团队赛第四场

A - 签到

小学数学问题，注意精度

```
void solve() {
    double n, t, m, x, y;
    cin >> n >> t;
    t *= 60;
    cin >> m;
    cin >> x >> y;
    double a = m;
    double b = n - m;
    if ((a / x + b / y) <= t) {
        cout << 0 ;
        return;
    }
    double k = (a / x + b / y - t) / 60;
    int r = 0;
    while (r < k) {
        r += 1;
    }
    cout << r;
}
```

B - 最短路dij

在**堆优化dij**的基础上修改，计算到达**某个点**的路线数量

```
// using ll = long long;
struct node {
    ll to, next, w;
    node() {}
    node(ll w, ll to) : to(to), next(0), w(w) {}
    friend bool operator<(node a, node b) { return a.w > b.w; }
};
auto dij = [&](vector<ll>& tt, vector<Z>& cnt,ll st) {
    vector<bool> vis(n + 1);
    tt[st] = 0;
    priority_queue<node> q;
    cnt[st] = 1;
    q.push({0, st});
    while (!q.empty()) {
        auto [u, _, tot] = q.top();
        q.pop();
        if (vis[u]) {
            continue;
        }
        vis[u] = 1;
        for (int i = h[u]; ~i; i = e[i].next) {
            ll v = e[i].to;
```

```
            ll w = e[i].w;
            if (w + tot <= tt[v]) {
                if (w + tot < tt[v])
                    cnt[v] = 0;
                cnt[v] += cnt[u];
                tt[v] = w + tot;
                q.push({tt[v], v});
            }
        }
    }
};
```

对**起点**和**终点**都做一次 dij 得到两个 **距离数组** 和 **计数数组**
然后对所有边进行遍历，题意说明路径始终是原本存在的一条最短路，那么我们在走偏一次以后一定会回到一条从**终点到该位置**的最短路路径，即把这条边长度设为0时,起点和终点到两端的距离之和为**最短路长度**

```
Z ans = cnt1[t] * (tot == dis1[t]);//取模数自行取模
repi(i, 1, n + 1) {
    if (i == t) {
        continue;
    }
    //枚举边
    for (int j = h[i]; ~j; j = e[j].next) {
        // 起点到i和终点到边的
        if (dis1[i] + e[j].w + dis2[e[j].to] == tot && dis1[t] != tot &&
            dis1[i] + dis2[i] == dis1[t]) {
            ans += cnt1[i] * cnt2[e[j].to];
        }
    }
}
cout << ans;
```

C - 贪心

题意要求所有组合中 **最大与最小的差最小** ，排序后存在 $a_1 + a_i \leq a_1 + a_n \leq a_j + a_n$ 可以推出 $a_1 + a_i \leq a_i + a_j \leq a_j + a_n$ （$i < j$） 即尽可能让 **较大的值跟较小的值** 两两组合，因为 n 为奇数，故我们假设第一个不拿，先将其余的两两组合，放进 $multiset$ ，遍历数组，把以 $a[i]$ 所在组合删去，用 $a[i-1]$ 替代 $a[i]$ ，然后取 $multiset$ 中的最大最小相减

```
// using ll = long long
cin >> n;
vector<ll> a(n);
for(int i = 0;i<n;i++) cin >>a[i];
sort(a.begin(), a.end());
ll ans = INF;
multiset<ll> b;
ll l = 1, r = n - 1;
while (l < r) {
    b.insert(a[l] + a[r]);
    l++;
    r--;
}
```

```
ans = *b.rbegin() - *b.begin();
for(int i=1;i < n;i++) {
    b.erase(b.find(a[n - i - (n / 2 < i)] + a[i]));
    b.insert(a[n - i - (n / 2 < i)] + a[i - 1]);
    ans = min(ans, *b.rbegin() - *b.begin());
}
cout << ans;
```

