Code Template

There is no gay @ CCNUACM

Central China Normal University

April 25, 2024

Contents

0-杂项

快读

```cpp
inline int read()
{
    int x = 0, w = 1;
    char ch = 0;
    while (ch < '0' || ch > '9')
    {
        ch = getchar();
        if (ch == '-')
        {
            w = -1;
        }
    }
    while (ch >= '0' && ch <= '9')
    {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    return x * w;
}
```

judge.sh

Assume using directory “./contest”.

```
../contest:
a.cpp  b.cpp  samples-a  samples-b  judge.sh

../contest/samples-a:
1.ans  1.in

../contest/samples-b:
1.ans  1.in  2.ans  2.in
```

judge.sh

```bash
#!bin/bash
set -e
[ $# == 2 ] || { echo invalid args ; exit 1 ; }
g++ $2.cpp || { echo CE ; exit 1 ; }
src=./samples-$1
dir=$1-test
mkdir -p $dir
cp $src/* $dir/
cd $dir
mv ../a.out ./$2
for input in *.in; do
  [ $input == "*.in" ] && exit 0
  cas=${input%.in}
  output=$cas.out
  answer=$cas.ans
  timeout 1 ./$2 < $input > $output 2> $cas.err || { echo Case $cas : TLE or RE ; continue ; }
  if diff -Za $output $answer > $cas.dif ; then
    echo Case $cas : AC
  else
    echo Case $cas : WA
    cat $cas.dif $cas.err
  fi
done
```

command:

```bash
cd ./contest
bash judge.sh a a.cpp
```

心态崩了

- `(int)v.size()`
- `1LL << k`
- 递归函数用全局或者 static 变量要小心
- 预处理组合数注意上限
- 想清楚到底是要 `multiset` 还是 `set`
- 提交之前看一下数据范围，测一下边界
- 数据结构注意数组大小（2 倍，4 倍）
- 字符串注意字符集
- 如果函数中使用了默认参数的话，注意调用时的参数个数。
- 注意要读完
- 构造参数无法使用自己
- 树链剖分/dfs 序，初始化或者询问不要忘记 idx, ridx
- 排序时注意结构体的所有属性是不是考虑了
- 不要把 while 写成 if
- 不要把 int 开成 char
- 清零的时候全部用 0~n+1。
- 模意义下不要用除法
- 哈希不要自然溢出
- 最短路不要 SPFA，乖乖写 Dijkstra
- 上取整以及 GCD 小心负数
- mid 用 `l + (r - l) / 2` 可以避免溢出和负数的问题
- 小心模板自带的意料之外的隐式类型转换
- 求最优解时不要忘记更新当前最优解
- 图论问题一定要注意图不连通的问题
- 处理强制在线的时候 lastans 负数也要记得矫正
- 不要觉得编译器什么都能优化
- 分块一定要特判在同一块中的情况

sol.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using VI = vector<ll>;
using PII = pair<int, int>;
template <typename T> using fc = function<T>;
using Graph = vector<vector<int>>;
#define pb push_back
#define debug(c) cerr << #c << " = " << c << endl;
#define rg(x) x.begin(), x.end()
#define rep(a, b, c) for (auto a = (b); (a) < (c); a++)
#define repe(a, b, c) for (auto a = (b); (a) <= (c); a++)
const int MOD = 998244353;
const int N = 0;
#ifdef ONLINE_JUDGE
#define cerr assert(false);
#endif

void solve()
{

}

int main()
{
    std::ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int T = 1;
    cin >> T;
    while (T--)
        solve();

    return 0;
}
```

三分

```
double cal()
{
    double l = 0, r = 1e10;
    for (int i = 1; i <= 100; i++)
    {
        double m1 = l + (r - l) / 3;
        double m2 = (r - l) / 3 * 2 + l;
        if (f(m1) < f(m2))
            l = m1;
        else
            r = m2;
    } // 求最大值
    return f(l);
}

int cal()
{
    int l = 0, r = 1e10;
    while (l + 2 < r)
    {
        int m1 = l + (r - l) / 3;
        int m2 = (r - l) / 3 * 2 + l;
        if (f(m1) < f(m2))
        {
            l = m1;
        }
        else
        {
            r = m2;
        }
    }
    int ans = f(l);
    for (int i = l + 1; i <= r; i++)
    {
        ans = max(ans, f(i));
    }
}
```

1-字符串

KMP

```
int f[N];
void kmp(string s, string p)
{
    p += '@';
    p += s;
    for (int i = 1; i < p.size(); i++)
    {
        int j = f[i - 1];
        while (j && p[j] != p[i])
            j = f[j - 1];
        if (p[j] == p[i])
            f[i] = j + 1;
    }
}
```

Manacher

