (a << 1)
#define rs(a) (a << 1 | 1)
struct node
{
    int val;
    int lazy;
};
node tree[N << 2];
int a[N];
void PushUp(int rt)
{
    tree[rt].val = tree[ls(rt)].val + tree[rs(rt)];
}
void PushDown(int ls, int rs, int rt)
{
    tree[ls(rt)].val += ls * tree[rt].lazy;
    tree[rs(rt)].val += rs * tree[rt].lazy;
    tree[ls(rt)].lazy += tree[rt].lazy;
    tree[rs(rt)].lazy += tree[rt].lazy;
    tree[rt].lazy = 0;
}
void Build(int left, int right, int rt)
{
    if (left == right)
    {
        tree[rt].val = a[left];
        return;
    }
    int mid = (left + right) >> 1;
    Build(left, mid, ls(rt));
    Build(mid + 1, right, rs(rt));
    PushUp(rt);
    //向上更新
}
```

5.2.2 单点更新

```
void Update(int left, int right, int rt, int pos, int val)
{
    if (left == right && left == pos)
    {
        tree[rt].val += val;
        return;
    }
    int mid = (left + right) >> 1;
    if (tree[rt].lazy)
    {
```

```
        PushDown(mid - left + 1, right - mid, rt);
    }
    if (mid >= pos)
        Update(left, mid, ls(rt), pos, val);
    else if (pos > mid)
        Update(mid + 1, right, rs(rt), pos, val);
    PushUp(rt);
}
```

例题：*https://www.luogu.org/problemnew/show/P3372*

5.2.3 区间更新

```
void Update(int left, int right, int rt, int s, int t, int val)
{
    if (left >= s && right <= t)
    {
        tree[rt].val += (right - left + 1) * val;
        tree[rt].lazy += val;
        return;
    }
    int mid = (left + right) >> 1;
    if (tree[rt].lazy)
    {
        PushDown(mid - left + 1, right - mid, rt);
    }
    if (mid < s)
        Update(mid + 1, right, rs(rt), s, t, val);
    else if (mid >= t)
        Update(left, mid, ls(rt), s, t, val);
    else
    {
        Update(left, mid, ls(rt), s, t, val);
        Update(mid + 1, right, rs(rt), s, t, val);
    }
    PushUp(rt);
}
```

5.2.4 区间查询

```
void Query(int left, int right, int s, int t, int rt)
{
    if (left >= s && right <= t)
    {
        return tree[rt].val;
    }
    int mid = (left + right) >> 1;
    if (tree[rt].lazy)
        PushDown(mid - left + 1, right - mid, rt);
    long long sum = 0;
    if (mid < s)
        sum += Query(mid + 1, right, rs(rt), s, t, val);
    else if (mid >= t)
```

```
        sum += Query(left, mid, ls(rt), s, t, val);
    else
    {
        sum += Query(left, mid, ls(rt), s, t, val);
        sum += Query(mid + 1, right, rs(rt), s, t, val);
    }
    return sum;
}
```

例题：*https://www.luogu.org/problemnew/show/P3373*

5.3 树状数组

推荐阅读：*https://www.cnblogs.com/RabbitHu/p/BIT.html*

5.3.1 单点修改，区间查询

```
#define N 1000100
long long c[N];
int n,q;
int lowbit(int x)
{
    return x&(-x);
}
void change(int x,int v)
{
    while(x<=n)
    {
        c[x]+=v;
        x+=lowbit(x);
    }
}
long long getsum(int x)
{
    long long ans=0;
    while(x>=1)
    {
        ans+=c[x];
        x-=lowbit(x);
    }
    return ans;
}
```

例题：*https://loj.ac/problem/130*

5.3.2 区间修改，单点查询

引入差分数组来解决树状数组的区间更新

```
//初始化
change(i,cur-pre);
//区间修改
change(l,x);
change(r+1,-x);
//单点查询
getsum(x)
```

例题：*https://loj.ac/problem/131*

5.3.3 区间修改，区间查询

```
//初始化
change(c1,i,cur-pre);
change(c2,i,i*(cur-pre));
//为什么这么写？ 你需要写一下前缀和的表达式
```

