Содержание

1 Теория чисел

1.1 КТО

```
int gcd(int a, int b, int &x, int &y) {
    if (b==0) { x = 1; y = 0; return a; }
    int d = gcd(b,a%b,x,y);
    swap(x,y);
    y-=a/b*x;
    return d;
}
int inv(int r, int m) {
    int x, y;
    gcd(r,m,x,y);
    return (x+m)%m;
}
int crt(int r, int n, int c, int m) { return r + ((c - r) % m + m) * inv(n, m) % m * n; }
```

1.2 Алгоритм Миллера — Рабина

```
__int128 one=1;
int po(int a,int b,int p)
{
    int res=1;
    while(b) {if(b & 1) {res=(res*one*a)%p;--b;} else {a=(a*one*a)%p;b>>=1;}} return res;
}
bool chprime(int n) ///miller-rabin
{
    if(n==2) return true;
    if(n<=1 || n%2==0) return false;
    int h=n-1;int d=0;while(h%2==0) {h/=2;++d;}
    for(int a:{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37})
        {
        if(a==n) return true;
        int u=po(a,h,n);bool ok=0;
        if(u%n==1) continue;
        for(int c=0;c<d;++c)
            {
            if((u+1)%n==0) {ok=1;break;}
            u=(u*one*u)%n;
        }
        if(!ok) return false;
    }
    return true;
}
```

2 Графы

2.1 2-$\mathcal{SAT}$

Алгоритм ищет сильносвязные компоненты в графе g, если есть путь $i \to j$, то $scc[i] \leq scc[j]$

В случае 2-$\mathcal{SAT}$ рёбра $i \Rightarrow j$ и $(j \oplus 1) \Rightarrow (i \oplus 1)$ должны быть добавлены одновременно.

```
vector<vector<int>> g(2 * n);
vector<vector<int>> r(g.size());
for (int i = 0; i < g.size(); ++i) {
    for (int j : g[i]) r[j].push_back(i);
}
vector<int> used(g.size()), tout(g.size());
int time = 0;
auto dfs = [&](auto dfs, int cur) -> void {
    if (used[cur]) return;
    used[cur] = 1;
    for (int nxt : g[cur]) {
        dfs(dfs, nxt);
    }
    // used[cur] = 2;
    tout[cur] = time++;
};
for (int i = 0; i < g.size(); ++i) if (!used[i]) dfs(dfs, i);
vector<int> ind(g.size());
iota(ind.begin(), ind.end(), 0);
```

```
sort(all(ind), [&](int i, int j){return tout[i] > tout[j];});
vector<int> scc(g.size(), -1);
auto go = [&](auto go, int cur, int color) -> void {
    if (scc[cur] != -1) return;
    scc[cur] = color;
    for (int nxt : r[cur]) {
        go(go, nxt, color);
    }
};
int color = 0;
for (int i : ind) {
    if (scc[i] == -1) go(go, i, color++);
}
for (int i = 0; i < g.size() / 2; ++i) {
    if (scc[2 * i] == scc[2 * i + 1]) "IMPOSSIBLE"
    if (scc[2 * i] < scc[2 * i + 1]) {
        // !i => i, assign i = true
    } else {
        // i => !i, assign i = false
    }
}
```

2.2 Эйлеров цикл

```
vector<vector<pair<int, int>>> g(n); // pair{nxt, idx}
vector<pair<int, int>> e(p.size());
// build graph
vector<int> in(n), out(n);
for (auto [u, v] : e) in[v]++, out[u]++;
vector<int> used(m), it(n), cycle;
auto dfs = [&](auto dfs, int cur) -> void {
    while (true) {
        while (it[cur] < g[cur].size() && used[g[cur][it[cur]].second]) it[cur]++;
        if (it[cur] == g[cur].size()) return;
        auto [nxt, idx] = g[cur][it[cur]];
        used[idx] = true;
        dfs(dfs, nxt);
        cycle.push_back(idx);
    }
};
int cnt = 0, odd = -1;
for (int i = 0; i < n; ++i){
    if (out[i] && odd == -1) odd = i;
    if (in[i] != out[i]) {
        if (in[i] + 1 == out[i]) odd = i;
        if (abs(in[i] - out[i]) > 1) return {}; // must hold
        cnt++;
    }
}
if (cnt != 0 && cnt != 2) return {}; // must hold
// for undirected find odd vertex (and count that # of odd is 0 or 2)
dfs(dfs, odd);
reverse(cycle.begin(), cycle.end());
if (cycle.size() != m) return {};
```