```
#include <cstdio>
using namespace std;
```

```cpp
using ll = long long;
// !!! N = n * 2, because you need to insert '#' !!!
const int N = 3e7;
#define min(A, B) ((A > B) ? B : A)
// p[i]: range of the palindrome i-centered.
int p[N];
// s: the string.
char s[N] = "@#";
// l: length of s.
int l = 2;

int main()
{
    char tmp = getchar();
    while (tmp > 'z' || tmp < 'a')
        tmp = getchar();
    while (tmp <= 'z' && tmp >= 'a')
        s[l++] = tmp, s[l++] = '#', tmp = getchar();
    /*<--- input & preparation --->*/
    int m = 0, r = 0;
    ll ans = 0;
    for (int i = 1; i < l; i++)
    {
        // evaluate p[i]
        if (i <= r)
            p[i] = min(p[m * 2 - i], r - i + 1);
        else
            p[i] = 1;
        // brute force!
        while (s[i - p[i]] == s[i + p[i]])
            ++p[i];
        // maintain m, r
        if (i + p[i] > r)
        {
            r = i + p[i] - 1;
            m = i;
        }
        // find the longest p[i]
        if (p[i] > ans)
            ans = p[i];
    }
    printf("%lld", ans - 1);

    return 0;
}

```

hash

随机素数表

42737, 46411, 50101, 52627, 54577, 191677, 194869, 210407, 221831, 241337, 578603, 625409, 713569, 788813, 862481, 2174729, 2326673, 2688877, 2779417, 3133583, 4489747, 6697841, 6791471, 6878533, 7883129, 9124553, 10415371, 11134633, 12214801, 15589333, 17148757, 17997457, 20278487, 27256133, 28678757, 38206199, 41337119, 47422547, 48543479, 52834961, 76993291, 85852231, 95217823, 108755593, 132972461, 171863609, 173629837, 176939899, 207808351, 227218703, 306112619, 311809637, 322711981, 330806107, 345593317, 345887293, 362838523, 373523729, 394207349, 409580177, 437359931, 483577261, 490845269, 512059357, 534387017, 698987533, 764016151, 906097321, 914067307, 954169327

1572869, 3145739, 6291469, 12582917, 25165843, 50331653 （适合哈希的素数）

```cpp
#include <string>
using ull = unsigned long long;
using std::string;
const int mod = 998244353;

int n;
ull Hash[2000200]; // 自然溢出法用 unsigned 类型
ull RHash[2000200];
ull base[2000200];
void init()
```

```cpp
{
    base[0] = 1;
    for (int i = 1; i <= 2000010; i++)
    {
        base[i] = base[i - 1] * 131 % mod;
    }
}
void get_hash(string s)
{
    for (int i = 1; i <= (int)s.size(); i++)
    {
        Hash[i] = Hash[i - 1] * base[1] % mod + s[i - 1];
        Hash[i] %= mod;
    }
}
void get_Rhash(string s)
{
    for (int i = (int)s.size(); i >= 1; i--)
    {
        RHash[s.size() - i + 1] = RHash[s.size() - i] * base[1] % mod + s[i - 1];
        RHash[i] %= mod;
    }
}
ull getR(int l, int r)
{
    if (l > r)
        return 0;
    return (RHash[r] - (RHash[l - 1] * base[r - l + 1]) % mod + mod) % mod;
}
ull get(int l, int r)
{
    if (l > r)
        return 0;
    return (Hash[r] - (Hash[l - 1] * base[r - l + 1]) % mod + mod) % mod;
}
```

字典树

```cpp
#include <string>
using std::string;
/* Last modified: 23/07/03 */
// Trie for string and prefix
class Trie
{

    static const int trie_tot_size = 1e5;
    // trie_node_size: modify if get() is modified.
    static const int trie_node_size = 64;
    int tot = 0;
    // end: reserved for count
    const int end = 63;
    int (*nxt)[trie_node_size];

  public:
    Trie()
    {
        nxt = new (int[trie_tot_size][trie_node_size]);
    }
    int get(char x)
    {
        // modify if x is in certain range, assuming 0-9 or a-z.
        if (x >= 'A' && x <= 'Z')
            return x - 'A';
        else if (x >= 'a' && x <= 'z')
            return x - 'a' + 26;
        else
            return x - '0' + 52;
    }
    int find(string s)
    {
        int cnt = 0;
        for (auto i : s)
        {
            cnt = nxt[cnt][get(i)];
            if (!cnt)
                return 0;
        }
        return cnt;
    }
    void insert(string s)
    {
        int cnt = 0;
        for (auto i : s)
        {
            auto j = get(i);
            // count how many strings went by
            nxt[cnt][end]++;
            if (nxt[cnt][j] > 0)
                // character i already exists.
                cnt = nxt[cnt][j];
            else
            {
                // doesn't exist, new node.
                nxt[cnt][j] = ++tot;
                cnt = tot;
            }
        }
        nxt[cnt][end]++;
    }
    int count(string s)
    {
        int cnt = find(s);
        if (!cnt)
            return 0;
        return nxt[cnt][end];
    }
    void clear()
    {
        for (int i = 0; i <= tot; i++)
            for (int j = 0; j <= end; j++)
                nxt[i][j] = 0;
        tot = 0;
    }
};
```

AC 自动机