```
//区间修改
change(c1,l,x);
change(c2,l,l*x);
change(c1,r+1,-x);
change(c2,r+1,-(r+1)*x);
//区间查询
temp1=l*getsum(c1,l-1)-getsum(c2,l-1);
temp2=(r+1)*getsum(c1,r)-getsum(c2,r);
ans=temp2-temp1
```

例题：*https://loj.ac/problem/132*

5.4 二维树状数组

5.4.1 单点修改，区间查询

```
#define N 5050
long long tree[N][N];
long long n,m;
long long lowbit(long long x)
{
    return x&(-x);
}
void change(long long x,long long y,long long val)
{
    long long init_y=y;
    //这里注意n,m的限制
    while(x<=n)
    {
        y=init_y;
        while(y<=m)
        {
            tree[x][y]+=val;
            y+=lowbit(y);
        }
        x+=lowbit(x);
    }
}
long long getsum(long long x,long long y)
{
    long long ans=0;
    long long init_y=y;
    while(x>=1)
    {
        y=init_y;
        while(y>=1)
        {
            ans+=tree[x][y];
            y-=lowbit(y);
        }
        x-=lowbit(x);
    }
    //这里画图理解
```

```
    return ans;
}
//初始化
change(x,y,k);
//二维前缀和
ans = getsum(c,d)+getsum(a-1,b-1)-getsum(a-1,d)-getsum(c,b-1);
```

例题：*https://loj.ac/problem/133*

5.4.2 区间修改，区间查询

```
#define N 2050
long long t1[N][N];
long long t2[N][N];
long long t3[N][N];
long long t4[N][N];
long long n,m;
long long lowbit(long long x)
{
    return x&(-x);
}
long long getsum(long long x,long long y)
{
    long long ans=0;
    long long init_y=y;
    long long init_x=x;
    while(x>=1)
    {
        y=init_y;
        while(y>=1)
        {
            ans+=(init_x+1)*(init_y+1)*t1[x][y];
            ans-=(init_y+1)*t2[x][y];
            ans-=(init_x+1)*t3[x][y];
            ans+=t4[x][y];
            y-=lowbit(y);
        }
        x-=lowbit(x);
    }
    return ans;
}
void change(long long x,long long y,long long val)
{
    long long init_x=x;
    long long init_y=y;
    while(x<=n)
    {
        y=init_y;
        while(y<=m)
        {
            t1[x][y]+=val;
            t2[x][y]+=init_x*val;
            t3[x][y]+=init_y*val;
            t4[x][y]+=init_x*init_y*val;
```

```
            y+=lowbit(y);
        }
        x+=lowbit(x);
    }
}
//区间修改
change(c+1,d+1,x);
change(a,b,x);
change(a,d+1,-x);
change(c+1,b,-x);
//区间查询
ans=getsum(c,d)+getsum(a-1,b-1)-getsum(c,b-1)-getsum(a-1,d);
```

例题： *https://loj.ac/problem/135*

5.5 左偏树

5.5.1 模板

```cpp
const int N = 1e3 + 10;
struct Node {
    int k, d, fa, ch[2];  // 键，距离，父亲，左儿子，右儿子
} t[N];

// 取右子树的标号
int& rs(int x) {
    return t[x].ch[t[t[x].ch[1]].d < t[t[x].ch[0]].d];
}

// 用于删除非根节点后向上更新、
// 建议单独用，因为需要修改父节点
void pushup(int x) {
    if (!x)
        return;
    if (t[x].d != t[rs(x)].d + 1) {
        t[x].d = t[rs(x)].d + 1;
        pushup(t[x].fa);
    }
}

