NOIP2018提高组题解

Day1 T1 铺设道路

题意

~~详见NOIP2013积木大赛~~

~~或者参考USACO 2013 March - Poker Hands~~

给定一个非负整数序列$\{a_n\}$，每次操作可以选择一个不含0的区间$[L,R]$，让区间中每个数都减1。求使得序列全为0的最少次数。

$n\leq 10^5,a_i\leq 10^4$

题解

解法很多。

解法一

差分。钦定$a_0=0$。

结论：对于所有的$1\leq i\leq n$，若$a_i>a_{i-1}$，则答案加上a_i-a_{i-1}。

可以用一个类似DP的东西证明。设dp_i表示使序列前i个数变为0的最小次数。显然，当$a_i\leq a_{i-1}$时，$dp_i=dp_{i-1}$；当$a_i\geq a_{i-1}$时，a_i可以在前面减a_{i-1}时顺便减去a_{i-1}，剩下的a_i-a_{i-1}只能通过自减，所以此时$dp_i=dp_{i-1}+a_i-a_{i-1}$。

其实就是上面那个结论。时间复杂度$\mathcal{O}(n)$。

解法二

我考场的做法。RMQ+分治。

显然每次把一段最长的不包含0的区间减1最优。

对于一个区间$[l,r]$，记mid为区间中最小值所在的位置，显然最多只能减a_{mid}次，然后a_{mid}变为0。其他的怎么办呢？分治即可。

找最小值可以用ST表，然而CCF的数据是随机的，分治期望$\log n$层，所以每次暴力找最小值也可以过此题。

ST+分治：$\mathcal{O}(n\log n+n)=\mathcal{O}(n\log n)$

暴力：期望$\mathcal{O}(n\log n)$，最坏$\mathcal{O}(n^2)$。

代码

ST+分治：

```
#include <cstdio>
#include <cctype>
#include <cstring>
#include <algorithm>
int read(){
    register int x = 0;
    register char f = 1, ch = getchar();
    for (; !isdigit(ch); ch = getchar()) if (ch == '-') f = 0;
    for (; isdigit(ch); ch = getchar()) x = (x << 1) + (x << 3) + (ch ^ '0');
    return f ? x : -x;
}
#define N 100005
int n, a[N], st[N][25], Log[N], ans;
int Min(int x, int y){ return a[x] < a[y] ? x : y; }
int query(int l, int r){
    int t = Log[r - l + 1];
    return Min(st[l][t], st[r - (1 << t) + 1][t]);
}
void solve(int l, int r, int h){
    if (l > r) return;
    int mid = query(l, r);
    ans += a[mid] - h;
    solve(l, mid - 1, a[mid]), solve(mid + 1, r, a[mid]);
}
int main(){
    freopen("road.in", "r", stdin);
    freopen("road.out", "w", stdout);
    n = read();
    for (register int i = 1; i <= n; ++i) a[i] = read(), st[i][0] = i;
    Log[1] = 0;
    for (register int i = 2; i <= n; ++i) Log[i] = Log[i >> 1] + 1;
    for (register int j = 1; j <= Log[n]; ++j)
        for (register int i = 1; i + (1 << j) - 1 <= n; ++i)
            st[i][j] = Min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    solve(1, n, 0);
    printf("%d", ans);
}
```

Day1 T2 货币系统

题意

给定一个大小为n的正整数集合S，求最小的m，使得存在一个大小为m的正整数集合S'满足：对于任意非负整数x，如果x可以用集合S中的数表示出来（可以重复，下同），那么x也可以用集合S'中的数表示出来；如果x可以用集合S'中的数表示出来，那么x也可以用集合S中的数表示出来。

