代 码 模 版

version 1.0

DopplerXD_yxc

24.04.22

CUGB

Doppler 的代码模版

一 基础操作

1 快读快写

```
// 关闭同步流（加速）
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);

更快但复杂的快读快写
int read() {
    int x = 0, w = 1;
    char ch = 0;
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            w = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9')
    {
        x = x * 10 + (ch - '0');
        ch = getchar();
    }
    return x * w;
}
void write(int x) {
    if (x < 0)
    {
        x = -x;
        putchar('-');
    }
    if (x > 9)
        write(x / 10);
    putchar(x % 10 + '0');
}
```

2 二分

```
int arr[N];
int binarySearch(int left, int right, int x) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) {
            return mid;
        }
        else if (arr[mid] < x) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
```

```
        }
    }
    return -1;
}
```

3 三分

P3382 模板【三分】https://www.luogu.com.cn/problem/P3382

```
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-8;
int n;
double a[15];
double l, r;
double f(double x) {
    double p = 1, res = 0;
    for (int i = n; i >= 0; i--) {
        res += a[i] * p;
        p *= x;
    }
    return res;
}
int main()
{
    cin >> n >> l >> r;
    for (int i = 0; i <= n; i++) cin >> a[i];
    double mid;
    while (r - l > eps) {
        mid = (l + r) / 2;
        if (f(mid - eps) < f(mid + eps)) l = mid;
        else r = mid;
    }
    cout << l << endl;
}
```

4 字符串与其他类型转换

```
整数转换为字符串
int num;
string s = to_string(num);

将一串 x 进制的字符串转换为 int 型数字
string s;
int a = stoi(s, 0, x);
long long b = stoll(s, 0, x);
stoull stold stod 同理
```

5 动态规划求最大子段和

```
int dp[N], a[N], ans = 0;
memset(dp, 0, sizeof(dp));
for(int i = 1; i <= n; i++) {
  if(dp[i - 1] > 0)
    dp[i] = dp[i - 1] + a[i];
  else
    dp[i] = a[i];
  ans = max(ans, dp[i]);
}
```

6 高精度加法

```
#include <bits/stdc++.h>
using namespace std;
vector<int> add(vector<int>& A, vector<int>& B) {
    if (A.size() < B.size()) return add(B, A);
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i++) {
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }
    if (t) C.push_back(t);
    return C;
}
int main()
{
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    for (int i = b.length() - 1; i >= 0; i--) B.push_back(b[i] - '0');
    auto C = add(A, B);
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    cout << endl;
}
```

7 高精度减法

```
#include <bits/stdc++.h>
using namespace std;
bool cmp(vector<int>& A, vector<int>& B) {
    if (A.size() != B.size()) return A.size() > B.size();
    for (int i = A.size() - 1; i >= 0; i--) {
        if (A[i] != B[i])
            return A[i] > B[i];
    }
    return true;
}
```

```
vector<int> sub(vector<int>& A, vector<int>& B) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i++) {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main()
{
    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    for (int i = b.length() - 1; i >= 0; i--) B.push_back(b[i] - '0');
    vector<int> C;
    if (cmp(A, B)) {
        C = sub(A, B);
    }
    else {
        C = sub(B, A);
        cout << "-";
    }
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    cout << endl;
}
```

8 高精度乘法

```
// 高精a * 低精b
#include <bits/stdc++.h>
using namespace std;
vector<int> mul(vector<int>& A, int b) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size() || t; i++) {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main()
{
    string a;
    int b;
```

```
    vector<int> A;
    cin >> a >> b;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    auto C = mul(A, b);
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    cout << endl;
}
```

9 高精度除法

```
#include <bits/stdc++.h>
using namespace std;
vector<int> div(vector<int>& A, int b, int& r) {
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i >= 0; i--) {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main()
{
    string a;
    int b;
    vector<int> A;
    cin >> a >> b;
    for (int i = a.size(); i >= 0; i--) A.push_back(a[i] - '0');
    int r;
    auto C = div(A, b, r);
    for (int i = C.size() - 1; i >= 0; i--) cout << C[i];
    cout << endl << r << endl; // r为余数
}
```

