树哈希

模板题链接：https://uoj.ac/problem/763

给定一棵以点 1 为根的树，你需要输出这棵树中最多能选出多少个互不同构的子树。

```
#include<bits/stdc++.h>
using namespace std;

#define debug(x) cerr << #x << ": " << (x) << endl
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define pb push_back
using ull = unsigned long long;
const int N=1e6+5;

int n;
vector<int> g[N];
set<ull> res;

mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
ull B=rnd();

ull hs(ull x){
    return x*x*x*19890535+19260817;
}

ull f(ull x){
    return hs(x&((1LL<<32)-1))+hs(x>>32);
}

ull h[N];

void dfs(int u, int fa){
    h[u]=B;
    for(auto go: g[u]) if(go!=fa) dfs(go, u), h[u]+=f(h[go]);
    res.insert(h[u]);
}

signed main(){
    cin>>n;
    if(~B&1) B++;
    rep(i, 1, n-1){
        int u, v; scanf("%d %d", &u, &v);
        g[u].pb(v), g[v].pb(u);
    }
    dfs(1, 0);
    cout<<res.size()<<endl;
    return 0;
}
```

欧拉路径

```cpp
// Problem: P7771 【模板】欧拉路径
// Contest: Luogu
// URL: https://www.luogu.com.cn/problem/P7771
// Memory Limit: 128 MB
// Time Limit: 1000 ms
//
// Powered by CP Editor (https://cpeditor.org)

#include<bits/stdc++.h>
using namespace std;

#define debug(x) cerr << #x << ": " << (x) << endl
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define dwn(i,a,b) for(int i=(a);i>=(b);i--)
#define pb push_back
#define all(x) (x).begin(), (x).end()

#define x first
#define y second
using pii = pair<int, int>;
using ll = long long;

inline void read(int &x){
    int s=0; x=1;
    char ch=getchar();
    while(ch<'0' || ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0' && ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=1e5+5, M=2e5+5;

int n, m;
vector<pii> g[N]; // x: to, y: index of edge
int din[N], dout[N];
int S, T;
int cur[N];
bool vis[M];

stack<int> res;

void dfs(int u){
    for(int i=cur[u]; i<g[u].size(); i=max(i+1, cur[u])){
        auto [go, id]=g[u][i];
        if(!vis[id]){
            vis[id]=true;
            cur[u]=i+1;
```

```
            dfs(go);
        }
    }
    res.push(u);
}

signed main(){
    cin>>n>>m;
    rep(i, 1, m){
        int u, v; read(u), read(v);
        dout[u]++, din[v]++;
        g[u].pb({v, i});
    }

    int chk=0;
    rep(i, 1, n){
        if(din[i]!=dout[i]){
            chk++;
            if(din[i]+1==dout[i]) S=i;
            if(dout[i]+1==din[i]) T=i;
        }
    }
    if(!(!chk || chk==2)) return puts("No"), 0;
    if(!chk) S=T=1;
    rep(i, 1, n) sort(all(g[i]));
    dfs(S);
    while(res.size()){
        cout<<res.top()<<' ';
        res.pop();
    }

    return 0;
}
```

DP

悬线法

```
#include<bits/stdc++.h>
using namespace std;

const int N=1005;
int n,m;
bool g[N][N];
int l[N][N],r[N][N],u[N][N];

void init(){
    // init l[][]
    for(int i=1;i<=n;i++)
        for(int j=2;j<=m;j++)
            if(g[i][j-1]) l[i][j]=l[i][j-1];
```

```
    // init r[][]
    for(int i=1;i<=n;i++)
        for(int j=m-1;j;j--)
            if(g[i][j+1]) r[i][j]=r[i][j+1];

    // init u[][]
    for(int i=2;i<=n;i++)
        for(int j=1;j<=m;j++)
            if(g[i-1][j]) u[i][j]=u[i-1][j]+1;
}

int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++){
            char ch; cin>>ch;
            g[i][j]= ch=='F'?1:0;
            l[i][j]=r[i][j]=j;
            u[i][j]=1;
        }

    init(); // 预处理出每个点能够向左、向右、走到的最远点对应的位置,以及向上走的最远距离。

    int res=0;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(!g[i][j]) continue;
            if(i>1 && g[i-1][j]){
                l[i][j]=max(l[i][j],l[i-1][j]), r[i][j]=min(r[i][j],r[i-1][j]);
            }
            res=max(res,u[i][j]*(r[i][j]-l[i][j]+1));
        }
    }

    cout<<res*3<<endl;

    return 0;
}
```