T组数据，$T \leq 20, n \leq 100, a_i \leq 25000$

题解

只要去掉S中能被其它数表示出来的数，排序后做完全背包即可。时间复杂度$\mathcal{O}(Tn \cdot \max\{a_i\})$

代码

```cpp
#include <cstdio>
#include <cctype>
#include <cstring>
#include <algorithm>
int read(){
    register int x = 0;
    register char f = 1, ch = getchar();
    for (; !isdigit(ch); ch = getchar()) if (ch == '-') f = 0;
    for (; isdigit(ch); ch = getchar()) x = (x << 1) + (x << 3) + (ch ^ '0');
    return f ? x : -x;
}
int T, n, a[105], m, dp[25005], ans;
int main(){
    freopen("money.in", "r", stdin);
    freopen("money.out", "w", stdout);
    T = read();
    while (T--){
        ans = n = read();
        for (register int i = 1; i <= n; ++i) a[i] = read();
        std :: sort(a + 1, a + 1 + n), m = a[n];
        memset(dp, 0, sizeof dp), dp[0] = 1;
        for (register int i = 1; i <= n; ++i){
            if (dp[a[i]]) --ans;
            for (register int j = a[i]; j <= m; ++j)
                dp[j] |= dp[j - a[i]];
        }
        printf("%d\n", ans);
    }
}
```

Day1 T3 赛道修建

题意

给定一棵n个节点的带边权树，你需要选择m条路径满足每条边最多只被一条路径覆盖，求m条路径中长度最短的路径最大的长度。

$n \leq 50000$

题解

最大化最小值的问题，可以用二分把问题转化成判定性问题：

> 给定一棵n个节点的带边权树，你需要选择m条路径满足每条边最多只被一条路径覆盖，**判断**是否有一种方案满足每条路径的长度$\geq mid$。

菊花树在一定程度上提示了正解 ~~（然而我被菊花树误导了）~~。

所以我们先考虑菊花树的做法。

把边按权值从小到大排序后，显然可以先把边权$\geq mid$的边去掉，贡献加1。

然后我们贪心的组合：因为不需要考虑其他的，只要让路径最多即可，所以最小的与最大的组合即可。

类比一般情况，向当前点的每个儿子往下延伸出一条最长的没有被组合掉的链，把这条链的长度当作边权，然后尽量组合，把一条没有被组合的链向上传即可。

可是，因为现在在贡献最大的情况下，我们应该保证传上去的链最长，所以我们不能直接把最大的和最小的组合。

一样地，先把单条长度$\geq mid$的链去掉，统计贡献。

然后按链长从小到大排序，对于每条链h，我们只要找到一个t满足$t > h$并且$a_t + a_h \geq mid$（a_i表示从小到大排序后第i条链的长度）。如果存在这样的t，那么把h, t组合，然后把h, t两条链删除即可。

具体实现可以用双向链表实现删除操作，h从前往后扫，t根据h变化，复杂度可以保证。

时间复杂度$\mathcal{O}(n \log^2 n)$。

一个log是二分，另一个log是对每个节点处理时的排序复杂度。

也可以使用单调栈实现，整体思想与上面的类似，不再详述。

代码

单调栈（不易写错）：

```
#include <cstdio>
#include <cctype>
#include <algorithm>
int read(){
    register int x = 0, ch = getchar();
    for (; !isdigit(ch); ch = getchar()) ;
    for (; isdigit(ch); ch = getchar()) x = (x << 1) + (x << 3) + (ch ^ '0');
    return x;
}
#define N 50005
int n, m;
int edge, to[N << 1], pr[N << 1], tw[N << 1], hd[N];
void addedge(int u, int v, int w){
    to[++edge] = v, tw[edge] = w, pr[edge] = hd[u], hd[u] = edge;
}
int l, r, mid, ans, s, val[N], cnt, a[N];
int dfs(int x, int u, int fa = 0){
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa) val[v] = dfs(x, v, u) + tw[i];
    cnt = 0;
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa) val[v] >= x ? ++s : a[++cnt] = val[v];
    if (!cnt) return 0;
    std :: sort(a + 1, a + 1 + cnt);
    int h = 1, t = cnt, mx = 0, top = 0, tmp = 0;
    while (h <= t){
        while (a[h] + a[t] >= x && h < t) ++top, !tmp ? tmp = a[t] : 0, --t;
        if (top) --top, ++s, ++h; else mx = a[h++];
    }
    return s += top >> 1, std :: max(mx, top & 1 ? tmp : 0);
}
bool check(int x){
    return s = 0, dfs(x, 1), s >= m;
}
int main(){
    freopen("track.in", "r", stdin);
    freopen("track.out", "w", stdout);
    n = read(), m = read();
    for (register int i = 1, u, v, w; i < n; ++i)
        u = read(), v = read(), r += w = read(), addedge(u, v, w), addedge(v, u, w);
    r /= m;
    while (l <= r) check(mid = (l + r) >> 1) ? ans = mid, l = mid + 1 : r = mid - 1;
    printf("%d", ans);
}
```