10 bitset

```
构造方法
bitset<4> bitset1;    //无参构造，长度为 4，默认每一位为 0
bitset<8> bitset2(12);  //长度为 8，二进制保存，前面用 0 补充

string s = "100101";
bitset<10> bitset3(s);    //长度为10，前面用 0 补充

char s2[] = "10101";
bitset<13> bitset4(s2);    //长度为13，前面用 0 补充

cout << bitset1 << endl;    //0000
cout << bitset2 << endl;    //00001100
```

```
cout << bitset3 << endl;    //0000100101
cout << bitset4 << endl;    //0000000010101

数据访问
cout << foo[0] << endl;//1,下标从 0 开始，表示二进制的最低位

数据操作
foo[i] = 0;//给第 i 位赋值
foo.flip();//反转全部
foo.flip(i);//反转下标为 i 的位
foo.set();//全部置1
foo.set(i);//第 i 位置 1
foo.set(i,0);//第 i 位置 0
foo.reset();//全部置 0
foo.reset(i);//第 i 位置 0

数据转换
bitset<8> foo ("10011011");
string s = foo.to_string();    //将bitset转换成string类型
unsigned long a = foo.to_ulong();    //将bitset转换成unsigned long类型
unsigned long long b = foo.to_ullong();    //将bitset转换成unsigned long long类型

常用函数
bitset<8> foo ("10011011");
cout << foo.count() << endl;    //5    （count函数用来求bitset中1的位数，foo中共有５个１
cout << foo.size() << endl;    //8    （size函数用来求bitset的大小，一共有８位
cout << foo.test(0) << endl;    //true    （test函数用来查下标处的元素是０还是１，并返回false或true，
此处foo[0]为１，返回true
cout << foo.test(2) << endl;    //false    （同理，foo[2]为０，返回false
cout << foo.any() << endl;    //true    （any函数检查bitset中是否有１
cout << foo.none() << endl;    //false    （none函数检查bitset中是否没有１
cout << foo.all() << endl;    //false    （all函数检查bitset中是全部为１

位操作符
bitset<4> foo (string("1001"));
bitset<4> bar (string("0011"));
cout << (foo^=bar) << endl;       // 1010 (foo对bar按位异或后赋值给foo)
cout << (foo&=bar) << endl;       // 0010 (按位与后赋值给foo)
cout << (foo|=bar) << endl;       // 0011 (按位或后赋值给foo)
cout << (foo<<=2) << endl;        // 1100 (左移２位，低位补０，有自身赋值)
cout << (foo>>=1) << endl;        // 0110 (右移１位，高位补０，有自身赋值)
cout << (~bar) << endl;           // 1100 (按位取反)
cout << (bar<<1) << endl;         // 0110 (左移，不赋值)
cout << (bar>>1) << endl;         // 0001 (右移，不赋值)
cout << (foo==bar) << endl;       // false (0110==0011为false)
cout << (foo!=bar) << endl;       // true  (0110!=0011为true)
cout << (foo&bar) << endl;        // 0010 (按位与，不赋值)
cout << (foo|bar) << endl;        // 0111 (按位或，不赋值)
cout << (foo^bar) << endl;        // 0101 (按位异或，不赋值)
```

11 结构体重载比较运算符

```
struct node {
  int dis, u;
  bool operator>(const node& a) const { return dis > a.dis; }
};
```

12 对顶堆

定义两个堆，大根堆维护较小值，小根堆维护较大值。将大于大根堆顶的值放到小根堆，将小于等于大根堆顶值的数放入大根堆，然后维护两个堆的 size 的差值不大于1，元素较多的堆的堆顶值即为当前中位数。

```
#include <bits/stdc++.h>
using namespace std;
priority_queue<int, vector<int>, greater<int> > g;
priority_queue<int, vector<int>, less<int> > l;
int main()
{
  ios::sync_with_stdio(false), cin.tie(0);
  int n, x;
  cin >> n >> x;
  int mid = x;
  cout << mid << '\n';
  for(int i = 2; i <= n; i++) {
    int x;
    cin >> x;
    if(x > mid) g.push(x);
    else l.push(x);
    if(i % 2) {
      while(g.size() > l.size()) {
        l.push(mid);
        mid = g.top();
        g.pop();
      }
      while(l.size() > g.size()) {
        g.push(mid);
        mid = l.top();
        l.pop();
      }
      cout << mid << '\n';
    }
  }
  return 0;
}
```