斜率优化

例题：

N 个高度递增的石头（高度记为 h），青蛙每次可以从 $i \to j$（$j > i$），花费为 $(h_j - h_i)^2 + C$，求青蛙 $1\to n$ 的最小花费。

解答：

斜率优化 dp。

记 $f(i)$ 为跳到 i 的最小花费。

~~显然~~，转移方程：$f(i) = \min (f(j)+(h_i-h_j)^2) + c$，其中 $j\in[1, i-1]$。

我们假设决策点为 j，那么有 $f(i) = f(j) + h_i^2 - 2h_ih_j + h_j^2 + c$。

进一步将上式化为：$f(j) + h_j^2 = 2h_ih_j + f(i) + c - h_i^2$

我们记 $y = f(j) + h_j^2, k=2h_i,x=h_j, b=f(i)+c-h_i^2$。

那么这正好为 $y = kx + b$ 的形式。

我们的目标是最小化 $f(i)$，这等价于最小化 b，也就是从维护的下凸壳中找到使直线的 b 最小的点即可。

```
const int N=2e5+5;

int n, c;
int h[N];
int q[N], l, r;
int f[N];

int Y(int i){return f[i]+h[i]*h[i];}
int X(int i){return h[i];}

bool remove(int a, int b, int k){
    return k*(X(b)-X(a))>Y(b)-Y(a);
}

bool insert(int x, int y, int z){
    return (Y(y)-Y(x))*(X(z)-X(x))>(X(y)-X(x))*(Y(z)-Y(x));
}

signed main(){
    cin>>n>>c;
    rep(i,1,n) read(h[i]);

    l=0, r=-1;
    q[++r]=1;
    f[1]=0;

    rep(i,2,n){
        while(r>l && remove(q[l], q[l+1], 2*h[i])) l++;
        f[i]=f[q[l]]+(h[i]-h[q[l]])*(h[i]-h[q[l]])+c;
        while(r>l && insert(q[r-1], q[r], i)) r--;
        q[++r]=i;
    }

    cout<<f[n]<<endl;

    return 0;
}
```