3 xor, and, or-свёртки

3.1 and-свёртка

```
vector<int> band(vector<int> a,vector<int> b)
{
    int n=0;while((1<<n)<a.size()) ++n;
    a.resize(1<<n);b.resize(1<<n);
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(mask & (1<<i)) {a[mask-(1<<i)]+=a[mask];a[mask-(1<<i)]%=p;}
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(mask & (1<<i)) {b[mask-(1<<i)]+=b[mask];b[mask-(1<<i)]%=p;}
    vector<int> c(1<<n,0);
    for(int mask=0;mask<(1<<n);++mask) {c[mask]=a[mask]*b[mask];c[mask]%=p;}
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(!(mask & (1<<i))) {c[mask]-=c[mask+(1<<i)];c[mask]%=p;}
    return c;
}
```

3.2 or-свёртка

```
vector<int> bor(vector<int> a,vector<int> b)
{
    int n=0;while((1<<n)<a.size()) ++n;
    a.resize(1<<n);b.resize(1<<n);
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(!(mask & (1<<i))) {a[mask+(1<<i)]+=a[mask];a[mask+(1<<i)]%=p;}
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(!(mask & (1<<i))) {b[mask+(1<<i)]+=b[mask];b[mask+(1<<i)]%=p;}
    vector<int> c(1<<n,0);
    for(int mask=0;mask<(1<<n);++mask) {c[mask]=a[mask]*b[mask];c[mask]%=p;}
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(mask & (1<<i)) {c[mask]-=c[mask-(1<<i)];c[mask]%=p;}
    return c;
}
```

3.3 xor-свёртка

```
vector<int> bxor(vector<int> a,vector<int> b)
{
    assert(p%2==1);int inv2=(p+1)/2;
    int n=0;while((1<<n)<a.size()) ++n;
    a.resize(1<<n);b.resize(1<<n);
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(!(mask & (1<<i))) {int u=a[mask],v=a[mask+(1<<i)];a[mask+(1<<i)]=(u+v)%p;a[mask]=(u-v)%p;}
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(!(mask & (1<<i))) {int u=b[mask],v=b[mask+(1<<i)];b[mask+(1<<i)]=(u+v)%p;b[mask]=(u-v)%p;}
    vector<int> c(1<<n,0);
    for(int mask=0;mask<(1<<n);++mask) {c[mask]=a[mask]*b[mask];c[mask]%=p;}
    for(int i=0;i<n;++i) for(int mask=0;mask<(1<<n);++mask) if(!(mask & (1<<i))) {int u=c[mask],v=c[mask+(1<<i)];c[mask+(1<<i)]=((v-u)*inv2)%p;c[mask]=((u+v)*inv2)%p;}
    return c;
```

4 Структуры данных

4.1 Дерево Фенвика

```
int fe[maxn]; /// fenwick tree
void pl(int pos,int val) {while(pos<maxn) {fe[pos]+=val;pos|=(pos+1);}}
int get(int pos) {int ans=0;while(pos>=0) {ans+=fe[pos];pos&=(pos+1);--pos;} return ans;} /// [0, pos] - vkluchitelno!!!
int get(int l,int r) {return get(r-1)-get(l-1);} /// summa na [l,r)
```

4.2 Ordered set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
using namespace std;

using ordered_set = tree<int, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update>;
```