13 vector 去重

```
vector<int> a;
int n;
cin >> n;
for(int i = 1; i <= n; i++) {
  int x;
  cin >> x;
  a.push_back(x);
}
sort(a.begin(), a.end());
a.erase(unique(a.begin(), a.end()), a.end());
```

14 iota生成连续数序列

```
// C++ 11 引入
vector<int> v;
iota(v.begin(), v.end(), 1);
// 生成1-n
```

15 sort 中使用 lambda 编写排序规则

```
vector<int> q;
iota(q.begin(), q.end(), 1);
vector<int> v;
// 按照v从小到大的顺序对p进行排序
sort(q.begin(), q.end(),
        [&](int i, int j) {
            return v[i] < v[j] || (v[i] == v[j] && i < j);
        });
```

16 滑动窗口

```
const int N = 1e5 + 5;
int n, k; // k for length of windows
int mx[N], a[N]; // mx下标从k到n
deque<int> q;
void solve() {
  cin >> n >> k;
  for(int i = 1; i <= n; i++) cin >> a[i];
  for(int i = 1; i <= n; i++) {
    while (!q.empty() && a[q.back()] < a[i]) q.pop_back();
        q.push_back(i);
        if (i >= k) {
            while (q.size() && q.front() <= i - k) q.pop_front();
            mx[i] = a[q.front()];
        }
  }
}
```

17 双向广搜

双向搜索，搜索到重复状态就停止，交替提取从start和end演变来的状态，进行搜索

能够将搜索次数从 $ a^b $ 优化到 $ 2a^{b/2} $

例题：P1032 [NOIP2002 提高组] 字串变换

已知有两个字串 A,B 及一组字串变换的规则（至多 6 个规则），规则的含义为：在 *A* 中的子串 *A*1 可以变换为 *B*1，*A*2 可以变换为 *B*2…

若在 10 步（包含 10 步）以内能将 *A* 变换为 *B*，则输出最少的变换步数；否则输出 `NO ANSWER!`

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
string s, t, a[10], b[10];
int cnt, ans = 11;
map<string, int> ma, mb;
int main()
{
    cin >> s >> t;
    while (cin >> a[cnt] >> b[cnt]) cnt++;
    queue<string> qa, qb;
    qa.push(s);
    qb.push(t);
    ma[s] = mb[t] = 0;
    while (qa.size() && qb.size()) {
        if (qa.size() < qb.size()) {
            string tmp = qa.front();
            qa.pop();
            int len = tmp.length();
            bool f = 0;
            for (int i = 0; i < len; i++) {
                for (int j = 0; j < cnt; j++) {
                    int l = a[j].length();
                    if (tmp.substr(i, l) != a[j]) continue;
                    string str = tmp.substr(0, i) + b[j] + tmp.substr(i + l);
                    if (mb.find(str) != mb.end()) {
                        ans = min(ans, ma[tmp] + 1 + mb[str]);
                        f = 1;
                        break;
                    }
                    if (ma.find(str) != ma.end()) continue;
                    ma[str] = ma[tmp] + 1;
                    qa.push(str);
                }
                if (f) break;
            }
        }
        else {
            string tmp = qb.front();
            qb.pop();
            int len = tmp.length();
            bool f = 0;
            for (int i = 0; i < len; i++) {
```

```
                for (int j = 0; j < cnt; j++) {
                    int l = b[j].length();
                    if (tmp.substr(i, l) != b[j]) continue;
                    string str = tmp.substr(0, i) + a[j] + tmp.substr(i + l);
                    if (ma.find(str) != ma.end()) {
                        ans = min(ans, mb[tmp] + 1 + ma[str]);
                        f = 1;
                        break;
                    }
                    if (mb.find(str) != mb.end()) continue;
                    mb[str] = mb[tmp] + 1;
                    qb.push(str);
                }
                if (f) break;
            }
        }
        if (ans <= 10) break;
    }
    if (ans == 11) cout << "NO ANSWER!";
    else cout << ans;
}
```

18 单调栈