// 整个堆加上、减去一个数或乘上一个整数(不改变相对大小)，类似于lazy标记
void pushdown(int x) {
}

// 合并x和y
int merge(int x, int y) {
    // 若一个堆为空，则返回另一个堆
    if (!x || !y)
        return x | y;
    // 取较小的作为根
    if (t[x].k > t[y].k)
        swap(x, y);
    // 下传标记，这么写的条件是必须保证堆顶元素时刻都是最新的
    pushdown(x);
    // 递归合并右儿子和另一个堆 // 若不满足左偏树性质则交换两儿子 // 更新右子树的父亲，只有右子树有父亲
    t[rs(x) = merge(rs(x), y)].fa = x;
    // 更新dist
    t[x].d = t[rs(x)].d + 1;
    return x;
}
```

5.5.2 模板题 P3377 【模板】左偏树（可并堆）

5.5.2.1 题目描述

如题，一开始有 N 个小根堆，每个堆包含且仅包含一个数。接下来需要支持两种操作：
操作 1：1 x y 将第 x 个数和第 y 个数所在的小根堆合并（若第 x 或第 y 个数已经被删除或

第 x 和第 y 个数在用一个堆内，则无视此操作）
操作 2：2 x 输出第 x 个数所在的堆最小数，并将其删除（若第 x 个数已经被删除，则输出-1 并无视删除操作）。当堆里有多个最小值时，优先删除原序列的靠前的。

5.5.2.2 涉及知识点

1. 左偏树的基本操作（合并、删除）
2. 并查集查询结点所在的堆的根

需要注意的是：
合并前要检查是否已经在同一堆中。
左偏树的深度可能达到 $O(n)$，因此找一个点所在的堆顶要用并查集维护，不能直接暴力跳父亲。（虽然很多题数据水，暴力跳父亲可以过⋯⋯）（用并查集维护根时要保证原根指向新根，新根指向自己。）

5.5.2.3 代码

```cpp
#include <bits/stdc++.h>

using namespace std;

const int N = 1e5 + 10;

bitset<N> f;  // 用于标记某个元素是否被删除

// 关键字
struct Key {
    int a, b;
    bool operator<(const Key& rhs) const {
        if (a != rhs.a)
            return a < rhs.a;
        return b < rhs.b;
    }
};

// 左偏树节点
struct Node {
    Key k;
    int d, fa, ch[2];
} t[N];

int& rs(int x) {
    return t[x].ch[t[t[x].ch[1]].d < t[t[x].ch[0]].d];
}

int merge(int x, int y) {
    if (!x || !y)
        return x | y;
    if (t[y].k < t[x].k)
        swap(x, y);
    t[rs(x) = merge(rs(x), y)].fa = x;
    t[x].d = t[rs(x)].d + 1;
    return x;
```

```cpp
}

struct UF {
    int fa, t;
} uf[N];  // 并查集

int find(int x) {
    if (x == uf[x].fa)
        return x;
    return uf[x].fa = find(uf[x].fa);
}

void ufUnion(int x, int y) {
    x = find(x), y = find(y);
    uf[y].fa = x;
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    int n, m, i, x, y;
    cin >> n >> m;
    for (i = 1; i <= n; i++) {
        cin >> x;
        uf[i].fa = i, uf[i].t = i;
        t[i].d = 1, t[i].k = Key({x, i});
    }
    for (i = 1; i <= m; i++) {
        cin >> x;
        if (x == 1) {
            cin >> x >> y;
            if (f.test(x) || f.test(y) || uf[find(x)].t == uf[find(y)].t)
                continue;
            int root = merge(uf[find(x)].t, uf[find(y)].t);
            ufUnion(x, y);
            uf[find(x)].t = root;
        } else {
            cin >> x;
            if (f.test(x)) {
                cout << -1 << endl;
                continue;
            }
            cout << t[uf[find(x)].t].k.a << endl;
            f[t[uf[find(x)].t].k.b] = 1;
            uf[find(x)].t = merge(t[uf[find(x)].t].ch[0], t[uf[find(x)].t].ch[1]);
        }
    }

