1.8 分治

主定理

- 即 Master Theorem，可用于推导由分治法得到的递推关系式的时间复杂度，设

$$T(n)=aT(\frac{n}{b})+f(n)$$

- 其中 a,b 为常数，$a\geq 1,b>1$，$f(n)$ 为递推以外进行的计算工作，只能为一般的多项式。
- 考虑递归到底层时底层的总时间复杂度为 $\Theta(a^{\log_b n})=\Theta(n^{\log_b a})$，讨论 $n^{\log_b a}$ 与 $f(n)$ 的关系，则有以下结果：
  - 若 $f(n)=\mathcal{O}(n^{\log_b a-\varepsilon}),\varepsilon>0$，则 $T(n)=\Theta(n^{\log_b a})$。
  - 若 $f(n)=\Theta(n^{\log_b a}\log^k n)$，则 $T(n)=\Theta(n^{\log_b a}\log^{k+1} n)$。
  - 若 $f(n)=\Omega(n^{\log_b a+\varepsilon}),\varepsilon>0$，且对于某个常数 $c<1$ 和所有充分大的 n 有 $af(\frac{n}{b})\leq cf(n)$（正则条件），那么 $T(n)=\Theta(f(n))$。

部分复杂度分析实例

- 考虑一种分治乘法：
  - 设相乘的数分别为 a,b，在十进制下有 n 位（n 为偶数），设 a,b 的前 $\frac{n}{2}$ 位分别为 a_1,b_1，后 $\frac{n}{2}$ 位分别为 a_2,b_2。
  - 则 $ab=(10^{\frac{n}{2}}a_1+a_2)(10^{\frac{n}{2}}b_1+b_2)=10^n a_1b_1+10^{\frac{n}{2}}(a_1b_2+a_2b_1)+a_2b_2$，设
    $$\begin{aligned}m_1&=a_1b_1\\m_2&=a_2b_2\\m_3&=(a_1+a_2)(b_1+b_2)\end{aligned}$$
    则只需做 3 次乘法，有 $T(n)=3T(\frac{n}{2})+n$，应用主定理 $a_1b_2+a_2b_1=m_3-m_1-m_2$ $T(n)=\Theta(n^{\log_2 3})$
  - **Strassen算法** 用类似的方法可设计出分治矩阵乘法，将 2×2 矩阵乘法的 8 次乘法运算降至 7 次，从而将时间复杂降至 $T(n)=7T(\frac{n}{2})+18(\frac{n}{2})^2$，从而求得 $T(n)=\Theta(n^{\log_2 7})$。

- 已知 $T(n)=T(\sqrt{n})+\log n$，求 $T(n)$。
  - 设 $k=\log n$，则 $T(2^k)=2T(2^{\frac{k}{2}})+k$。
  - 再设 $S(k)=T(2^k)$，则 $S(k)=2S(\frac{k}{2})+k$，则 $S(k)=\Theta(k\log k)$。
  - 则 $T(n)=\Theta(\log n\log\log n)$。

平面最近点对

- 将所有点按 x 坐标排序，分治时二路归并实现按 y 坐标排序。
- 设当前区间为 $[l,r]$，取中点 mid，其 x 为 x_{mid}，设分治得到 $[l,mid]$ 和 $[mid+1,r]$ 内最近点对距离的最小值为 δ，我们只需取出所有满足 $|x_i-x_{mid}|<\delta$ 的点 i 组成一个点集，并检查点集内部比每个点 y 坐标小且相差不超过 δ 的点更新 δ 即可。
- 运用鸽巢原理可以证明，对于点集内的每个点，每次检查的点数为 $\mathcal{O}(1)$，总时间复杂度 $\Theta(n\log n)$。

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

template <class T>
inline void read(T &res)
{
    char ch; bool flag = false; res = 0;
    while (ch = getchar(), !isdigit(ch) && ch != '-');
    ch == '-' ? flag = true : res = ch ^ 48;
    while (ch = getchar(), isdigit(ch))
        res = res * 10 + ch - 48;
    flag ? res = -res : 0;
}

typedef long long ll;
typedef long double ld;
const ld Maxn = 4e18;
const int N = 2e5 + 5;
int n, tn;

template <class T>
inline T Min(T x, T y) {return x < y ? x : y;}
template <class T>
inline void CkMin(T &x, T y) {x > y ? x = y : 0;}
template <class T>
inline T Abs(T x) {return x < 0 ? -x : x;}

struct point
{
    int x, y;