```
//单调栈：输出左边第一个比它小的数，如果没有输出-1
int s[N];
int top;
int main(){
int n,i,j,a;
cin>>n;
    for(i = 1;i<=n;i++){
        cin>>a;
        while(top > 0 && s[top] >= a)
            top--;
        if(top == 0)    cout<<"-1 ";
        else    cout<<s[top]<<" ";
        s[++top] = a;
    }
    return 0;
}
```

19 全排列函数

```cpp
int n;
cin >> n;
vector<int> a(n);
// iota(a.begin(), a.end(), 1);
for (auto &it : a) cin >> it;
sort(a.begin(), a.end());

do {
    for (auto it : a) cout << it << " ";
    cout << endl;
} while (next_permutation(a.begin(), a.end()));
```

20 判断非递减 is_sorted

```cpp
//a数组[start,end)区间是否是非递减的，返回bool型
cout << is_sorted(a + start, a + end);
```

21 cout 输出流控制

```
stew(x) 补全 x 位输出，默认用空格补全
setfill(char) 设定补全类型
cout << setw(12) << setfill('*') << 12 << endl;
```

22 约瑟夫问题

约瑟夫问题

n 个人编号 $0,1,2...,n-1$ ，每次数到 k 出局，求最后剩下的人的编号。

$O(N)$ 。

```cpp
int jos(int n,int k){
    int res=0;
    repeat(i,1,n+1)res=(res+k)%i;
    return res; // res+1，如果编号从1开始
}
```

$ O(K\log N) $ ，适用于 K 较小的情况。

```cpp
int jos(int n,int k){
    if(n==1 || k==1)return n-1;
    if(k>n)return (jos(n-1,k)+k)%n; // 线性算法
    int res=jos(n-n/k,k)-n%k;
    if(res<0)res+=n; // mod n
    else res+=res/(k-1); // 还原位置
    return res; // res+1，如果编号从1开始
}
```

23 日期换算（基姆拉尔森公式）

已知年月日，求星期数。

```
int week(int y,int m,int d){
    if(m<=2)m+=12,y--;
    return (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7+1;
}
```

二 数论

1 乘法取模

```
ll mul(ll a, ll b, ll m)
{
    a = a % m, b = b % m;
    ll res = 0;
    while (b > 0) {
        if (b & 1) res = (res + a) % m;
        a = (a + a) % m;
        b >>= 1;
    }
    return res;
}
```

2 快速幂取模 fastPow(a, n, m) = (a^n)%m

```
ll fastPow(ll a, ll n, ll mod) // (a^n)%m
{
    ll ans = 1;
    a %= mod;
    while (n) {
        if (n & 1)ans = (ans * a) % mod;
        a = (a * a) % mod;
        n >>= 1;
    }
    return ans;
}
```

3 矩阵乘法&快速幂

```
int mod;
struct matrix { int m[N][N]; };
matrix operator * (const matrix& a, const matrix& b) {
    matrix c;
    memset(c.m, 0, sizeof(c.m));
    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            for (int k = 0; k < N; k++)
                c.m[i][j] = (c.m[i][j] + a.m[i][k] * b.m[k][j]) % mod;
    return c;
}
```

```
matrix pow_matrix(matrix a, int n)
{
    matrix ans;
    memset(ans.m, 0, sizeof(ans.m));
    for (int i = 0; i < N; i++)
        ans.m[i][i] = 1;
    while (n) {
        if (n & 1) ans = ans * a;
        a = a * a;
        n >>= 1;
    }
    return ans;
}
```

4 GCD & LCM

裴蜀定理 如果 a 和 b 均为整数，则有整数 x 和 y 使 ax+by=gcd(a,b)；ab 互素当且仅当存在 x 和 y 使得 ax+by=1

```
int gcd(int a, int b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
int lcm(int a, int b) {
    return a / gcd(a, b) * b;
}
```

5 扩展欧几里得

ax+by=c 有解的充要条件是 d=gcd(a,b) 能整除 c
扩展欧几里得算法求解二元丢番图方程，进而求解 ax+by=c 的特解

1. 判断方程是否有整数解，即 d 能整除 c
2. exgcd 求 ax+by=d 的特解 xx 和 yy
3. axx+byy=d 两边同乘 c/d，对照 ax+by=c，得特解 xx'=xx*c/d*，*yy'*=*yy*c/d
4. 方程 ax+by=c 的通解为 x = xx'+(b/d)n, y = yy'-(a/d)n