    return 0;
}
```

5.5.3 洛谷 P1552 [APIO2012] 派遣

5.5.3.1 题目描述

题目太长，简述。你有预算 m 元，给定 n 个人，具有上下级关系，构成一棵树，每个人有两个参数–花费、领导力。让你选择先选一个点作为领导者，然后在以领导者为根的子树中任意选择一些点 (要求花费不超过 m)，得到的价值为领导者的领导力 * 选定的人数。问最大价值为多少?

5.5.3.2 涉及知识点

树上问题
可并堆的合并
用堆维护背包

5.5.3.3 思路

大根堆中存储每个点的花费。递归地，对于一个点 x，我们合并 x 的所有儿子节点的堆，并计算其总和，如果总和大于 m，不断弹出堆顶元素并更新总和，直到总和小于等于 m。有点像带反悔的贪心。

5.5.4 洛谷 P3261 [JLOI2015] 城池攻占

5.5.4.1 题目描述

你要用 m 个骑士攻占 n 个城池。n 个城池 (1 到 n) 构成了一棵有根树，1 号城池为根，其余城池父节点为 fi。m 个骑士 (1 到 m)，其中第 i 个骑士的初始战斗力为 si，第一个攻击的城池为 ci。
每个城池有一个防御值 hi，如果一个骑士的战斗力大于等于城池的生命值，那么骑士就可以占领这座城池；否则占领失败，骑士将在这座城池牺牲。占领一个城池以后，骑士的战斗力将发生变化，然后继续攻击管辖这座城池的城池，直到占领 1 号城池，或牺牲为止。
除 1 号城池外，每个城池 i 会给出一个战斗力变化参数 ai;vi。若 ai =0，攻占城池 i 以后骑士战斗力会增加 vi；若 ai =1，攻占城池 i 以后，战斗力会乘以 vi。注意每个骑士是单独计算的。也就是说一个骑士攻击一座城池，不管结果如何，均不会影响其他骑士攻击这座城池的结果。
现在的问题是，对于每个城池，输出有多少个骑士在这里牺牲；对于每个骑士，输出他攻占的城池数量。

5.5.4.2 涉及知识点

树上问题
可并堆的合并
堆的整体操作 (打标记，pushdown)

5.5.4.3 注意

打标记之后，应该立即更新被打标记的点，然后 pushdown，pushdown 总是由父节点发起帮助儿子提前更新。这样，如果某个点有标记，则表示该点本身是最新的，但他应该为儿子更新。也即上一层的标记是为下一层准备的。

5.5.4.4 打标记、下传代码

```cpp
// x被打标记，立即更新x的值，并且将标记存放在此(准备之后让pushdown给下一层更新)
inline void mark(ll x, ll a, ll b) {
    if (!x)
        return;
    t[x].k.s = t[x].k.s * b + a;            // 更新自身
    t[x].a *= b, t[x].b *= b, t[x].a += a;  // 寄存标记
}

// 下传标记，本质上就是再给儿子们mark()一下，然后清空自身标记
inline void pushdown(ll x) {
    if (!x)
        return;
    mark(t[x].ch[0], t[x].a, t[x].b);
    mark(t[x].ch[1], t[x].a, t[x].b);
    t[x].a = 0, t[x].b = 1;
}
```

5.5.5 洛谷 P3273 [SCOI2011] 棘手的操作

5.5.5.1 题目描述

有 N 个节点，标号从 1 到 N，这 N 个节点一开始相互不连通。第 i 个节点的初始权值为 a[i]，接下来有如下一些操作：
U x y: 加一条边，连接第 x 个节点和第 y 个节点
A1 x v: 将第 x 个节点的权值增加 v
A2 x v: 将第 x 个节点所在的连通块的所有节点的权值都增加 v
A3 v: 将所有节点的权值都增加 v
F1 x: 输出第 x 个节点当前的权值
F2 x: 输出第 x 个节点所在的连通块中，权值最大的节点的权值
F3: 输出所有节点中，权值最大的节点的权值

5.5.5.2 涉及知识点

左偏树
并查集
multiset
整体标记
启发式合并

5.5.5.3 思路

这题题如其名，非常棘手。
首先，找一个节点所在堆的堆顶要用并查集，而不能暴力向上跳。
再考虑单点查询，若用普通的方法打标记，就得查询点到根路径上的标记之和，最坏情况下可以达到的复杂度。如果只有堆顶有标记，就可以快速地查询了，但如何做到呢？
可以用类似启发式合并的方式，每次合并的时候把较小的那个堆标记暴力下传到每个节点，然后把较大的堆的标记作为合并后的堆的标记。由于合并后有另一个堆的标记，所以较小的堆下传标记时要下传其标记减去另一个堆的标记。由于每个节点每被合并一次所在堆的大小至少乘二，所以每个节点最多被下放次标记，暴力下放标记的总复杂度就是 $O(n)$。
再考虑单点加，先删除，再更新，最后插入即可。
然后是全局最大值，可以用一个平衡树/支持删除任意节点的堆（如左偏树）/multiset 来维

护每个堆的堆顶。

所以，每个操作分别如下：
1. 暴力下传点数较小的堆的标记，合并两个堆，更新 size、tag，在 multiset 中删去合并后不在堆顶的那个原堆顶。
2. 删除节点，更新值，插入回来，更新 multiset。需要分删除节点是否为根来讨论一下。
3. 堆顶打标记，更新 multiset。
4. 打全局标记。
5. 查询值 + 堆顶标记 + 全局标记。
6. 查询根的值 + 堆顶标记 + 全局标记。
7. 查询 multiset 最大值 + 全局标记。

5.5.6 洛谷 P4331 Sequence 数字序列

5.5.6.1 题目描述

这是一道论文题
给定一个整数序列 $a_1, a_2, ..., a_n$，求出一个递增序列 $b_1 < b_2 < ... < b_n$，使得序列 a_i 和 b_i 的各项之差的绝对值之和 $|a_1 - b_1| + |a_2 - b_2| + ... + |a_n - b_n|$ 最小。

5.5.6.2 涉及知识点

堆的合并 (因为没有整体标记，所以这里可以用 __gnu_pbds::priority_queue)
堆维护区间中位数
递增序列转非递减序列（减下标法）

5.5.6.3 思路

递增序列转非递减序列：把 a[i] 减去 i，易知 b[i] 也减去 i 后答案不变，本来 b 要求是递增序列，这样就转化成了不下降序列，方便操作。
堆维护区间中位数：大根堆，每次合并之后，如果堆内元素个数大于区间的一半，则一直 pop 直到等于一半，堆顶元素即为中位数。(这么做的前提是中位数大的区间内的最小值小于等于另一个区间内仅比那个区间中位数大的数)

Chapter 6

图论

6.1 最短路

6.1.1 单源最短路径

6.1.1.1 Dijkstra