    point() {}
    point(int X, int Y):
        x(X), y(Y) {}

    inline ld dist()
    {
        return sqrt((ld)x * x + (ld)y * y);
    }

    inline void scan()
    {
        read(x);
```

```
        read(y);
    }

    inline point operator - (const point &a) const
    {
        return point(x - a.x, y - a.y);
    }
}p[N], t[N];

inline bool cmpx(const point &a, const point &b)
{
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

inline bool cmpy(const point &a, const point &b)
{
    return a.y < b.y || a.y == b.y && a.x < b.x;
}

inline ld solve(int l, int r)
{
    if (l == r)
        return Maxn;
    int mid = l + r >> 1, xmid = p[mid].x;
    ld dist = Min(solve(l, mid), solve(mid + 1, r));
    std::inplace_merge(p + l, p + mid + 1, p + r + 1, cmpy);
    tn = 0;
    for (int i = l; i <= r; ++i)
        if (Abs(p[i].x - xmid) < dist)
        {
            for (int j = tn; j >= 1 && p[i].y - t[j].y < dist; --j)
                CkMin(dist, (p[i] - t[j]).dist());
            t[++tn] = p[i];
        }
    return dist;
}

int main()
{
    read(n);
    for (int i = 1; i <= n; ++i)
        p[i].scan();
    std::sort(p + 1, p + n + 1, cmpx);
    printf("%.4lf\n", (double)solve(1, n));
}
```

求序列第 k 小数

- 即 STL 中 `nth_element` 函数实现原理，可用类似快速排序的 `Partition` 函数求解该问题。
- 设总时间复杂度为 $T(n)$，假设我们严格随机地选取 $pivot$，有递推式：

$$T(n)=\frac{1}{n}\sum_{i=1}^{n-1}T(i)+n\Leftrightarrow nT(n)=\sum_{i=1}^{n-1}T(i)+n^2$$

- 另取 $(n-1)T(n-1)=\sum\limits_{i=1}^{n-2}T(i)+(n-1)^2$，错位相消可得：

$$T(n)=T(n-1)+2-\frac{1}{n}\Rightarrow T(n)=\mathcal{O}(n)$$

- 该算法最坏情况下复杂度为 $\mathcal{O}(n^2)$，但概率极低，可忽略不计。

```cpp
inline int Rand(int l, int r)
{
    return 1ll * rand() * rand() % (r - l) + l;
}

inline int Partition(int *a, int l, int r)
{
    std::swap(a[l], a[Rand(l, r)]);
    int pivot = a[l];
    while (l < r)
    {
        while (l < r && a[r] >= pivot)
            --r;
        a[l] = a[r];
        while (l < r && a[l] <= pivot)
            ++l;
        a[r] = a[l];
    }
    a[l] = pivot;
    return l;
}