D - 单调队列/贪心

从直观方面来看，每次都要行驶到一个范围内并且最便宜的站点进行补给，那么我们就可以拿优先队列存可以到达的站点，每次到最便宜的站点加尽可能多的油，然后更新行驶距离，故我们每次计算时，距离按最远部分考虑，对于负数距离说明已经经过这个站点。

```
// using ll = long long
cin >> n >> m >> k;
vector<ll> x(n + 2), p(n + 2);
vector<ll> dir(n + 2);
iota(dir.begin(), dir.end(), 0);
repi(i, 1, n + 1) { cin >> x[i] >> p[i]; }
x[n + 1] = INF;
sort(dir.begin() + 1, dir.end(), [&](auto i, auto j) {
    if (x[i] == x[j]) {
        return p[i] < p[j];
    }
    return x[i] < x[j];
});
ll ans = 0;
priority_queue<Pa, vector<Pa>, greater<Pa>> q;
repi(i, 1, n + 2) {
    // 队首的元素即为可以到达且最便宜的加油站
    while (!q.empty() && q.top().second + k < x[dir[i]]) {
        ll l = q.top().second + k;
        ll ff = q.top().first;
        // 可以加的油和油箱剩余的空间取最小
        ll power = min(l - ans, m / ff);
        // 已经经过这个站点
        if (power <= 0) {
            q.pop();
            continue;
        }
        ans += power;
        m -= power * ff;
        q.pop();
    }
    if (!q.empty()) {
        // 在当前站点和之前站点中选一个最优的
        ll l = min(x[dir[i]], q.top().second + k);
        ll ff = q.top().first;
        ll power = min(l - ans, m / ff);
        ans += power;
        m -= power * ff;
    }
    // 已经到不了这里 直接跳出
```

```
        if (ans < x[dir[i]]) {
            break;
        }
        q.push({p[dir[i]], x[dir[i]]});
    }
    cout << ans;
```

E - 二分答案/签到

对数组排序后，二分 mex 结果即可

```
cin >> n >> m;
vector<ll> a(n + 1);
repi(i, 1, n + 1) { cin >> a[i]; }
sort(a.begin() + 1, a.end());
FOR(m) {
    cin >> x;
    ll l = 0, r = INF, mid;
    while (l <= r) {
        mid = l + r >> 1;
        auto it = upper_bound(a.begin() + 1, a.end(), mid) - a.begin() - 1
        if (mid + 1 - it >= x) {
            r = mid - 1;
        } else {
            l = mid + 1;
        }
    }
    cout << l << ' ';
}
```

F 签到/贪心

首先考虑一种贪心策略。

对于时间 $T = 1, 2, 3 \dots$，若此时有打印机，取一个 $x_i + y_i$ 最小的放进去。因为 $x_i + y_i$ 更大的打印机**有可能**可以放在更后面，这一定比将它放在 T 更优。

但这样做会 TLE。考虑先将打印机对于 x_i 排个序，x_i 相同的按照 $x_i + y_i$ 排序。我们在将 T 扫过去的过程中，很多时间点其实都用不了打印机，白白浪费了时间。所以我们用一个优先队列 q 记录当前等待的打印机，如果 q 空了，就将 T 跳到下一个打印机的可用的起始位置。

```
void solve(){
    cin >>n ;
    vector<Pa> a(n);
    for(auto &[i,j]:a){
        cin >>i>>j;
        j = i+j;
    }
    sort(a.begin(),a.end(),[&](const Pa &a,const Pa &b){
            return a.first<b.first;
    });
    priority_queue<ll,vector<ll>,greater<ll>>q;
    ll ans = 0;
    a.push_back({INF,INF});
```

```
    for (int i = 0; i < n; i++){
        ll j = i,now = a[i].first;
        while (j < n && a[j].first == now) q.push(a[j++].second);
        while (!q.empty() && now < a[j].first){
            ll t = q.top();
            q.pop();
            if (t >= now){
                now++;
                ans++;
            }
        }
        i = j - 1;
    }
    cout << ans;
}
```

G 概率dp

有两类勋章
第一类a种勋章，每种勋章x个活动
第二类b种勋章，每种勋章y个活动

设$f_{i,j}$表示差i种第一类勋章和j种第二类勋章所需要的期望次数，因此$f_{0,0} = 0$

对于枚举到的状态$f_{i,j}$，选到一种新的概率$p = \frac{ix+jy}{n}$，期望次数$E = 1/p$。其中选到第一类的概率是$\frac{ix}{ix+jy}$，选到第二类的概率是$\frac{jy}{ix+jy}$，所以