```
#include <queue>
#include <string.h>
#include <string>
using std::string, std::queue;

struct AC_automaton
{
    static const int _N = 1e6;

    int (*trie)[27];
    int tot = 0;
    int *fail;
    int *e;
    AC_automaton()
    {
        trie = new int[_N][27];
        fail = new int[_N];
        e = new int[_N];
        memset(trie, 0, sizeof(trie));
        memset(fail, 0, sizeof(fail));
        memset(e, 0, sizeof(e));
    }

    void insert(string s)
    {
```

```cpp
        int now = 0;
        for (auto i : s)
            if (trie[now][i - 'a'])
                now = trie[now][i - 'a'];
            else
                trie[now][i - 'a'] = ++tot, now = tot;
        e[now]++;
    }

    void build()
    {
        queue<int> q;
        for (int i = 0; i < 26; i++)
            if (trie[0][i])
                q.push(trie[0][i]);
        while (q.size())
        {
            int u = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (trie[u][i])
                    fail[trie[u][i]] = trie[fail[u]][i], q.push(trie[u][i]);
                else
                    trie[u][i] = trie[fail[u]][i];
        }
    }

    int query(string t)
    {
        int u = 0, res = 0;
        for (auto c : t)
        {
            u = trie[u][c - 'a'];
            for (int j = u; j && e[j] != -1; j = fail[j])
                res += e[j], e[j] = -1;
        }
        return res;
    }
};

```

exKMP (Z Function)

对 KMP 中 next 数组的定义稍作改变，就会得到 exKMP。

在 exKMP 中，我们记录 z[i] 为：s[i:] 与 s 的最长公共前缀。

```cpp
/// @brief Z function, or exKMP.
/// @param s the string.
/// @return the Z function array.
vector<int> Z_function(string &s, VI &z)
{
    // indexing from 1
    int len = s.size();
    s = "#" + s;
    // vector<int> z(len + 1);
    z[1] = 0;
    int l = 1, r = 1;
    for (int i = 2; i <= len; i++)
    {
        if (i <= r && z[i - l + 1] < r - i + 1)
            z[i] = z[i - l + 1];
        else
        {
            z[i] = std::max(0, r - i + 1);
            while (i + z[i] <= len && s[z[i] + 1] == s[i + z[i]])
                z[i]++;
        }
        if (i + z[i] - 1 > r)
        {
            l = i;
```

```
                r = i + z[i] - 1;
            }
        }
        z[1] = len;
        return z;
}
```

2-数据结构

树状数组

```
class BIT
{
    int n = 2e6;
    long long *a;

  public:
    BIT(int size) : n(size)
    {
        a = new long long[size + 10];
    }
    void update(int p, long long x)
    {
        while (p <= n)
            a[p] += x, p += (p & (-p));
    }

    long long query(int l, int r)
    {
        long long ret = 0;
        l--;
        while (r > 0)
            ret += a[r], r -= (r & (-r));
        while (l > 0)
            ret -= a[l], l -= (l & (-l));
        return ret;
    }
};

```

```
template<typename T>
struct Fenwick{
    int n;
    vector<T> tr;

    Fenwick(int n) : n(n), tr(n + 1, 0){}

    int lowbit(int x){
        return x & -x;
    }

    void modify(int x, T c){//单点添加
        for(int i = x; i <= n; i += lowbit(i)) tr[i] += c;
    }

    void modify(int l, int r, T c){//区间添加
        modify(l, c);
        if (r + 1 <= n) modify(r + 1, -c);
    }

    T query(int x){
        T res = T();
        for(int i = x; i; i -= lowbit(i)) res += tr[i];
        return res;
    }

    T query(int l, int r){
        return query(r) - query(l - 1);
    }

    int find_first(T sum){//和出现的第一位置
        int ans = 0; T val = 0;
        for(int i = __lg(n); i >= 0; i--){
            if ((ans | (1 << i)) <= n && val + tr[ans | (1 << i)] < sum){
                ans |= 1 << i;
                val += tr[ans];
            }
        }
        return ans + 1;
    }

    int find_last(T sum){
        int ans = 0; T val = 0;
        for(int i = __lg(n); i >= 0; i--){
            if ((ans | (1 << i)) <= n && val + tr[ans | (1 << i)] <= sum){
                ans |= 1 << i;
                val += tr[ans];
            }
        }
        return ans;
    }