inline void nthElement(int *a, int l, int r, int k)
{
    if (l == r)
        return ;
    int pos = Partition(a, l, r);
    int cnt = pos - l + 1;
    if (k == cnt)
        return ;
    else if (k > cnt)
        nthElement(a, pos + 1, r, k - cnt);
    else
        nthElement(a, l, pos - 1, k);
}
```

网格图最短路

- 给定一个总结点数为 S 的网格图，边权均为正，q 次询问两点间的最短距离，考虑一种分治做法：
  - 设当前分治区域的顶点数为 S，取较长边的中线作划分，显然中线上的顶点数至多为 $\sqrt{S}$，分别以中线上所有顶点为起点做单源最短路。
  - 对于跨过中线的询问，暴力枚举其最短路径跨过中线的结点即可求解。
  - 对于未跨过中线的询问，递归两个子区域求解。
  - 设做单源最短路的总时间复杂度为 $T(S)$，由主定理 $T(S)=2T(\frac{S}{2})+\mathcal{O}(S\sqrt{S}\log S)$，故 $T(S)=\mathcal{O}(S\sqrt{S}\log S)$，总时间复杂度 $\mathcal{O}((S\log S+q)\sqrt{S})$。
- 把每层预处理的结果存储下来，就能支持在线操作。
- 该分治方法可以推广到任意具有可分治性质且多次询问起终点最短路的图中。

CDQ分治

- 设当前分治区间为 $[l,r]$，取中点 mid，执行如下步骤：
  - 递归 $[l,mid]$。
  - 处理 $[l,mid]$ 到 $[mid+1,r]$ 的影响。
  - 递归 $[mid+1,r]$。
- 在保证正确性的情况下，上述步骤具体执行顺序可视情况而定。

典例 [Violet]天使玩偶

题目大意

- 动态加点，每次询问距离某点曼哈顿距离最小的点，允许离线。

解法

- 先考虑计算每个点左下角的最近点，其余情况对坐标做变换后处理方式相同。
- 即求时间顺序在该询问之前且横纵坐标均不大于该点的点中横纵坐标之和的最大值，显然这是一个三维偏序问题，可用 CDQ 分治解决。

线段树分治

- 考虑下面这一问题：
  - 无向图 G_1 由 n 个点 m 条边构成，$G_i(1<i\le k)$ 由 $G_{p_i}(p_i<i)$ 增加一条边/删去一条边生成。
  - 求将 $G_1,G_2,\dots,G_k$ 分组，使得任意两张任意两点的连通性相同的图被分在同一组。
  - $1\le n,m,k\le 10^5$。
- 首先我们简化任意两张图连通性的判断，定义无向图 $G=(V,E)$ 连通性的哈希函数为：

$$H(G)=\left(\sum_{G\text{的连通分量}G'=(V',E')}\min_{x\in V'}\{x\}\sum_{x\in V'}B^x\right)\mod P$$

- 其中 B,P 为任取的质数，我们只需将哈希值相同的图分为一组即可。
- 显然 $p_i\to i$ 构成一个有根树结构，若该树退化成 k 个点的链，则是经典的线段树分治。
- 把链上每个点看做是一个时间点，则我们可以得到每条边的出现时间的区间，以时间为域建线段树，任意一个区间都可以被拆分成线段树上的 $\mathcal{O}(\log k)$ 个区间，我们将这些边挂在这些区间上并 DFS 线段树，用按秩合并的并查集即可实现可撤销地维护图的连通性和其哈希值，到达叶子结点时即可得到当前时间点对应的哈希值。
- 对于一般的树结构，我们只需要 DFS 整棵树，将从父结点向子结点走和从子结点向父结点走视作相反的操作，即可转化成链上的问题。
- 注意线段树叶结点的回溯。
- 类似的思想也可以拓广到其它二叉树的数据结构上，如 **Trie** 等。
- 可撤销并查集可以通过用栈记录指针和其原始值来实现，具体代码如下。

```
struct traceback
{
    int *addr, val;
    traceback() {}
    traceback(int *Addr, int Val):
        addr(Addr), val(Val) {}
}stk[L];

inline void Change(int &x, int v)
{
    stk[++top] = traceback(&x, x);
    x = v;
```

```
}

inline void Back(int _top)
{
    while (top != _top)
    {
        *(stk[top].addr) = stk[top].val;
        --top;
    }
}
```

点分治

典例/模板 洛谷P2664 树上游戏

题目大意

- 给定一个 n 个点（$n \leq 10^5$）的树，定义 $s(i,j)$ 表示 i 到 j 的颜色数，对于每个 i，求

$$sum_i = \sum_{j=1}^{n} s(i,j)$$

解法

- 由于并不需要去计算 $s(i,j)$，考虑对于每个 sum_i 每种颜色的贡献。
- 具体地，对于每个点分治重心 x 下子树 y 内的一个点 u，
  - 对于 x 到 u 路径上出现的颜色，每种颜色的贡献为 $\text{size}_x - \text{size}_y$。
  - 对于 x 到 u 路径上未出现的颜色，设颜色 c 的贡献为 csize_c，则对于与 u 分属不同子树的结点 v，若点 v 的颜色为 c 且 c 在 x 到 v 的路径上第一次出现，则对 csize_c 的贡献为 size_v，具体计算时可以先算出全局的 csize_c，再扣除子树 y 内的部分。

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

using std::ios;
using std::cin;
using std::cout;

typedef long long ll;
const int Maxn = 1e9;
const int N = 1e5 + 5;
const int N2 = 2e5 + 5;
int n, m, Gsze, Gid, tot, top, apr_col, extra;
int sze[N], col[N], stk[N], cnt[N], csze[N];
ll tot_csze, sum[N]; bool vis[N];

struct arc
{
    int to, cst;
    arc *nxt;
}p[N2], *adj[N], *P = p;

inline void linkArc(int x, int y)
{
    (++P)->nxt = adj[x]; adj[x] = P; P->to = y;
    (++P)->nxt = adj[y]; adj[y] = P; P->to = x;
```

```
}

inline void CkMax(int &x, int y) {if (x < y) x = y;}

inline void dfsInit(int x, int fa)
{
    int cnt = 0;
    sze[x] = 1;
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (y == fa || vis[y])
            continue;
        dfsInit(y, x);
        sze[x] += sze[y];
        CkMax(cnt, sze[y]);
    }
    CkMax(cnt, tot - sze[x]);
    if (cnt < Gsze)
        Gsze = cnt, Gid = x;
}

inline int findG(int x)
{
    Gsze = Maxn;
    dfsInit(x, 0);
    return Gid;
}

inline void addCol(int x, int t)
{
    stk[++top] = x;
    if (++cnt[col[x]] == 1)
    {
        csze[col[x]] += t * sze[x];
        tot_csze += t * sze[x];
    }
}

inline void delCol()
{
    int x = stk[top--];
    --cnt[col[x]];
}

inline void dfsCol1(int x, int Fa, int t)
{
    addCol(x, t);
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (vis[y] || y == Fa)
            continue ;
        dfsCol1(y, x, t);
    }
    delCol();
}