```
void Dijkstra()
{
    memset(dist, 0x3f, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    priority_queue<pii, vector<pii>, greater<pii>> q;
    dist[1] = 0;
    q.push({dist[1], 1});
    while (!q.empty())
    {
        int x = q.top().second;
        q.pop();
        if (!vis[x])
        {
            vis[x] = 1;
            for (auto it : v[x])
            {
                int y = it.first;
                if (dist[y] > dist[x] + it.second)
                {
                    dist[y] = dist[x] + it.second;
                    q.push({dist[y], y});
                }
            }
        }
    }
}
```

6.1.1.2 Bellman-Ford 和 SPFA

```
void SPFA()
{
    memset(dis, 0x3f, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    queue<int> q;
    dis[1] = 0;
    vis[1] = 1;
    q.push(1);
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for (int i = 0; i < v[x].size(); i++)
        {
            int y = v[x][i].first;
            int z = v[x][i].second;
            if (dis[y] > dis[x] + z)
```

```
            {
                dis[y] = dis[x] + z;
                if (!vis[y])
                    q.push(y), vis[y] = 1;
            }
        }
    }
}
```

例题分析

POJ3662 Telephone Lines（分层图最短路/二分答案，双端队列 BFS）

P1073 最优贸易（原图与反图，枚举节点）

P3008 [USACO11JAN] 道路和飞机 Roads and Planes（DAG，拓扑序，连通块）

6.1.2 任意两点间最短路径

6.1.2.1 Floyd

```
void get_path(int i, int j)
{
    if (!path[i][j])
        return;
    get_path(i, path[i][j]);
    p.push_back(path[i][j]);
    get_path(path[i][j], j);
}
void Floyd()
{
    memcpy(d, a, sizeof(d));
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i < k; i++)
        {
            for (int j = i + 1; j < k; j++)
            {
                //注意溢出
                ll temp = d[i][j] + a[i][k] + a[k][j];
                if (ans > temp)
                {
                    ans = temp;
                    p.clear();
                    p.push_back(i);
                    get_path(i, j);
                    p.push_back(j);
                    p.push_back(k);
                }
            }
        }
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                ll temp = d[i][k] + d[k][j];
                if (d[i][j] > temp)
```

```
                {
                    d[i][j] = temp;
                    path[i][j] = k;
                }
            }
        }
    }
}
```

例题分析

POJ1094 Sorting It All Out（传递闭包）

POJ1734 Sightseeing trip（无向图最小环）

POJ3613 Cow Relays（离散化，广义矩阵乘法，快速幂）

6.2 最小生成树

6.2.1 Kruskal

基于并查集

```
void Init()
{
    for (int i = 1; i <= n; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    int ans=0;
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        int fu = Find(u), fv = Find(v);
        if (fu != fv)
        {
            fa[fu] = fv;
            ans += e[i].w;
        }
    }
}
```

6.2.2 Prim