```
ll extend_gcd(ll a, ll b, ll& x, ll& y) // 返回 d=gcd(a,b), x 和 y 是 ax+by=d 的一组特解
{
    if (b == 0) { x = 1; y = 0; return a; }
    ll d = extend_gcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

6 求解同余方程（求逆）

```
ll mod_inverse(ll a, ll m) // ax===1(mod m)
{
    ll x, y;
    extend_gcd(a, m, x, y);
    return (x % m + m) % m;
}
```

7 费马小定理

设 n 为素数，a 是正整数且与 n 互素，则有 a^(n-1)===1(mod n)

```
ll mod_inverse(ll a, ll mod)
{
    return fastPow(a, mod - 2, mod);
}
```

8 Miller-Rabin 素性测试

```
ll fastPow();
bool witness(ll a, ll n)
{
    ll u = n - 1;
    int t = 0;
    while (u & 1 == 0) u >>= 1, t++;
    ll x1, x2;
    x1 = fastPow(a, u, n);
    for (int i = 1; i <= t; i++) {
        x2 = fastPow(x1, 2, n);
        if (x2 == 1 && x1 != 1 && x1 != n - 1) return true;
        x1 = x2;
    }
    if (x1 != 1) return true;
    return false;
}
int miller_rabin(ll n, int s)
{
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;
    for (int i = 0; i < s && i < n; i++) {
        ll a = rand() % (n - 1) + 1;
        if (witness(a, n)) return 0;
    }
    return 1;
}
void solve() // main
{
    int m;
    while (scanf("%d", &m) != EOF) {
        int cnt = 0;
        for (int i = 0; i < m; i++) {
```

```
            ll n;
            scanf("%lld", &n);
            int s = 50;
            cnt += miller_rabin(n, s);
        }
        printf("%d\n", cnt);
    }
}
```

9 欧拉筛

```
int prime[N];
bool vis[N];
int euler_sieve(int n)
{
    int cnt = 0;
    memset(prime, 0, sizeof(prime));
    memset(vis, 0, sizeof(vis));
    for (int i = 2; i <= n; i++) {
        if (!vis[i]) prime[cnt++] = i;
        for (int j = 0; j < cnt; j++) {
            if (i * prime[j] > n) break;
            vis[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    return cnt;
}
```

10 欧拉函数

威尔逊定理：若 p 为素数，则 p 可以整除 (p-1)!+1

```
int euler(int n) // 求欧拉函数值
{
    int ans = n;
    for (int p = 2; p * p <= n; ++p) {
        if (n % p == 0) {
            ans = ans / p * (p - 1);
            while (n % p == p)
                n /= p;
        }
    }
    if (n != 1) ans = ans / n * (n - 1);
    return ans;
}
```

11 二项式定理&卢卡斯定理

二项式定理：C(n,r)%m=(n! mod m)((r!)^(-1) mod m)((n-r!)^(-1) mod m) mod m
卢卡斯定理：要求 m 为素数

```
ll fac[N];
ll fastPow(ll a, ll n, ll m)
{
    ll ans = 1;
    a %= m;
    while (n) {
        if (n & 1) ans = (ans * a) % m;
        a = (a * a) % m;
        n >>= 1;
    }
    return ans;
}
ll inverse(ll a, int m) { return fastPow(fac[a], m - 2, m); }
ll C(ll n, ll r, int m)
{
    if (r > n) return 0;
    return ((fac[n] * inverse(r, m)) % m * inverse(n - r, m) % m);
}
ll lucas(ll n, ll r, int m)
{
    if (r == 0) return 1;
    return C(n % m, r % m, m) * lucas(n / m, r / m, m) % m;
}
```

12 Bash Game

```
void Bash_game()
{
    int n, m; // 共 n 个，每次最多拿 m 个
    cin >> n >> m;
    if (n % (m + 1) == 0) cout << "second\n";
    else cout << "first\n";
}
```