};
using BIT = Fenwick<int>;
```

并查集

```
#include <cassert>
#include <iostream>
#include <set>
/* Last modified: 23/08/01 */
class DSU
{
  private:
    int *f;
    int size;

  public:
    DSU(int size) : size(size)
    {
        assert(size > 1);
        f = new int[size + 10];
        for (int i = 1; i <= size; i++)
            f[i] = i;
    }
    int find(int x)
    {
        return f[x] == x ? x : (f[x] = find(f[x]));
    };
    bool same(int x, int y)
    {
        return find(x) == find(y);
    };
    bool merge(int x, int y)
    {
        int fx = find(x), fy = find(y);
        return ((fx != fy) ? f[fx] = fy : false);
    };
    int count()
    {
        std::set<int> s;
        for (int i = 1; i <= size; i++)
            s.insert(find(i));
        return s.size();
    }
};
```

ST 表

```cpp
// log2(x) 的预处理
// 1. 递推
lg[2] = 1;
for (int i = 3; i < N; i++)
    lg[i] = lg[i / 2] + 1;
// 2. 基于编译期计算
using std::array;
// WARNING: LOG_SIZE may cause CE if too big.
const int LOG_SIZE = 1e5 + 10;
constexpr array<int, LOG_SIZE> LOG = []() {
    array<int, LOG_SIZE> l{0, 0, 1};
    for (int i = 3; i < LOG_SIZE; i++)
        l[i] = l[i / 2] + 1;
    return l;
}();
// 3. 直接计算
int lg(int x)
{
    return 31 - __builtin_clz(x);
}
// STL 提供了 std::lg(), 底数是 e.
```

```cpp
class SparseTable
{
  private:
    // SIZE depends on range of f[i][0].
    // 22 is suitable for 1e5.
    static const int SIZE = 22;
    // f[i][j] maintains the result from i to i + 2 ^ j - 1;
    int (*f)[SIZE];
    using func = std::function<int(int, int)>;
    func op;
    // length of f from 1 to l;
    int l;

  public:
    SparseTable(int a[][SIZE], func foo, int len) : f(a), op(foo), l(len)
    {
        for (int j = 1; j < SIZE; j++)
            for (int i = 1; i + (1 << j) - 1 <= len; i++)
                // f[i][j] comes from f[i][j - 1].
                // f[i][j - 1], f[i + 2^(j - 1)] cover the range of f[i][j].
                f[i][j] = foo(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
    };
    int query(int x, int y)
    {
        int s = LOG[y - x + 1];
        return op(f[x][s], f[y - (1 << s) + 1][s]);
    }
};
```

带懒标记线段树

```cpp
const int N = 1e6;
int a[N];
int tag[4 * N];
int tree[4 * N];
int n;
void push_up(int p)
{
    tree[p] = tree[ls(p)] + tree[rs(p)];
}
void build(int p, int l, int r)
{
    if (l == r)
    {
        tree[p] = a[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(ls(p), l, mid);
    build(rs(p), mid + 1, r);
    push_up(p);
}
void push_down(int p, int l, int r)
{
    int mid = (l + r) >> 1;
    tag[ls(p)] += tag[p];
    tag[rs(p)] += tag[p];
    tree[ls(p)] += tag[p] * (mid - l + 1);
    tree[rs(p)] += tag[p] * (r - mid);
    tag[p] = 0;
}
void update(int nl, int nr, int k, int p = 1, int l = 1, int r = n)
{
    if (nl <= l && r <= nr)
    {
        tag[p] += k;
        tree[p] += k * (r - l + 1);
        return;
    }
    push_down(p, l, r);
    int mid = (l + r) >> 1;
    if (nl <= mid)
        update(nl, nr, k, ls(p), l, mid);
    if (nr > mid)
        update(nl, nr, k, rs(p), mid + 1, r);
    push_up(p);
}
int query(int x, int y, int l = 1, int r = n, int p = 1)
{
    int res = 0;
    if (x <= l && y >= r)
        return tree[p];
    int mid = (l + r) >> 1;
    push_down(p, l, r);
    if (x <= mid)
        res += query(x, y, l, mid, ls(p));
    if (y > mid)
        res += query(x, y, mid + 1, r, rs(p));
    return res;
}
int main()
{
    int q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    build(1, 1, n);
    while (q--)
    {
        int op, x, y, k;
        cin >> op;
        if (op == 1)
        {
            cin >> x >> y >> k;
            update(x, y, k);
        }
        else
        {
            cin >> x >> y;
            cout << query(x, y) << endl;
        }
    }
    return 0;
}
```

区间最值线段树