双向链表（h,t的移动容易写错，但是比较好理解）：

```
#include <cstdio>
#include <cctype>
#include <algorithm>
int read(){
    register int x = 0, ch = getchar();
    for (; !isdigit(ch); ch = getchar()) ;
    for (; isdigit(ch); ch = getchar()) x = (x << 1) + (x << 3) + (ch ^ '0');
    return x;
}
#define N 50005
int n, m;
int edge, to[N << 1], pr[N << 1], tw[N << 1], hd[N];
void addedge(int u, int v, int w){
    to[++edge] = v, tw[edge] = w, pr[edge] = hd[u], hd[u] = edge;
}
int l, r, mid, ans, s, cnt, a[N], L[N], R[N], val[N];
void del(int x){
    R[L[x]] = R[x], L[R[x]] = L[x];
}
int dfs(int x, int u, int fa = 0){
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa) val[v] = dfs(x, v, u) + tw[i];
    cnt = 0;
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa) val[v] >= x ? ++s : a[++cnt] = val[v];
    if (!cnt) return 0;
    std :: sort(a + 1, a + 1 + cnt), a[cnt + 1] = 1000000000;
    register int h = 1, t = cnt;
    for (register int i = 0; i <= cnt + 1; ++i) L[i] = i - 1, R[i] = i + 1;
    R[cnt + 1] = cnt + 1;
    while (h <= cnt){
        while (h < t && a[h] + a[t] >= x) t = L[t];
        while ((t = R[t]) <= h) ;
        if (t != cnt + 1) ++s, del(h), del(t), R[h] == t ? h = R[h] : 0, t = L[t];
        h = R[h];
    }
    return a[L[cnt + 1]];
}
bool check(int x){
    return s = 0, dfs(x, 1), s >= m;
}
int main(){
    freopen("track.in", "r", stdin);
    freopen("track.out", "w", stdout);
    n = read(), m = read();
    for (register int i = 1, u, v, w; i < n; ++i)
        u = read(), v = read(), r += w = read(), addedge(u, v, w), addedge(v, u, w);
    r /= m;
    while (l <= r) check(mid = (l + r) >> 1) ? ans = mid, l = mid + 1 : r = mid - 1;
    printf("%d", ans);
}
```

Day2 T1 旅行

转化后的题意

1. 给定一棵n个点的树，求最小的DFS序。
2. 给定一棵n个点的基环树，删去环上的任意一条边，求所有情况中的最小DFS序。

$n \leq 5000$

题解

原题转化为上面的题意：树的情况显然是对的；基环树的情况，因为最后的方案中一定会有一条环上的边没有经过，尝试删去每条边即可。

先考虑树的情况。我们只要把每个点连着的点按编号排序（或者一开始把边排序，然后按顺序连），然后在DFS中对于每个点，从小到大依次DFS每个儿子即可。一开始$\mathrm{DFS}(1)$。

下文讨论基环树的情况。

解法一

按这个转化后的题意，枚举环上每条边删去，然后按树的方法做DFS即可。时间复杂度$\mathcal{O}(n^2)$。

如果不会找环，也可以枚举所有边删去，在DFS时加入vis数组防止进入环中无限循环，然后在更新答案时判断访问到的点是否$= n$。

此种方法实现较简单，可以参考解法二的部分代码，故不给出代码。

解法二

考虑直接求出字典序最小时删除的边。

先找环，求出nx_i, fa_i, r, fr。nx_i表示编号为i的点在环上的下一个节点，fa_i表示上一个节点，r表示从1开始DFS时到环中的第一个点（暂且称为入点），fr表示以1为根时r的父亲节点。