```
void Prim()
{
    memset(vis, 0, sizeof(vis));
    memset(d, 0x3f, sizeof(d));
    d[1] = 0;
    int temp = n;
    int ret = 0;
    while (temp--)
    {
        int min_pos = 0;
        for (int i = 1; i <= n; i++)
            if (!vis[i] && (!min_pos || d[i] < d[min_pos]))
                min_pos = i;
        if (min_pos)
        {
            vis[min_pos] = 1;
            ret += d[min_pos];
```

```
            for (int i = 1; i <= n; i++)
                if (!vis[i]) d[i] = min(d[i], weight[min_pos][i]);
        }
    }
}
```

例题分析

走廊泼水节（Kruskal，最小生成树扩充为完全图）

POJ1639 Picnic Planning（度限制最小生成树，连通块，树形 DP）

```
#include <algorithm>
#include <cstring>
#include <iostream>
#include <map>
#include <string>
#include <vector>
using namespace std;
#define inf 0x3f3f3f3f
#define N 25
#define M 500
map<string, int> name;
struct edge
{
    int u, v, w;
    bool operator<(const edge &e) const
    {
        return w < e.w;
    }
};
int n, s, ptot = 0, a[N][N], ans, fa[N], d[N], ver[N];
vector<edge> e;
bool vis[N][N];
edge dp[N]; //dp[i]  1...i路径上的最大边
void Init()
{
    for (int i = 1; i <= ptot; i++)
        fa[i] = i;
}
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
void Kruskal()
{
    Init();
    sort(e.begin(), e.end());
    for (int i = 0; i < e.size(); i++)
    {
        int u = e[i].u, v = e[i].v;
        if (u != 1 && v != 1)
        {
            int fu = Find(u), fv = Find(v);
            if (fu != fv)
```

```
                {
                    fa[fu] = fv;
                    vis[u][v] = vis[v][u] = 1;
                    ans += e[i].w;
                }
            }
        }
}
void DFS(int cur, int pre)
{
    for (int i = 2; i <= ptot; i++)
    {
        if (i != pre && vis[cur][i])
        {
            if (dp[i].w == -1)
            {
                if (dp[cur].w < a[cur][i])
                {
                    dp[i].u = cur;
                    dp[i].v = i;
                    dp[i].w = a[cur][i];
                }
                else
                    dp[i] = dp[cur];
            }
            DFS(i, cur);
        }
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    string s1, s2;
    int len;
    name["Park"] = ++ptot;
    memset(a, 0x3f, sizeof(a));
    memset(d, 0x3f, sizeof(d));
    //Park: 1
    for (int i = 0; i < n; i++)
    {
        cin >> s1 >> s2 >> len;
        if (!name[s1])
            name[s1] = ++ptot;
        if (!name[s2])
            name[s2] = ++ptot;
        int u = name[s1], v = name[s2];
        a[u][v] = a[v][u] = min(a[u][v], len); //无向图邻接矩阵
        e.push_back({u, v, len});
    }
    cin >> s; //度数限制
    ans = 0;
```

```
    Kruskal();
    for (int i = 2; i <= ptot; i++)
    {
        if (a[1][i] != inf)
        {
            int rt = Find(i);
            if (d[rt] > a[1][i])
                d[rt] = a[1][i], ver[rt] = i;
        }
    }
    for (int i = 2; i <= ptot; i++)
    {
        if (d[i] != inf)
        {
            s--;
            ans += d[i];
            vis[1][ver[i]] = vis[ver[i]][1] = 1;
        }
    }
    while (s-- > 0)
    {
        memset(dp, -1, sizeof(dp));
        dp[1].w = -inf;
        for (int i = 2; i <= ptot; i++)
        {
            if (vis[1][i])
                dp[i].w = -inf;
        }
        DFS(1, -1);
        int w = -inf;
        int v;
        for (int i = 2; i <= ptot; i++)
        {
            if (w < dp[i].w - a[1][i])
            {
                w = dp[i].w - a[1][i];
                v = i;
            }
        }
        if (w <= 0)
            break;
        ans -= w;
        vis[1][v] = vis[v][1] = 1;
        vis[dp[v].u][dp[v].v] = vis[dp[v].v][dp[v].u] = 0;
    }
    cout << "Total miles driven: " << ans << endl;
    system("pause");
    return 0;
}
```

POJ2728 Desert King （最优比率生成树，0/1 分数规划，二分）
黑暗城堡（最短路径生成树计数，最短路，排序）

6.3 树的直径

6.3.1 树形 DP 求树的直径

仅能求出直径长度，无法得知路径信息，可处理负权边。