```
struct node
{
    int maxn;
} tree[800005];
int n;
int tag[800005];
void push_down(int p, int l, int r)
{ // 标记下压
    int mid = (r + l) / 2;
    tree[2 * p].maxn += tag[p];
    tree[2 * p + 1].maxn += tag[p];
    tag[2 * p] += tag[p];
    tag[2 * p + 1] += tag[p];
    tag[p] = 0;
}
void update(int l, int r, int k, int cl = 1, int cr = n, int p = 1)
{ // 更新
    if (cl > r || cr < l)
    {
        return;
    }
    if (cl >= l && cr <= r)
    {
        tree[p].maxn += k;
        if (cl < cr)
        {
            tag[p] += k;
        }
    }
    else
    {
        int mid = (cl + cr) >> 1;
        push_down(p, cl, cr);
        if (l <= mid)
            update(l, r, k, cl, mid, 2 * p);
        if (r > mid)
            update(l, r, k, mid + 1, cr, 2 * p + 1);
        tree[p].maxn = max(tree[p << 1].maxn, tree[p * 2 + 1].maxn);
    }
}
int query(int l, int r, int cl = 1, int cr = n, int p = 1)
{ // 查询
    if (cl >= l && cr <= r)
    {
        return tree[p].maxn;
    }
    else
    {
        int mid = (cl + cr) >> 1;
        push_down(p, cl, cr);
        int tmp = 0;
        if (l <= mid)
            tmp = max(tmp, query(l, r, cl, mid, 2 * p));
        if (r > mid)
            tmp = max(tmp, query(l, r, mid + 1, cr, 2 * p + 1));
        return tmp;
    }
}
```

单调队列

```
int p[N];
int head=1,tail=0;
for(int i=1;i<=n;i++){
    if(head<=tail&&p[head]==i-k){//当前 0          区间长度大于 k 时扔掉头部
        head++;
    }
    while(head<=tail&&a[p[tail]]<=a[i]) tail--;//此时求最大值
    p[++tail]=i;
    //则 head 记录区间内最值
}
```

3-动态规划

背包 DP

```
// 多重背包
for (int i = 1; i <= n; i++)
{
    int q = a[i].m;
    for (int j = 1; q; j *= 2)
    {
        if (j > q)
        {
            j = q;
        }
        q -= j;
        for (int k = w; k >= j * a[i].w; k--)
        {
            f[k] = max(f[k], f[k - j * a[i].w] + j * a[i].v);
        }
    }
}
```

二进制分组优化

```
index = 0;
for (int i = 1; i <= m; i++)
{
    int c = 1, p, h, k;
    cin >> p >> h >> k;
    while (k > c)
    {
        k -= c;
        list[++index].w = c * p;
        list[index].v = c * h;
        c *= 2;
    }
    list[++index].w = p * k;
    list[index].v = h * k;
}
```

状态压缩 DP

```
int cnt[1024];
int dp[40][1024][90];
int can[2000], num = 0;
int S = 1 << n;
for (int s = 0; s < S; s++)
{
    if ((s << 1) & s)
    {
        continue;
    }
    can[++num] = s;
    for (int j = 0; j < n; j++)
    {
        if ((s >> j) & 1)
        {
            cnt[num]++;
        }
    }
    dp[1][num][cnt[num]] = 1;
}
for (int i = 2; i <= n; i++)
```

```
{
    for (int j = 1; j <= num; j++)
    {
        int x = can[j];
        for (int p = 1; p <= num; p++)
        {
            int y = can[p];
            if ((y & x) || ((y << 1) & x) || ((y >> 1) & x))
                continue;
            for (int l = 0; l <= k; l++)
            {
                dp[i][j][cnt[j] + l] += dp[i - 1][p][l];
            }
        }
    }
}
```

枚举 s 的二进制真子集

```
for(int j=s;j;j=(j-1)&s){
    //...
}
```

数位 DP

现在问有多少的数比 12345 小

1. 关于前导 0：00999→999，09999→9999

2. 关于 limit 前面的数是否紧贴上限

如果前面的数是紧贴上限的，当前这位枚举的上限便是当前数的上限

如果前面的数不是紧贴上限的，当前这位枚举的上限便是 9

3. 关于 DP 维度

一般来说，DFS 有几个状态，DP 就几个维度

比如现在 DP 就是 DP [pos] [limt] [zero]

4. 关于记忆化 DP

现在枚举到了 10××× 和 11×××

显然这两种状态后面的 ××× 状态数是一样的

重点：dp[pos][limit][zero] 表示前面的数枚举状态确定，后面的数有多少种可能

5. 关于 DP 细节

一般来说我们一开始都 memset(dp,-1,sizeof(dp))

如果 dp[pos][limt][zero]!=-1 return dp[pos][limit][zero];

6. 关于初始化：

一开始 limit 是 1，表示一开始的数只能选 1~a[1]

一开始 zero 是 1，假定表示前面的数全为 0

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define mp make_pair
#define pb push_back  // vector 函数
#define popb pop_back // vector 函数
#define fi first
#define se second
const int N = 20;
// const int M=;
// const int inf=0x3f3f3f3f;     //一般为 int 赋最大值，不用于 memset 中
// const ll INF=0x3ffffffffffff; //一般为 ll 赋最大值，不用于 memset 中
```

```cpp
int T, n, len, a[N], dp[N][2][2];
inline int read()
{
    int x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
    {
        x = (x << 1) + (x << 3) + (ch ^ 48);
        ch = getchar();
    }
    return x * f;
}
int dfs(int pos, bool lim, bool zero)
{
    if (pos > len)
        return 1;
    if (dp[pos][lim][zero] != -1)
        return dp[pos][lim][zero];
    int res = 0, num = lim ? a[pos] : 9;
    for (int i = 0; i <= num; i++)
        res += dfs(pos + 1, lim && i == num, zero && i == 0);
    return dp[pos][lim][zero] = res;
}
int solve(int x)
{
    len = 0;
    memset(dp, -1, sizeof(dp));
    for (; x; x /= 10)
        a[++len] = x % 10;
    reverse(a + 1, a + len + 1);
    return dfs(1, 1, 1);
}
int main()
{
    int l = read(), r = read();
    printf("%d\n", solve(r) - solve(l - 1));
    return 0;
}
```