这些都可以用一个DFS在$\mathcal{O}(n)$的复杂度内求出。

然后我们可以求出mxs_i，表示环上的节点i，不在环上的但与i相连的节点（姑且称这些点为i的儿子）中编号最大的点。

显然，我们从1开始DFS，到达入点r并且将要进入环时，一定会选择小的点进入。假设$nx_r < fa_r$，即我们在环上按$r \to nx_r \to nx_{nx_r} \to \cdots$的方向走。

假设当前点为i，若$nx_i < mxs_i$，那么i与nx_i的连边一定不能删除，否则将会导致答案变劣。

可是这样就够了吗？如果$nx_i > mxs_i$难道一定要删除吗？

我们发现，当$nx_i > mxs_i$时，如果删除i和nx_i之间的连边，我们一定会先遍历完i的所有儿子，然后回到上一个还有儿子未遍历的点并且遍历还未遍历过的点中编号最小的点。那么如果这个点比nx_i大，那么答案就变劣了。

基于这个原因，我们再定义数组cut_i。在r到i的路径中，设x为存在编号$> i$的儿子且离i最近的点，则cut_i为x的最小的$> i$的儿子编号。

所以，i与nx_i的连边**保留**的条件为$nx_i < mxs_i$或者$nx_i < cut_i$。

时间复杂度为给边排序的复杂度$\mathcal{O}(n \log n)$。

代码

$\mathcal{O}(n \log n)$的代码，可以通过数据加强版 - 洛谷。

```
#include <cstdio>
#include <cctype>
#include <algorithm>
int read(){
    register int x = 0, ch = getchar();
    for (; !isdigit(ch); ch = getchar());
    for (; isdigit(ch); ch = getchar()) x = (x << 1) + (x << 3) + (ch ^ '0');
    return x;
}
#define N 500005
#define INF 0x3f3f3f3f
int n, m, k, a[N], x, y, r, fr, fa[N], nx[N], mxs[N];
int edge, to[N << 1], pr[N << 1], hd[N << 1];
struct Edge{
    int x, y;
    bool operator < (const Edge &res) const {
        return x < res.x || x == res.x && y < res.y;
    }
}E[N];
void addedge(int u, int v){
    to[++edge] = v, pr[edge] = hd[u], hd[u] = edge;
}
bool find_cycle(int u, int Fa = 0){
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != Fa){
            if (!fa[v]) fa[v] = u; else return r = v, fr = fa[v], fa[v] = u, nx[u] = v, 1;
            if (find_cycle(v, u)) return nx[u] = v, 1;
        }
    return 0;
}
void get_mxs(){
    for (register int u = nx[r]; u != r; u = nx[u]){
        for (register int i = hd[u], v; i; i = pr[i])
            if ((v = to[i]) != fa[u] && v != nx[u]) mxs[u] = std :: max(mxs[u], v);
    }
}
int get_cut(int u, int Fa){
    int res = INF;
    for (register int i = hd[u], Nx = nx[u], v; i; i = pr[i])
        if ((v = to[i]) != Fa && v > Nx) res = std :: min(res, v);
    return res;
}
void get_cut_edge(){
    get_mxs();
    int i = nx[r], cut = get_cut(r, fr), tmp;
    for (; nx[i] != r && (nx[i] < mxs[i] || nx[i] > mxs[i] && nx[i] < cut); i = nx[i])
        tmp = get_cut(i, fa[i]), tmp != INF ? cut = tmp : 0;
    x = i, y = nx[i];
}
void get_ans(int u, int fa = 0){
    a[++k] = u;
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa && (x != u || y != v) && (x != v || y != u)) get_ans(v, u);
}
int main(){
    freopen("travel.in", "r", stdin);
    freopen("travel.out", "w", stdout);
    n = read(), m = read();
    for (register int i = 1; i <= m; ++i)
        E[i].x = read(), E[i].y = read(), E[i].x > E[i].y ? std :: swap(E[i].x, E[i].y), 0 : 0;
    std :: sort(E + 1, E + 1 + m);
    for (register int i = m; i; --i) addedge(E[i].x, E[i].y), addedge(E[i].y, E[i].x);
    if (m == n) fa[1] = -1, find_cycle(1), get_cut_edge();
    get_ans(1);
    for (register int i = 1; i < n; ++i) printf("%d ", a[i]); printf("%d\n", a[n]);
}
```