$f_{i,j} = \frac{ix}{ix+jy} f_{i-1,j} + \frac{jy}{ix+jy} f_{i,j-1} + \frac{n}{ix+jy}$

```
#define repi(i, a, b) for (int i = (int)a; i < (int)b; i++)
void solve(){
    cin >> n >> m;
    ll res1 = m - n % m, res2 = n % m;
    x = n / m;
    y = n / m + 1;
    vector<vector<double>> dp(res1 + 1, vector<double>(res2 + 1));
    repi(u, 1, m + 1){
        repi(i, 0, u + 1){
            int j = u - i;
            if (i>res1||j>res2) continue;
            ll x1 = i * x;
            ll y1 = y * j;
            dp[i][j] = n * 1.0 / (x1 + y1);
            if (i > 0)
                dp[i][j] += dp[i - 1][j] * x1 / (x1 + y1);
            if (j > 0)
                dp[i][j] += dp[i][j - 1] * y1 / (x1 + y1);
        }
    }
    printf("%.8lf\n", dp[res1][res2]);
}
```

H 签到/bfs

首先分析一点：若 a 点坐标为 (x,y)，b 点坐标为 $(x+a,y+b)$，那么 b 点坐标为 $(x+a,y+b)$，a 点坐标就为 $((x+a)-a,(y+b)-b)$。

因此，若有关系 (x,y,a,b)，那么就有关系 $(y,x,-a,-b)$.

而可以通过已知点的坐标与给定的关系推出未知点的坐标。

时间复杂度 $O(N+M)$。

```
struct node{
    ll to, next, w,w2;
    node() {}
    node(ll w, ll to) : to(to), next(0), w(w) {}
    friend bool operator<(node a, node b){
        return a.w > b.w;
    }
};
ll nxt, rnxt;
node e[maxn * 2];
ll h[N];
bool vis[N];
ll dis[N];
void add(ll u,ll v,ll w,ll w2){
    e[++nxt].next=h[u];
    e[nxt].w=w;
    e[nxt].w2=w2;
    e[nxt].to=v;
    h[u] = nxt;
}
void init(){
    nxt=0;
    for (int i = 0; i <= n; i++){
        h[i] = -1;
    }
}
struct pos{
    ll x,y;
};
void solve(){
    cin >>n>>m ;
    init();
    for (int i = 0; i < m; i++){
        cin >>x>>y>>z>>k;
        add(x,y,z,k);
        add(y,x,-z,-k);
    }
    vector<pos> a(n+1);
    a[1].x=a[1].y;
    vector<bool> vis2(2*(m+1));
    vector<bool> vis(n+1);
    queue<ll> q;
    q.push(1);
    vis[1] = 1;
```

```
    vector<bool> ok(n+1);
    ok[1] =1;
    while (!q.empty()){
        ll tt = q.front();
        q.pop();
        for (int i = h[tt];~i; i=e[i].next){
            ll v=  e[i].to,dx=e[i].w,dy=e[i].w2;
            if (!vis[v]){
                vis[v] = 1;
                ok[v] = 1;
                a[v].x=a[tt].x+dx,a[v].y=a[tt].y+dy;
                q.push(v);
            }else {
                if (a[tt].x+dx!=a[v].x||a[tt].y+dy!=a[v].y){
                    ok[v] = 0;
                }
            }
        }
    }
    for (int i = 1; i <= n; i++){
        if (ok[i]) {
            cout << a[i].x<< ' ' <<a[i].y<< '\n';
        }else {
            cout << "undecidable\n";
        }
    }
}
```

I 线段树基础

先考虑如果没有交换操作的情况，对于区间 $[l,r]$，如果这是一个合法的序列串：说明左括号的数量等于右括号，且从左往右左括号的数量一定大于等于右括号的数量。

于是考虑另 s 为字符数组 a 的前缀和数组，则判断条件为 $s_r - s_l - 1 = 0$ 且 $min_{i=l}^{r} s_i \geq s_{l-1}$，

所以我们考虑用线段树维护两个东西，一个是区间 $[l,r]$ 的和，还有一个就是当前区间前缀和的最小值，这里记作 sum 和 Min。

现在考虑加入交换操作，只需要区间加减即可，时间复杂度为 $O(nlogn)$。
参考代码:https://atcoder.jp/contests/abc343/submissions/53748315

J - 树的直径/搜索

题意：构造一条路径可以遍历所有节点一次并且路径**步长不大于2**，验证树为毛毛虫树，即除了直径以外，不存在一条从直径延伸的长度大于2的链。
那么本质上就是在遍历直径的过程中先遍历非直径点,再遍历直径点