状压 DP 解哈密顿回路问题

```cpp

int dp[(1 << 21)][20];

for (int i = 0; i < (1 << (n + 1)); i++)
            for (int j = 1; j <= n; j++)
                if (dp[i][j])
                    for (int k = 1; k <= n; k++)
                        if (a[j][k] && (((i >> k) & 1) == 0))
                            dp[i | (1 << k)][k] = 1;

```

4-数学

快速幂

```cpp
int qpow(int a, int n)
{
    int ans = 1;
    while (n)
    {
        if (n & 1)
            ans = ans * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return ans;
}
```

exgcd

```
int exgcd(int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y), x0 = x, y0 = y;
    x = y0;
    y = x0 - (a / b) * y0;
    return d;
}
```

线性 inv

```
void getinv(int n)
{
    inv[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        inv[i] = mod - ((mod / i) * inv[mod % i]) % mod;
    }
}
```

分块

```
int ans = 0;
for (int l = 1, r; l <= n; l = r + 1)
{
    r = n / (n / l);
    ans += (r - l + 1) * (n / l);
}
cout << ans << endl;
```

欧拉筛

```
int Eular(int n)
{
    int cnt = 0;
    memset(is_prime, true, sizeof(is_prime));
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i <= n; i++)
    {
        if (is_prime[i])
        {
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
        {
            is_prime[i * prime[j]] = 0;
            if (i % prime[j] == 0)
                break;
        }
    }
    return cnt;
}
```

欧拉函数

```
int Eular(int n)
{
    int cnt = 0;
    memset(is_prime, true, sizeof(is_prime));
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i <= n; i++)
    {
        if (is_prime[i])
        {
            prime[++cnt] = i;
        }
        for (int j = 1; j <= cnt && i * prime[j] <= n; j++)
        {
            is_prime[i * prime[j]] = 0;
            if (i % prime[j] == 0)
                break;
        }
    }
    return cnt;
}
```

组合数

```
int fac[N];
int inv[N];
void init(int n)
{
    fac[0] = 1;
    inv[0] = 1;
    inv[1] = 1;
    fac[1] = 1;
    for (int i = 2; i <= 2 * n; i++)
    {
        fac[i] = fac[i - 1] * i % mod;
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    }
    for (int i = 1; i <= n; i++)
    {
        inv[i] = inv[i] * inv[i - 1] % mod;
    }
}
int C(int n, int m)
{
    if (m > n || m < 0 || n < 0)
        return 0;
    return fac[n] * inv[m] % mod * inv[n - m] % mod;
}
```

扩展欧拉定理

定义

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \gcd(a,m)=1, \\ a^b, & \gcd(a,m)\neq 1, b<\varphi(m), \\ a^{(b \bmod \varphi(m))+\varphi(m)}, & \gcd(a,m)\neq 1, b\geq\varphi(m). \end{cases} \pmod m$$

解释

读者可能对第二行产生疑问，这一行表达的意思是：如果 $b < \varphi(m)$ 的话，就不能降幂了。

主要是因为题目中 m 不会太大，而如果 $b < \varphi(m)$，自然复杂度是可以接受的。而如果 $b \geq \varphi(m)$ 的话，复杂度可能就超出预期了，这个时候我们才需要降幂来降低复杂度。

卡特兰数

```
int C(int n, int m)
{
    return fac[n] * qpow(fac[n - m], mod - 2) % mod * qpow(fac[m], mod - 2) % mod;
}
int cat(int n)
{
    return C(2 * n, n) * qpow(n + 1, mod - 2) % mod;
}
```