Day2 T2 填数游戏

题意

有一个$n \times m$的矩阵，你可以在每个格子里填入0或1。

定义一条合法路径P，为从$(0,0)$开始，每次只能向右(R)或向下(D)，走到$(n-1,m-1)$的路径。

对于一条合法路径P，记$w(P)$为由路径每次的“决策”组成（即由D,R组成）的字符串，$s(P)$为路径经过的格子上的数依次连接的字符串。

求满足以下条件的填格子的方案数对10^9+7取模的结果：

对于两条合法路径P_1,P_2，如果$w(P_1)>w(P_2)$，那么$s(P_1) \leq s(P_2)$。

$n,m \leq 10^6$

题解

有许多种解法，这里介绍一种基于打表的做法，能AC但非正解。

我们先尝试写出一个复杂度非常优秀的搜索算法。

我们用$a_{i,j}$表示(i,j)填的数，$f_{n,m}$表示答案。

直接搜太慢，所以我们先概括一下满足题意的充要条件（改写自如何评价 NOIP2018？ - Wearry的回答 - 知乎）：

1. 对于$i+j$为定值x的所有点(i,j)构成的斜线部分所填数字单调（即上面一段0，下面一段1，当然也可以不存在0或1）。
2. 若$a_{i,j+1}=a_{i+1,j}$，则以$(i+1,j+1)$为左上角的子矩形内斜线部分填的数相同。

据此，我们可以从右下往左上处理每条斜线，枚举$0,1$分界处，然后对于每对$a_{i,j+1}=a_{i+1,j}$，判断以$(i+1,j+1)$为左上角的子矩形是否满足条件。这样，大部分分界处是不符合条件的，可以剪去大量无用的状态，每次搜到最左上角的斜线——$(1,1)$，就可以直接统计答案，复杂度与答案有关。

在判断时在做一些优化，可以在约$10s \sim 20s$的时间内算出$n \leq 8, m \leq 9$或是其他一些范围较小的全部答案。

然后我们通`(运)`过`(用)`观`(人)`察`(类)`发`(智)`现`(慧)`，就可以发现一系列的规律：

1. $f_{n,m}=f_{m,n}$。所以我们可以强制$n<m$。
2. $f_{1,m}=2^m$。所以我们可以特判$n=1$。
3. $f_{n,m}=3 \cdot f_{n,m-1}\ (n>1,m>n+1)$。这点是关键。

根据这三点，于是我们只要用上面的暴力算出当$2 \leq n \leq 8$时$f_{n,n}$和$f_{n,n+1}$的值即可，可以直接推出$n \leq 8, m \leq 10^6$时的所有答案。

因为暴力仍然比较慢，我们可以预先算出我们需要的值，直接用快速幂或直接递推得出答案。

时间复杂度$\mathcal{O}(\log m)$或$\mathcal{O}(m)$。

代码

暴力代码（部分）：

```
#define N 10
int a[N][N];
int len[N << 1], b[N << 1][N][2];
void dfs(int x){
    if (x < 0) return ++ans, void(0);
    for (register int i = 0; i < len[x]; ++i) a[b[x][i][0]][b[x][i][1]] = 1;
    for (register int i = 0; i <= len[x]; ++i){
        if (i > 0) a[b[x][i - 1][0]][b[x][i - 1][1]] = 0;
        bool flag = 1;
        for (register int p = 1; p < len[x]; ++p)
            if (a[b[x][p][0]][b[x][p][1]] == a[b[x][p - 1][0]][b[x][p - 1][1]]){
                for (register int j = n + m - 2; j > x; --j){
                    for (register int k = 1; k < len[j]; ++k)
                        if (b[j][k][0] >= b[x][p][0] && b[j][k][1] > b[x][p][1]
                            && b[j][k - 1][0] >= b[x][p][0] && b[j][k - 1][1] > b[x][p][1]
                            && a[b[j][k][0]][b[j][k][1]] != a[b[j][k - 1][0]][b[j][k - 1][1]])
                            { flag = 0; break; }
                    if (!flag) break;
                }
                if (p <= i - 1 && !flag) return;
                if (!flag) break;
            }
        if (flag) dfs(x - 1);
    }
}
void Main(){
    if (n > m) std :: swap(n, m);
    for (register int i = 0, x, y; i <= n + m - 2; ++i){
        len[i] = 0;
        if (i < m) x = 0, y = i; else x = i - m + 1, y = m - 1;
        for (; x < n && y >= 0; ++x, --y, ++len[i]) b[i][len[i]][0] = x, b[i][len[i]][1] = y;
    }
    ans = 0, dfs(n + m - 2);
    printf("%d\n", ans);
}
```