$O(nlogn)$ 暴力解法

```
struct node{
    ll to, next, w;
    node() {}
    node(ll w, ll to) : to(to), next(0), w(w) {}
    friend bool operator<(node a, node b){
```

```
        return a.w > b.w;
    }
};
ll nxt, rnxt;
node e[maxn * 2];
ll h[N];
void add(ll u, ll v, ll w = 0){
    e[++nxt].next = h[u];
    e[nxt].w = w;
    e[nxt].to = v;
    h[u] = nxt;
}
void init(){
    nxt = 1;
    for (int i = 0; i <= n; i++) // 网络流使用2*n+7
    {
        h[i] = -1;
    }
}
struct pos{
    ll val,idx,pri;
    bool operator<(const pos&a) const{
        if (val==a.val){
            return pri>a.pri;
        }
        return val>a.val;
    }
};
void solve(){
    cin >> n;
    init();
    ll rt1, rt2;
    vector<ll> dis(n + 1), dis2(n + 1);
    ll Max = 0;
    FOR(n - 1){
        cin >> x >> y;
        add(x, y);
        add(y, x);
    }
    vector<ll> pre(n+1);
    auto dfs = [&](auto self, ll u, ll f) -> void{
        dis[u] = dis[f] + 1;
        if (dis[u] > Max){
            rt1 = u;
            Max = dis[u];
        }
        for (int i = h[u]; ~i; i = e[i].next){
            ll v = e[i].to;
            if (v != f) {
                pre[v] = u;
                self(self, v, u);
            }
        }
    };
    dfs(dfs,1,0);
    rt2 = rt1;
```

```
    Max =0 ;
    ll len =0;
    //   rt A
    dfs(dfs,rt2,0);
    dis2 = dis;
    len = Max;
    rt2 = rt1;
    Max= 0 ;
    pre[rt2] = rt2;
    //rt B
    dfs(dfs,rt2,0);
    ll tt = rt1;
    vector<ll> tr;
    vector<ll> ins(n+1);
    priority_queue<pos> q;
    tr.push_back(tt);
    while (pre[tt]!=rt2){
        tt =pre[tt];
        ins[tt] = 1;
        tr.push_back(tt);
    }
    ins[rt2] = ins[rt1] = 1;
    bool ok =1;
    repi(i,1,n+1){
        if (dis[i]+dis2[i]>len+3){
            ok =0;
        }
    }
    if (!ok){
        NO;
        return;
    }
    ll ed = tr[0];
    // dis2->rt   dis->rt2
    if (len%2==1){
        ed = tr[1];
    }
    repi(i,1,n+1){
        if (dis[i]%2==1){
            q.push({dis[i],i,ins[i]});
        }
    }
    vector<ll> ans;
    while (!q.empty()){
        ans.push_back(q.top().idx);
        q.pop();
    }
    queue<ll> qq;
    dis[ed] = 1;
    qq.push(ed);
    vector<bool> vis(n+1);
    vis[ed] =1;
    while (!qq.empty()){
        ll u = qq.front();
        qq.pop();
        if (dis[u]%2==1){
```

```
            q.push({dis[u],u,ins[u]});
        }
        for (int i = h[u]; ~i; i=e[i].next){
            ll v=e[i].to;
            if (!vis[v]){
                qq.push(v);
                vis[v] =1;
                dis[v] = dis[u]+1;
            }
        }
    }
    while (!q.empty()){
        ans.push_back(q.top().idx);
        q.pop();
    }
    YES;
    for(auto i:ans){
        cout << i<< ' ' ;
    }
    cout << endl;
}
```