矩阵

```
class matrix
{
public:
    int x[105][105];
    int sz;
    matrix(int n)
    {
        sz = n;
        for (int i = 1; i <= sz; i++)
        {
            for (int j = 1; j <= sz; j++)
            {
                x[i][j] = 0;
            }
        }
    }
    matrix mul(matrix a, matrix b);
    matrix qpow(matrix a, int n);
    void tra(matrix a);
};

matrix matrix::mul(matrix a, matrix b)
{
    matrix c(a.sz);
    for (int i = 1; i <= a.sz; i++)
        for (int j = 1; j <= a.sz; j++)
            for (int k = 1; k <= a.sz; k++)
                c.x[i][j] = (c.x[i][j] % mod + (a.x[i][k] * b.x[k][j]) % mod) % mod;
    return c;
}
matrix matrix::qpow(matrix a, int n)
{
    matrix res(a.sz);
    for (int i = 1; i <= a.sz; i++)
        res.x[i][i] = 1;
    while (n > 0)
    {
        if (n & 1)
            res = mul(res, a);
        a = mul(a, a);
        n >>= 1;
    }
    return res;
}
void matrix::tra(matrix a)
{
    for (int i = 1; i <= a.sz; i++)
    {
        for (int j = 1; j <= a.sz; j++)
        {
            cout << a.x[i][j] << " ";
        }
        cout << endl;
    }
}
```

高斯消元

求解线性方程组

将各系数合为矩阵，再将其变为上三角矩阵

过程中通常要保证选择的主元绝对值最大以保证精度

n^3

```cpp
const int N = 100;
const double eps = 1e-10;
int n;
double a[N + 1][N + 1], b[N + 1];

void gauss()
{
    int l = 1;
    for (int i = 1; i <= n; i++)
    { // n 列
        for (int j = l; j <= n; j++)
        { // 找下面所有行中这一列处绝对值最大的
            if (abs(a[j][i]) > abs(a[l][i]))
            {
                for (int k = i; k <= n; k++)
                {
                    swap(a[l][k], a[j][k]);
                }
                swap(b[l], b[j]);
            }
        }
        if (abs(a[l][i]) < eps)
            continue;
        for (int j = 1; j <= n; j++)
        { // 对所有其他行，更新值
            if (j != l && abs(a[j][i]) > eps)
            {
                double delta = a[j][i] / a[l][i];
                for (int k = i; k <= n; k++)
                {
                    a[j][k] -= a[l][k] * delta;
                }
                b[j] -= b[l] * delta;
            }
        }
        ++l;
    }

    for (int i = l; i <= n; i++)
    { // 假如有剩下的行且 b 值不为 0 则无解
        if (abs(b[i]) > eps)
        {
            cout << " 无解" << endl;
            return;
        }
    }
    if (l <= n)
    {
        cout << " 无穷多解" << endl;
    }
    else
    {
        for (int i = 1; i <= n; i++)
        {
            cout << fixed << setprecision(10) << b[i] / a[i][i] << endl;
        }
    }
}
```

第二类斯特林数（Stirling Number）

第二类斯特林数（斯特林子集数）$\begin{Bmatrix} n \\ k \end{Bmatrix}$，也可记做 $S(n,k)$，表示将 n 个两两不同的元素，划分为 k 个互不区分的非空子集的方案数。

递推式

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

通项公式

$$\begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_{i=0}^{m} \frac{(-1)^{m-i} i^n}{i!(m-i)!}$$

5-图论

---

dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const ll maxn = 10+1e5;
const ll inf = 0x3f3f3f3f;
typedef pair<ll,ll> pll;
    vector<pll> mp[100100];
    ll n,m,s;
    ll dis[100100];
    ll vis[100100];

void dij(ll s){
    for(ll i=1;i<=n;i++){
        dis[i]=inf;
    }
    dis[s]=0;
    priority_queue<pll,vector<pll>,greater<pll> > q;
    q.push({dis[s],s});
    while(!q.empty()){
        ll u=q.top().second;
        q.pop();
        if(vis[u]) continue;
        vis[u]=1;
        for(auto [w,v]:mp[u]){
            if(dis[u]+w<dis[v]){
                dis[v]=dis[u]+w;
                q.push({dis[v],v});
            }
        }
    }
}
int main() {
    cin>>n>>m>>s;
    for(ll i=1;i<=m;i++){
        ll u,v,w;
        cin>>u>>v>>w;
        mp[u].push_back({w,v});
    }
    dij(s);
    for(ll i=1;i<=n;i++)
        cout<<dis[i]<<" ";
    return 0;
}
```

```cpp
ll fa[maxn];
void init(ll n){
    for(ll i=1;i<=n;i++){
        fa[i]=i;
    }
}
ll find(ll x){
    if(fa[x]==x)
        return x;
    fa[x]=find(fa[x]);
    return fa[x];
}
void merge(ll x,ll y){
    if(find(x)==find(y)) return ;
    fa[find(x)]=find(y);
}
```

最小生成树

prim O(n^2)