13 素数筛 O(n)

```
int cnt = 0, prime[N], v[N];
for(int i = 2; i <= n; i++) {
    if(v[i] == 0) {
        v[i] = i;
        prime[++cnt] = i;
    }
    for(int j = 1; j <= cnt; j++) {
        if(prime[j] > v[i] || i * prime[j] > n) break; // 保证每一个合数都是被他的最小质因子筛掉
        v[i * prime[j]] = prime[j]; // v[i]表示i的最小质因子
    }
}
```

14 分解质因数

```
void divide(int n){
  for(int i = 2;i<=n/i;i++){
    if(n%i==0){
      int t = 0;
      while(n % i == 0){
        n/=i;
        t++;
      }
      //输出分解出的因数和个数
      printf("%d %d\n",i,t);
    }
  }
  //如果n大于1，则n为质数
  if(n>1) printf("%d 1\n",n);
}
```

预处理质数后用prime数组分解

```
void div(int n){
    int i,j;
    for(i = 1;i<=n/prime[i];i++){
        int t = prime[i];
        if(n%t == 0){
            int num = 0;
            while(n%t == 0){
                n/=t;
                num++;
            }
            res[fcnt++] ={t,num};
        }
    }
    if(n>1)
        res[fcnt++] = {n,1};
}
```

15 约数

算数基本定理

$$N = {P_1}^{a_1}{P_2}^{a_2}{P_3}^{a_3}\dots{P_n}^{a_n} \tag{1}$$

其中 P 为质数, a 为正整数

试除法求约数

```
void divisors(int n){
  vector<int> v;
  for(int i = 1;i<=n/i;i++){
    if(n%i == 0){
      v.push_back(i);
      if(i*i != n)
        v.push_back(n/i);
```

```
    }
  }
  //按照从小到大的顺序输出它的所有约数
  sort(v.begin(),v.end());
  for(int i = 0;i<v.size();i++)
    printf("%d ",v[i]);
}
```

求约数的个数

$$N = (a_1 + 1) * (a_2 + 1) * (a_3 + 1) * \cdots * (a_n + 1) \tag{2}$$

求所有约数的和

$$N = ({p_1}^0 + {p_1}^1 + {p_1}^2 \ldots + {p_1}^{a_1}) * ({p_2}^0 + {p_2}^1 + \ldots + p2a2) * \ldots * ({p_n}^0 + {p_n}^1 + \cdots + {p_n}^{a_n}) \tag{3}$$

三 图论

1 Floyd

适用**最短路存在**的任何图，f[i][j] 为所求最短路长度

```
void floyd() {
    int f[N][N];
    for (int k = 1; k <= n; k++) {
        for (int x = 1; x <= n; x++) {
            for (int y = 1; y <= n; y++) {
                f[x][y] = min(f[x][y], f[x][k] + f[k][y]);
            }
        }
    }
}
```

2 Bellman_Ford

可以求出有负权的图的最短路，或对最短路不存在的情况进行判断

```
struct edge {
    int v, w;
};
vector<edge> e[N];
int dis[N], cnt[N], vis[N];
queue<int> q;
bool spfa(int n, int s) {
    memset(dis, 63, sizeof(dis));
    dis[s] = 0, vis[s] = 1;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop(), vis[u] = 0;
        for (auto ed : e[u]) {
            int v = ed.v, w = ed.w;
```

```
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u] + 1;  // 记录最短路经过的边数
                if (cnt[v] >= n) return false;
                // 在不经过负环的情况下，最短路至多经过 n - 1 条边
                // 因此如果经过了多于 n 条边，一定说明经过了负环
                if (!vis[v]) q.push(v), vis[v] = 1;
            }
        }
    }
    return true;
}
```

更新部分的另一种写法

```
if(dis[v] > dis[u] + w) {
  dis[v] = dis[u] + w;
  if(!vis[v]) {
    vis[v] = 1;
    q.push(v);
    cnt[v]++;
    if(cnt[v] >= n) return false;
  }
}
```

差分约束

形如 `a - b >= c` 的约束条件，在图中添加有向边 `ed[a].push_back({b, -c});`
对于 `a - b <= c`，添加 `ed[b].push_back({a, c});`
对于 `a == b`，添加 a b 之间权值为 0 的边即可。