```
int dp[N];
//dp[rt] 以rt为根的子树 从rt出发最远可达距离
/*
    对于每个结点x f[x]:经过节点x的最长链长度
*/
void DP(int rt)
{
    dp[rt]=0;//单点
    vis[rt]=1;
    for(int i=head[rt];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            DP(s);
            diameter=max(diameter,dp[rt]+dp[s]+edge[i]);
            dp[rt]=max(dp[rt],dp[s]+edge[i]);
        }
    }
}
```

6.3.2 两次 BFS/DFS 求树的直径

无法处理负权边，容易记录路径

```
void DFS(int start,bool record_path)
{
    vis[start]=1;
    for(int i=head[start];i;i=nxt[i])
    {
        int s=ver[i];
        if(!vis[s])
        {
            dis[s]=dis[start]+edge[i];
            if(record_path) path[s]=i;
            DFS(s,record_path);
        }
    }
    vis[start]=0;//清理
}
```

例题分析

P3629 [APIO2010] 巡逻（两种求树直径方法的综合应用）

P1099 树网的核（枚举）

6.4 最近公共祖先（LCA）

6.4.1 树上倍增

```
void BFS()
{
    queue<int> q;
    q.push(1);
    d[1] = 1;
    while (!q.empty())
    {
        int x = q.front();
        q.pop();
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver[i];
            if (!d[y])
            {
                d[y] = d[x] + 1;
                fa[y][0] = x;
                for (int j = 1; j <= k; j++)
                {
                    fa[y][j] = fa[fa[y][j - 1]][j - 1];
                }
                q.push(y);
            }
        }
    }
}
int LCA(int x, int y)
{
    if (d[x] < d[y])
        swap(x, y);
    for (int i = k; i >= 0; i--)
        if (d[fa[x][i]] >= d[y])
            x = fa[x][i];
    if (x == y)
        return y;
    for (int i = k; i >= 0; i--)
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}
```

6.4.2 Tarjan 算法

```
int Find(int x)
{
    if (x == fa[x])
        return x;
    return fa[x] = Find(fa[x]);
}
```

```
void Tarjan(int x)
{
    vis[x] = 1;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!vis[y])
        {
            Tarjan(y);
            fa[y] = x;
        }
    }
    for (int i = 0; i < q[x].size(); i++)
    {
        int y = q[x][i].first, id = q[x][i].second;
        if (vis[y] == 2)
            lca[id] = Find(y);
    }
    vis[x] = 2;
}
```

6.5 树上差分与 LCA 的综合应用

6.6 负环与差分约束

6.6.1 负环

例题分析

POJ3621 Sightseeing Cows（0/1 分数规划，SPFA 判定负环）

6.6.2 差分约束系统

例题分析

POJ1201 Intervals（单源最长路）

6.7 Tarjan 算法与无向图连通性

6.7.1 无向图的割点与桥

6.7.1.1 割边判定法则

```
void Tarjan(int x, int in_edge)
{
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y, i);
            low[x] = min(low[x], low[y]);
            if (low[y] > dfn[x])
            {
                bridge[i] = bridge[i ^ 1] = true;
            }
        }
        else if (i != (in_edge ^ 1))
            low[x] = min(low[x], dfn[y]);
    }
}
```

6.7.1.2 割点判定法则

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
                if (x != root || flag >= 2)
                    cut[x] = true;
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
```

例题分析
P3469 [POI2008]BLO-Blockade（割点，连通块计数）

6.7.2 无向图的双连通分量

6.7.2.1 边双连通分量 e-DCC 与其缩点

```
void DFS(int x)
{
    color[x] = dcc;
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!color[y] && !bridge[i])
            DFS(y);
    }
}
void e_DCC()
{
    dcc = 0;
    for (int i = 1; i <= n; i++)
        if (!color[i])
            ++dcc, DFS(i);
    totc = 1;
    for (int i = 2; i <= tot; i++)
    {
        int u = ver[i ^ 1], v = ver[i];
        if (color[u] != color[v])
            add_c(color[u], color[v]);
    }
    origin_bridges = (totc - 1) / 2;
    k = log2(dcc) + 1;
}
```