```
    vector<pll> edge[maxn];
    ll dis[maxn];
    ll vis[maxn];
void update(ll u){
    for(auto [v,w]:edge[u]){
        dis[v]=min(dis[v],w);
    }

}
ll prim(ll n){
    for(ll i=1;i<=n;i++){
        vis[i]=0;
        dis[i]=inf;
    }
    vis[1]=1;
    update(1);
    ll ans=0;
    for(ll i=1;i<=n-1;i++){//连接 n-1 条边
        ll pos,mi=inf;
        for(ll j=1;j<=n;j++){
            if(vis[j]) continue;
            if(dis[j]<mi){
                mi=dis[j];
                pos=j;
            }
        }
        ans+=mi;
        vis[pos]=1;
        update(pos);
    }
    return ans;
}
```

prim O(mlogm) (小根堆优化)

```
    vector<pll> edge[maxn];
    ll dis[maxn];
    ll vis[maxn];
ll prim(ll n){
    for(ll i=1;i<=n;i++){
        vis[i]=0;
        dis[i]=inf;
    }
    ll ans=0;
    priority_queue<pll,vector<pll>,greater<pll>> pq;
    pq.push({0,1});
    while(!pq.empty()){
        auto [d,u]=pq.top();
        pq.pop();
        if(vis[u]) continue;
        vis[u]=1;
        ans+=d;
        for(auto [v,w]:edge[u]){
            if(!vis[v]&&w<dis[v]){
                dis[v]=w;
                pq.push({dis[v],v});
            }
        }
    }
    return ans;
}
```

kruskal

```
typedef struct{
    ll u,v,w;
}eg;
    eg e[maxm];
    vector<pll> edge[maxn];
    ll fa[maxn];
bool cmp(eg a,eg b){
    return a.w<b.w;
}
ll find(ll a){
    return (fa[a]==a)?a:(fa[a]=find(fa[a]));
}
void merge(ll a,ll b){
    fa[a]=b;
}
ll kruscal(ll n,ll m){
    for(ll i=1;i<=n;i++){
        fa[i]=i;
    }
    sort(e+1,e+m+1,cmp);
    ll ans=0;
    for(ll i=1;i<=m;i++){
        ll a=find(e[i].u);
        ll b=find(e[i].v);
        if(a!=b){
            merge(a,b);
            ans+=e[i].w;
            n--;
        }
    }
    if(n==1){
        return ans;
    }else{
        return inf;
    }
}
for(ll i=1;i<=m;i++){
    ll u,v,w;
    cin>>u>>v>>w;
    edge[u].push_back({v,w});
    edge[v].push_back({u,w});
    e[i].u=u;e[i].v=v;e[i].w=w;
}
```

LCA

```
#include <bits/stdc++.h>
using namespace std;
#define IOS ios::sync_with_stdio(false),cin.tie(nullptr), cout.tie(nullptr);
#define ll long long
#define ull unint long long
#define lowbit(i) ((i) & (-i))
#define ls(p) (p << 1)
#define rs(p) (p << 1 | 1)
#define rep(i, a, b) for (ll i = a; i <= b; i++)
#define per(i, a, b) for (ll i = a; i >= b; i--)

```

```
typedef pair<ll, ll> pll;
const ll mod = 1e9 + 7;
const ll inf = 0x3f3f3f3f;
const ll maxn = 5e5 + 200;

    ll n, q, root;
    vector<ll> mp[N];
    ll lg2[maxn];
    ll dep[maxn];
    ll f[maxn][20];
    ll vis[maxn];
void dfs(ll u, ll fa = 0){
    if (vis[u]) return;
    vis[u] = 1;
    dep[u] = dep[fa] + 1;
    f[u][0] = fa;
    for (ll i = 1; i <= lg2[dep[u]]; i++){
        f[u][i] = f[f[u][i - 1]][i - 1];
    }
    for (auto v : mp[u]){
        dfs(v, u);
    }
}
ll lca(ll a, ll b){
    if (dep[a] > dep[b])
        swap(a, b);
    while (dep[a] != dep[b])
        b = f[b][lg2[dep[b] - dep[a]]];
    if (a == b)
        return a;
    for (ll k = lg2[dep[a]]; k >= 0; k--){
        if (f[a][k] != f[b][k]){
            a = f[a][k], b = f[b][k];
        }
    }
    return f[a][0];
}
int main(){
    IOS
    cin >> n >> q >> root;
    for (ll i = 1; i < n; i++){
        ll u, v;
        cin >> u >> v;
        mp[u].push_back(v);
        mp[v].push_back(u);
    }
    for (ll i = 2; i <= n; i++){
        lg2[i] = lg2[i / 2] + 1;
    }
    dfs(root);
    while (q--){
        ll u, v;
        cin >> u >> v;
        cout << lca(u, v) << endl;
    }
    return 0;
}
```