四边形不等式优化

例题 1

高速公路旁边有一些村庄。高速公路表示为整数轴，每个村庄的位置用单个整数坐标标识。没有两个在同样地方的村庄。两个位置之间的距离是其整数坐标差的绝对值。

邮局将建在一些，但不一定是所有的村庄中。为了建立邮局，应选择他们建造的位置，使每个村庄与其最近的邮局之间的距离总和最小。

你要编写一个程序，已知村庄的位置和邮局的数量，计算每个村庄和最近的邮局之间所有距离的最小可能的总和。

输入：

第一行包含两个整数：第一个是村庄 V 的数量，第二个是邮局的数量 P。

第二行包含 V 个整数。这些整数是村庄的位置。

输出：

第一行包含一个整数S，它是每个村庄与其最近的邮局之间的所有距离的总和。

对于 100% 的数据，$1 \leq P \leq 300$，$P \leq V \leq 3000$，$1 \leq $ 村庄位置 ≤ 10000。

```
// Problem: 邮局
// Contest: AcWing
// URL: https://www.acwing.com/problem/content/338/
// Memory Limit: 10 MB
// Time Limit: 1000 ms
//
// Powered by CP Editor (https://cpeditor.org)

#include<bits/stdc++.h>
using namespace std;

#define debug(x) cerr << #x << ": " << (x) << endl
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define dwn(i,a,b) for(int i=(a);i>=(b);i--)
#define pb push_back
#define all(x) (x).begin(), (x).end()

#define x first
#define y second
using pii = pair<int, int>;
using ll = long long;

inline void read(int &x){
    int s=0; x=1;
    char ch=getchar();
    while(ch<'0' || ch>'9') {if(ch=='-')x=-1;ch=getchar();}
    while(ch>='0' && ch<='9') s=(s<<3)+(s<<1)+ch-'0',ch=getchar();
    x*=s;
}

const int N=3300;

int n, p;
int w[N], s[N];
int c[N][N];
int f[2][N];

void get_c(){
```

```
    rep(i,1,n){
        rep(j,i,n){
            int mid=i+j>>1;
            c[i][j]=((mid-i+1)*w[mid]-(s[mid]-s[i-1]))+((s[j]-s[mid])-(j-mid)*w[mid]);
        }
    }
}

void divi(int *f, int *g, int l, int r, int lo, int ro){
    if(l>r || lo>ro) return;
    int mid=l+r>>1;
    pii best={2e9, -1};
    rep(k,lo,min(ro, mid)) best=min(best, {f[k-1]+c[k][mid], k});
    g[mid]=best.x;
    int idx=best.y;
    divi(f, g, l, mid-1, lo, idx);
    divi(f, g, mid+1, r, idx, ro);
}

signed main(){
    cin>>n>>p;
    rep(i,1,n) read(w[i]);
    sort(w+1, w+1+n);
    rep(i,1,n) s[i]=w[i]+s[i-1];

    get_c();

    memset(f, 0x3f, sizeof f);
    rep(i,1,n) f[1][i]=c[1][i];

    rep(i,2,p){
        divi(f[i-1&1], f[i&1], 1, n, 1, n);
    }
    cout<<f[p&1][n]<<endl;

    return 0;
}
```

例题 2：

小 G 给每首诗定义了一个行标准长度（行的长度为一行中符号的总个数），他希望排版后每行的长度都和行标准长度相差不远。

显然排版时，不应改变原有的句子顺序，并且小 G 不允许把一个句子分在两行或者更多的行内。

在满足上面两个条件的情况下，小 G 对于排版中的每行定义了一个不协调度，为这行的实际长度与行标准长度差值绝对值的 P 次方，而一个排版的不协调度为所有行不协调度的总和。

小 G 最近又作了几首诗，现在请你对这几首诗进行排版，使得排版后的诗尽量协调。

```
#include<bits/stdc++.h>
using namespace std;

using ld=long double;
```

```cpp
const int N=1e5+5;

int n, L, P;
char str[N][35];
int opt[N];
int s[N];
ld f[N];
struct Node{
    int j, l, r;
}q[N];
int tt, hh;

ld val(int j, int i){
    ld res=1, t=abs(s[i]-s[j]-L+i-j-1); // j+1 -- i
    for(int i=0; i<P; i++) res*=t;
    return res+f[j];
}

void insert(int i){
    int pos=n+1;
    // 更新队尾
    // 弹出队尾坏的元素
    while(tt>=hh && val(q[tt].j, q[tt].l)>=val(i, q[tt].l)) pos=q[tt--].l;
    // 从当前队尾二分出一段应该更新的区间
    if(tt>=hh && val(q[tt].j, q[tt].r)>=val(i, q[tt].r)){
        int l=q[tt].l, r=q[tt].r;
        while(l<r){
            int mid=l+r>>1;
            if(val(q[tt].j, mid)>=val(i, mid)) r=mid;
            else l=mid+1;
        }
        pos=r;
        q[tt].r=pos-1;
    }
    if(pos!=n+1) q[++tt]={i, pos, n};
}

int main(){
    // ios::sync_with_stdio(false);
    int T; cin>>T;
    while(T--){
        cin>>n>>L>>P;
        for(int i=n; i; i--) cin>>str[i];
        for(int i=1; i<=n; i++) s[i]=s[i-1]+strlen(str[i]);

        tt=-1, hh=0;
        q[++tt]={0, 1, n};
        for(int i=1; i<=n; i++){
            f[i]=val(q[hh].j, i), opt[i]=q[hh].j;
            // 更新队头
            if(q[hh].r==i) hh++;
            q[hh].l=i+1;
```

```
            insert(i);
        }

        if(f[n]>1e18){
            puts("Too hard to arrange");
            puts("--------------------");
            continue;
        }
        cout<<(long long)f[n]<<endl;
        for(int i=n; i; i=opt[i]){
            for(int j=i; j>opt[i]; j--){
                cout<<str[j];
                if(j!=opt[i]+1) cout<<' ';
            }
            puts("");
        }
        puts("--------------------");
    }
    return 0;
}
```