6.7.2.2 点双连通分量 v-DCC 与其缩点

```
void Tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    int flag = 0;
    stack[++top] = x;
    if (x == root && !head[x])
    {
        dcc[++cnt].push_back(x);
        return;
    }
    for (int i = head[x]; i; i = nxt[i])
    {
        int y = ver[i];
        if (!dfn[y])
        {
            Tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] >= dfn[x])
            {
                flag++;
```

```
                if (x != root || flag >= 2)
                    cut[x] = true;
                cnt++;
                int z;
                do
                {
                    z = stack[top--];
                    dcc[cnt].push_back(z);
                } while (z != y);
                dcc[cnt].push_back(x);
            }
        }
        else
            low[x] = min(low[x], dfn[y]);
    }
}
void v_DCC()
{
    cnt = 0;
    top = 0;
    for (int i = 1; i <= n; i++)
    {
        if (!dfn[i])
            root = i, Tarjan(i);
    }
    // 给每个割点一个新的编号(编号从cnt+1开始)
    num = cnt;
    for (int i = 1; i <= n; i++)
        if (cut[i]) new_id[i] = ++num;
    // 建新图，从每个v-DCC到它包含的所有割点连边
    tc = 1;
    for (int i = 1; i <= cnt; i++)
        for (int j = 0; j < dcc[i].size(); j++)
        {
            int x = dcc[i][j];
            if (cut[x]) {
                add_c(i, new_id[x]);
                add_c(new_id[x], i);
            }
            else c[x] = i; // 除割点外，其它点仅属于1个v-DCC
        }
}
```

例题分析

POJ3694 Network（e-DCC 缩点，LCA，并查集）

POJ2942 Knights of the Round Table（补图，v-DCC，染色法奇环判定）

6.7.3 欧拉路问题

欧拉图的判定

无向图连通，所有点度数为偶数。

欧拉路的存在性判定

无向图连通，恰有两个节点度数为奇数，其他节点度数均为偶数

```
// 模拟系统栈，答案栈
void Euler() {
    stack[++top] = 1;
    while (top > 0) {
        int x = stack[top], i = head[x];
        // 找到一条尚未访问的边
        while (i && vis[i]) i = Next[i];
        // 沿着这条边模拟递归过程，标记该边，并更新表头
        if (i) {
            stack[++top] = ver[i];
            head[x] = Next[i];
            vis[i] = vis[i ^ 1] = true;
        }
        // 与x相连的所有边均已访问，模拟回溯过程，并记录于答案栈中
        else {
            top--;
            ans[++t] = x;
        }
    }
}
```

例题分析

POJ2230 Watchcow（欧拉回路）

6.8 Tarjan 算法与有向图连通性

6.8.1 强连通分量（SCC）判定法则

```
void Tarjan(int x)
{
    dfn[x]=low[x]=++num;
    stack[++top]=x,in_stack[x]=true;
    for(int i=head[x];i;i=nxt[i])
    {
        int y=ver[i];
        if(!dfn[y])
        {
            Tarjan(y);
            low[x]=min(low[x],low[y]);
        }
        else if(in_stack[y])
            low[x]=min(low[x],dfn[y]);
    }
    if(dfn[x]==low[x])
    {
        cnt++;
        int y;
        do
        {
            y=stack[top--],in_stack[y]=false;
            color[y]=cnt, scc[cnt].push_back(y);
        } while (x!=y);
    }
}
```

6.8.2 SCC -> DAG

```
void SCC()
{
    for (int i = 0; i <= n; i++)
        if (!dfn[i])
            Tarjan(i);
    //缩点
    for (int x = 1; x <= n; x++)
    {
        for (int i = head[x]; i; i = nxt[i])
        {
            int y = ver1[i];
            if (color[x] != color[y])
                add_c(color[x], color[y]);
        }
    }
}
```

例题分析

POJ1236 Network of Schools（SCC->DAG，入度出度）
P3275 [SCOI2011] 糖果（SPFA TLE，SCC->DAG，Topo，DP）

6.8.3 有向图的必经点与必经边