```

```cpp
inline void dfsCol2(int x, int Fa)
{
    if (++cnt[col[x]] == 1)
    {
        tot_csze -= csze[col[x]];
        ++apr_col;
    }
    sum[x] += 1ll * apr_col * extra + tot_csze;
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (y == Fa || vis[y])
            continue ;
        dfsCol2(y, x);
    }
    if (--cnt[col[x]] == 0)
    {
        tot_csze += csze[col[x]];
        --apr_col;
    }
}

inline void dfsSze(int x, int Fa)
{
    sze[x] = 1;
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (vis[y] || y == Fa)
            continue ;
        dfsSze(y, x);
        sze[x] += sze[y];
    }
}

inline void solve(int x)
{
    x = findG(x);
    dfsSze(x, 0);
    //注意求完重心后一定要重新求一遍 size，否则递归到子树时 tot 的计算是错误的
    vis[x] = true;
    addCol(x, 1);
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (vis[y])
            continue ;
        dfsCol1(y, x, 1);
    }
    sum[x] += tot_csze;
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (vis[y])
            continue ;
        dfsCol1(y, x, -1);

        extra = tot - sze[y];
```

```
        tot_csze -= csze[col[x]];
        ++apr_col;
        dfsCol2(y, x);
        --apr_col;
        tot_csze += csze[col[x]];

        dfsCol1(y, x, 1);
    }
    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (vis[y])
            continue ;
        dfsCol1(y, x, -1);
    }
    top = cnt[col[x]] = csze[col[x]] = tot_csze = 0;

    for (arc *e = adj[x]; e; e = e->nxt)
    {
        int y = e->to;
        if (vis[y])
            continue ;
        tot = sze[y];
        solve(y);
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    cin >> n;
    for (int i = 1; i <= n; ++i)
        cin >> col[i];
    for (int i = 1, x, y; i < n; ++i)
    {
        cin >> x >> y;
        linkArc(x, y);
    }
    tot = n;
    solve(1);
    for (int i = 1; i <= n; ++i)
        cout << sum[i] << '\n';
}
```

动态点分治

- 本质上是在点分树上维护一些数据结构，注意需要避免计算来自同子树的贡献，支持强制在线。
- 一般会预处理出某点到其所有点分树祖先的距离，只能逐个求 LCA 计算。

典例 洛谷P2056

题目大意

- 在静态树上动态增删点，询问最远点对。

解法

- 在点分树上维护若干可删堆，实现见代码。
  - dist_x 维护点分树上以 x 为根的子树内所有结点到 x 父结点的距离的集合
  - ch_x 维护点分树上以 x 每个子结点 y 为根的子树内所有结点到 x 的距离的最大值的集合，用 dist_y 中的最大值来更新。
  - ans 维护经过点分树上某点 x 的最远点对的集合，用 ch_x 中的最大值和次大值来更新。

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

using std::cin;
using std::cout;
using std::ios;
using std::priority_queue;
using std::vector;

const int N = 1e5 + 5;
const int Maxn = 1e9;
int col[N], sze[N], anc[N][20], tdep[N], dep[N], tanc[N], tdis[N][20];
int n, Q, rt, Gsze, Gid, tot;
vector<int> e[N]; bool vis[N];

template <class T>
inline void CkMax(T &x, T y) {x < y ? x = y : 0;}

struct delHeap //可删大根堆
{
    priority_queue<int> a, b; //heap = a - b

    inline void Push(int v) {a.push(v);}
    inline void Erase(int v) {b.push(v);}
    inline int Size() {return a.size() - b.size();}

    inline void Pop()
    {
        while (!b.empty() && a.top() == b.top())
            a.pop(), b.pop();
        a.pop();
    }

    inline int Top()
    {
        while (!b.empty() && a.top() == b.top())
            a.pop(), b.pop();
        return a.top();
    }

    inline int Top2()
    {
        int x = Top();
```

```
        a.pop();
        int y = Top();
        a.push(x);
        return y;
    }
}dist[N], ch[N], ans;