AC代码（部分）：

```
int qpow(int a, int b){
    int s = 1;
    for (; b; b >>= 1, a = 1ll * a * a % P) if (b & 1) s = 1ll *s * a % P;
    return s;
}
int main(){
    freopen("game.in", "r", stdin);
    freopen("game.out", "w", stdout);
    scanf("%d%d", &n, &m);
    if (n > m) std :: swap(n, m);
    if (n == 1) ans = qpow(2, m);
    if (n == 2) ans = qpow(3, m - 1) * 4ll % P;
    if (n == 3) if (m == n) ans = 112; else ans = 336ll * qpow(3, m - n - 1) % P;
    if (n == 4) if (m == n) ans = 912; else ans = 2688ll * qpow(3, m - n - 1) % P;
    if (n == 5) if (m == n) ans = 7136; else ans = 21312ll * qpow(3, m - n - 1) % P;
    if (n == 6) if (m == n) ans = 56768; else ans = 170112ll * qpow(3, m - n - 1) % P;
    if (n == 7) if (m == n) ans = 453504; else ans = 1360128ll * qpow(3, m - n - 1) % P;
    if (n == 8) if (m == n) ans = 3626752; else ans = 10879488ll * qpow(3, m - n - 1) % P;
    printf("%d\n", ans);
}
```

Day2 T3 保卫王国

已经是相当心力憔悴了。然而我还是得写下这道压（模）轴（板）题的题意和题解。

题意

给定一棵n个点的有点权无根树。

你需要选择若干树上的节点，满足对于树上每条边直接连接的两点中至少有一个被你选择，记费用为你选的点的点权之和。

有m个询问，每次给定两个限制（询问之间是独立的）u, x和v, y，若$x = 0$，则u必须不选；若$x = 1$，则u必须选。v, y同理。对于每个询问，求最小费用。

题解

解法较多。这里介绍基于倍增的在线做法，其他还有动态DP、离线等。

首先考虑最简单的$\mathcal{O}(mn)$的树形DP。

规定1为根。设$f_{u,0/1}$表示以u为根的子树，其中u选/不选时的最小费用。不考虑限制，可以得出以下转移方程：

$$f_{u,x} = \begin{cases} \sum f_{v,1} & \text{if } x = 0 \\ \sum \min\{f_{v,0}, f_{v,1}\} & \text{if } x = 1 \end{cases}$$

考虑限制的话，只要在转移时特判即可。

> 因为插图不方便，请在纸上画图尝试理解以下内容。

我们发现上面的做法主要慢在只改变了两个点的状态，而重新计算整棵树的f值，其实有大量的f是不变的。

设$l = \mathrm{LCA}(u, v)$，我们发现只有$u \to l, v \to l, l \to 1$路径上的$f$值改变了。

那么我们只要考虑维护这三条路径上的f值即可。考虑预处理。

一开始当然要先求出f。在求出f后，我们加入$g_{u,0/1}$数组表示u选/不选时，整棵树去掉以u为根的子树的最小费用。

那么，怎么加速修改f值呢？有了f和g后，我们可以处理出数组$F_{u,j,0/1,0/1}$表示u选/不选，u的2^j祖先选/不选时，以u的2^j祖先为根的子树中去掉以u为根的子树的最小费用。这个数组可以通过枚举u的2^{j-1}祖先的选/不选状态求得。