$O(n)$ 解法

```
struct node{
    ll to, next, w;
    node() {}
    node(ll w, ll to) : to(to), next(0), w(w) {}
    friend bool operator<(node a, node b){
        return a.w > b.w;
    }
};
ll nxt, rnxt;
node e[maxn * 2];
ll h[N];
void add(ll u, ll v, ll w = 0){
    e[++nxt].next = h[u];
    e[nxt].w = w;
    e[nxt].to = v;
    h[u] = nxt;
}
void init(){
    nxt = 1;
    for (int i = 0; i <= n; i++) // 网络流使用2*n+7
    {
        h[i] = -1;
    }
}
void solve(){
    string ss;
    cin >> n;
    init();
    ll rt1, rt2;
    vector<ll> dis(n + 1), dis2(n + 1);
    ll Max = 0;
```

```
    vector<ll>siz(n+1);
    FOR(n - 1){
        cin >> x >> y;
        siz[x]++;
        siz[y]++;
        add(x, y);
        add(y, x);
    }
    vector<ll> pre(n+1);
    auto dfs = [&](auto self, ll u, ll f) -> void{
        dis[u] = dis[f] + 1;
        if (dis[u] > Max){
            rt1 = u;
            Max = dis[u];
        }
        for (int i = h[u]; ~i; i = e[i].next){
            ll v = e[i].to;
            if (v != f) {
                pre[v] = u;
                self(self, v, u);
            }
        }
    };
    dfs(dfs,1,0);
    rt2 = rt1;
    Max =0 ;
    ll len =0;
    dfs(dfs,rt2,0);
    dis2 = dis;
    len = Max;
    rt2 = rt1;
    Max= 0 ;
    pre[rt2] = rt2;
    dfs(dfs,rt2,0);
    bool ok =1;
    repi(i,1,n+1){
        if (dis[i]+dis2[i]>len+3){
            ok =0;
        }
    }
    if (!ok){
        NO;
        return;
    }
    vector<ll> ans;
    ans.push_back(rt1);
    auto dfs2=[&](auto self,ll u,ll f)->void{
        vector<ll> son;
        for (int i = h[u]; ~i; i=e[i].next){
            ll v=e[i].to;
            if (v!=f){
                if (siz[v]>1){
                    for (int r = h[v]; ~r; r=e[r].next){
                        ll v2 = e[r].to;
                        if (siz[v2]==1&&v2!=u){
                            ans.push_back(v2);
```

```
                        }
                    }
                    for (int r = h[v]; ~r; r = e[r].next){
                        ll v2 = e[r].to;
                        if (siz[v2]>1&&v2!=u){
                            ans.push_back(v2);
                            self(self,v2,v);
                        }
                    }
                    ans.push_back(v);
                }else {
                    son.push_back(v);
                }
            }
        }
        for(auto i:son){
            ans.push_back(i);
        }
    };
    dfs2(dfs2,rt1,0);
    YES;
    for(auto i:ans){
        cout << i<< ' ' ;
    }
}
```

K - 调和级数背包dp

我的博客https://www.luogu.com.cn/article/eng9omcw

L - 单调栈/ST表

我的博客https://www.luogu.com.cn/article/puuw8cmf

M - 二分答案/哈希

如果一个字符串 s 是只出现一次的字符串，那么他的子串至少出现一次，故可以二分出现的长度，用哈希比较等于

```
struct hash_val{
    vector<unsigned long long>has1,has2;
    vector<unsigned long long>base1,base2;
    int p1=131,p2=13331;
    hash_val(string s) {
        int n=s.length();
        has1.resize(n+1,0);
        has2.resize(n+1,0);
        base1.resize(n+1,1);
        base2.resize(n+1,1);
        for (int i=1;i<=n;i++) {
            has1[i]=has1[i-1]*p1+s[i-1];
            has2[i]=has2[i-1]*p2+s[i-1];
            base1[i]=base1[i-1]*p1;
            base2[i]=base2[i-1]*p2;
```

```
        }
    }
    ull calchas1(int l,int r) {
        return has1[r]-base1[r-l+1]*has1[l-1];
    }
    ull calchas2(int l,int r) {
        return has2[r]-base2[r-l+1]*has2[l-1];
    }
    bool same(int l1,int r1,int l2,int r2) {
        return
calchas1(l1,r1)==calchas1(l2,r2)&&calchas2(l1,r1)==calchas2(l2,r2);
    }
};
void solve(){
    cin >>ss;
    hash_val has(ss);
    n = ss.length();
    ll l = 1,r = ss.length(),mid,ans;
    ll st = 0;
    auto check = [&](ll mid)->bool{
        unordered_map<ull,Pa> mm;
        repi(i,1,n-mid+2){
            ull tt = has.calchas1(i,i+mid-1);
            mm[tt].first=i;
            mm[tt].second++;

        }
        repi(i,1,n+2-mid){
            ull tt =has.calchas1(i,i+mid-1);
            ll cnt = mm[tt].second;
            if (cnt==1){
                st = i;
                return 1;
            }
        }
        return 0;
    };
    while (l<=r)
    {
        mid = l+r>>1;
        if (check(mid)){
            ans = mid;
            r = mid-1;
        }else {
            l = mid+1;
        }
    }
    repi(i,st-1,st-1+ans){
        cout << ss[i];
    }
    cout << endl;
}
```