inline int queryLCA(int x, int y)
{
    if (x == y)
        return x;
    if (dep[x] < dep[y])
        std::swap(x, y);
    for (int i = 18; i >= 0; --i)
    {
        if (dep[anc[x][i]] >= dep[y])
            x = anc[x][i];
        if (x == y)
            return x;
    }
    for (int i = 18; i >= 0; --i)
        if (anc[x][i] != anc[y][i])
            x = anc[x][i], y = anc[y][i];
    return anc[x][0];
}

inline int queryDis(int x, int y)
{
    return dep[x] + dep[y] - (dep[queryLCA(x, y)] << 1);
}

inline void initLCA(int x)
{
    dep[x] = dep[anc[x][0]] + 1;
    for (int i = 0; anc[x][i]; ++i)
        anc[x][i + 1] = anc[anc[x][i]][i];
    for (int y : e[x])
    {
        if (y == anc[x][0])
            continue ;
        anc[y][0] = x;
        initLCA(y);
    }
}

inline void dfsInit(int x, int Fa)
{
    int cnt = 0;
    sze[x] = 1;
    for (int y : e[x])
    {
        if (vis[y] || y == Fa)
            continue;
        dfsInit(y, x);
        sze[x] += sze[y];
        CkMax(cnt, sze[y]);
    }
    CkMax(cnt, tot - sze[x]);
```

```
    if (cnt < Gsze)
        Gsze = cnt, Gid = x;
}

inline void dfsSze(int x, int Fa, int rt, int d)
{
    sze[x] = 1;
    if (rt)
        dist[rt].Push(tdep[x]);
    tdep[x] = d;
    for (int y : e[x])
    {
        if (vis[y] || y == Fa)
            continue ;
        dfsSze(y, x, rt, d + 1);
        sze[x] += sze[y];
    }
}

inline int findG(int x)
{
    Gsze = Maxn;
    dfsInit(x, 0);
    return Gid;
}

inline void solve(int x, int Fa)
{
    vis[x = findG(x)] = true;
    tanc[x] = Fa;
    dfsSze(x, 0, Fa ? x : 0, 0);
    if (!Fa)
        rt = x;
    else
        ch[Fa].Push(dist[x].Top());
    ch[x].Push(0);
    for (int y : e[x])
    {
        if (vis[y])
            continue ;
        tot = sze[y];
        solve(y, x);
    }
    if (ch[x].Size() >= 2)
        ans.Push(ch[x].Top() + ch[x].Top2());
}

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    cin >> n;
    for (int i = 1, u, v; i < n; ++i)
    {
        cin >> u >> v;
        e[u].emplace_back(v);
```

```
        e[v].emplace_back(u);
    }
    initLCA(1);
    tot = n;
    solve(1, 0);
    for (int x = 1; x <= n; ++x)
        col[x] = 1;
    for (int x = 1; x <= n; ++x)
        for (int y = tanc[x], t = 0; y; y = tanc[y], ++t)
            tdis[x][t] = queryDis(x, y);
    char opt; int x, cnt = n;
    cin >> Q;
    while (Q--)
    {
        cin >> opt;
        if (opt == 'C')
        {
            cin >> x;
            int u = x;
            if (ch[x].Size() >= 2)
                ans.Erase(ch[x].Top() + ch[x].Top2());
            if (col[u])
                ch[x].Erase(0);
            else
                ch[x].Push(0);
            if (ch[x].Size() >= 2)
                ans.Push(ch[x].Top() + ch[x].Top2());
            for (int y = tanc[x], t = 0; x; x = y, y = tanc[y], ++t)
            {
                if (y)
                {
                    if (ch[y].Size() >= 2)
                        ans.Erase(ch[y].Top() + ch[y].Top2());
                    if (dist[x].Size())
                        ch[y].Erase(dist[x].Top());
                }
                if (col[u])
                    dist[x].Erase(tdis[u][t]);
                else
                    dist[x].Push(tdis[u][t]);
                if (y)
                {
                    if (dist[x].Size())
                        ch[y].Push(dist[x].Top());
                    if (ch[y].Size() >= 2)
                        ans.Push(ch[y].Top() + ch[y].Top2());
                }
            }
            col[u] ^= 1;
            if (col[u])
                ++cnt;
            else
                --cnt;
        }
        else
        {
            if (ans.Size())
                cout << ans.Top() << '\n';
```

```
            else if (cnt == 0)
                cout << -1 << '\n';
            else
                cout << 0 << '\n';
        }
    }
}
```