处理了这些数组以后，你惊喜地发现，可以用倍增算答案了！

若u, v在同一条链上，则直接让深的点向上跳，跳的过程中同样地，枚举当前点和跳上去的点的状态进行转移即可。

否则，先把u, v按链的方法跳到l的两个儿子处，然后枚举l的状态和两个儿子的状态进行转移。

注意两种情况最后都要加上l的g值。

时间复杂度$\mathcal{O}((n + m) \log n)$。

如果理解不了以上内容，可以结合图和代码进行理解。

代码

代码主要部分：

```
void dfs(int u){  // 处理f,fa,dep数组
    f[u][0] = 0, f[u][1] = p[u];
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa[u][0])
            fa[v][0] = u, dep[v] = dep[u] + 1, dfs(v),
            f[u][0] += f[v][1], f[u][1] += dp[v];
    dp[u] = std :: min(f[u][0], f[u][1]);
}
void dfs_g(int u){  // 处理g数组
    if (!fa[u][0]) g[u][0] = g[u][1] = 0;
    for (register int i = hd[u], v; i; i = pr[i])
        if ((v = to[i]) != fa[u][0]){
            g[v][0] = g[u][1] + f[u][1] - dp[v];
            g[v][1] = std :: min(g[u][0] + f[u][0] - f[v][1], g[v][0]);
            dfs_g(v);
        }
}
void pre(){  // 处理F数组
    dep[1] = 1, dfs(1), dfs_g(1);
    for (register int i = 1; i <= n; ++i)
        F[i][0][0][0] = INF, F[i][0][1][0] = f[fa[i][0]][0] - f[i][1],
        F[i][0][1][1] = F[i][0][0][1] = f[fa[i][0]][1] - dp[i];
    for (register int j = 1; j <= 18; ++j)
        for (register int i = 1, k; i <= n; ++i){
            fa[i][j] = fa[k = fa[i][j - 1]][j - 1];
            for (register int x = 0; x <= 1; ++x)
                for (register int y = 0; y <= 1; ++y){
                    F[i][j][x][y] = INF;
                    for (register int z = 0; z <= 1; ++z)
                        F[i][j][x][y] = std :: min(F[i][j][x][y], F[i][j - 1][x][z] + F[k][j - 1][z][y]);
                }
        }
}
long long solve(int u, int x, int v, int y){  // 处理询问部分
    if (dep[u] < dep[v]) std :: swap(u, v), std :: swap(x, y);
    register long long fu[2], fv[2], tu[2], tv[2], k, ans[2];
    fu[x] = f[u][x], fu[!x] = INF, fv[y] = f[v][y], fv[!y] = INF;
    for (register int i = 18; i >= 0; --i)
        if (dep[fa[u][i]] >= dep[v]){
            tu[0] = tu[1] = INF;
            for (register int X = 0; X <= 1; ++X)
                for (register int Y = 0; Y <= 1; ++Y)
                    tu[X] = std :: min(tu[X], fu[Y] + F[u][i][Y][X]);
            fu[0] = tu[0], fu[1] = tu[1], u = fa[u][i];
        }
    if (u == v) return fu[y] + g[u][y];
    for (register int i = 18; i >= 0; --i)
        if (fa[u][i] != fa[v][i]){
            tu[0] = tu[1] = tv[0] = tv[1] = INF;
            for (register int X = 0; X <= 1; ++X)
                for (register int Y = 0; Y <= 1; ++Y)
                    tu[X] = std :: min(tu[X], fu[Y] + F[u][i][Y][X]),
                    tv[X] = std :: min(tv[X], fv[Y] + F[v][i][Y][X]);
            fu[0] = tu[0], fu[1] = tu[1], fv[0] = tv[0], fv[1] = tv[1];
            u = fa[u][i], v = fa[v][i];
        }
    k = fa[u][0];
    ans[0] = f[k][0] - f[u][1] - f[v][1] + fu[1] + fv[1];
    ans[1] = f[k][1] - dp[u] - dp[v] + std :: min(fu[0], fu[1]) + std :: min(fv[0], fv[1]);
    return std :: min(ans[0] + g[k][0], ans[1] + g[k][1]);
    // 无解的情况可以通过判断solve的返回值是否>=INF来判断。
}
```