从 0 向其他所有点增加一条权值为 0 的边，注意此时建立了一个虚拟源点，所以判断要改成 `cnt[v] > n`

建图完成后，以 0 为起点跑一边 SPFA，若存在负环，则无解。否则，xi = dis[i] 为该差分约束系统的一组解。

3 Dijkstra

求解非赋权图单源最短路

```
struct edge {
    int v, w;
};
struct node {
    int dis, u;
    bool operator>(const node& a) const { return dis > a.dis; }
};
vector<edge> e[N];
int dis[N], vis[N];
priority_queue<node, vector<node>, greater<node> > q;
void dijkstra(int n, int s)
{
```

```
    memset(dis, 63, sizeof(dis));
    dis[s] = 0;
    q.push((node) { 0, s });
    while (!q.empty()) {
        int u = q.top().u;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto ed : e[u]) {
            int v = ed.v, w = ed.w;
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                q.push((node) { dis[v], v });
            }
        }
    }
}
```

4 Kruskal

```
const int N = 2e3 + 5;
struct edge {
    int u, v, w;
    bool operator> (const edge& a) const { return w > a.w; }
};
priority_queue<edge, vector<edge>, greater<edge> > q; // 边集按权值排序
vector<edge> mp;
int fa[N];
int n, m, ans = 0;
int find(int x) { // 并查集判环
    int root = fa[x];
    while (root != fa[root]) root = fa[root];
    while (x != root) {
        int t = fa[x];
        fa[x] = root;
        x = t;
    }
    return root;
}
void Kruskal() {
    for (int i = 1; i <= n; i++) fa[i] = i;
    while (mp.size() < n - 1 && !q.empty()) { // 克鲁斯卡尔
        edge ed = q.top();
        q.pop();
        int u = find(ed.u), v = find(ed.v);
        if (u != v) {
            mp.push_back(ed);
            ans += ed.w;
            fa[u] = fa[v];
        }
    }
```

```
}
int main()
{
    cin >> n >> m;
    while (m--) {
        int u, v, w;
        cin >> u >> v >> w;
        q.push((edge) { u, v, w });
    }
    Kruskal();
    if (mp.size() < n - 1 && q.empty()) {
        cout << "无法构成最小生成树";
        return 0;
    }
    cout << "总权值: " << ans << endl;
    cout << "构成最小生成树的边是: " << endl;
    for (auto e : mp)
        printf("%d %d %d\n", e.u, e.v, e.w);
    return 0;
}
```

5 prim

```
const int N = 5e3 + 5;
struct edge {
    int v, w;
    bool operator> (const edge& e) const { return w > e.w; }
};
int n, m;
ll ans = 0;
bool vis[N];
vector<edge> ed[N];
void prim(int s) {
    int cnt = 1;
    vis[s] = 1;
    priority_queue<edge, vector<edge>, greater<edge> > q;
    for (auto e : ed[s])
        q.push(e);
    while (!q.empty() && cnt < n) {
        edge e = q.top();
        q.pop();
        if (vis[e.v]) continue;
        for (auto e : ed[e.v])
            q.push(e);
        ans += e.w;
        vis[e.v] = 1;
    }
}
int main()
{
    cin >> n >> m;
    while (m--) {
```

```
        int u, v, w;
        cin >> u >> v >> w;
        ed[u].push_back((edge) { v, w });
        ed[v].push_back((edge) { u, w });
    }
    prim(1);
    for (int i = 1; i <= n; i++)
        if (!vis[i]) {
            cout << "impossible";
            return 0;
        }
    cout << ans << endl;
    return 0;
}
```

6 拓扑排序（常用BFS判环）

复杂度 O(n + m)

```
vector<int> ed[N]; // 存图
int in[N]; // 保存入度
bool vis[N];
int main()
{
  int n, m;
    cin >> n >> m;
    while(m--) {
    int u, v;
        cin >> u >> v;
        ed[u].push_back(v);
        in[v]++;
    }
    queue<int> q;
    vector<int> L;
    for (int i = 1; i <= n; i++)
        if (!in[i])
            q.push(i);
    while (!q.empty()) {
        int u = q.front();
        L.push_back(u);
        q.pop();
        for (int v : ed[u]) {
            if (in[v] == 0) continue;
            in[v]--;
            if (!in[v])
                q.push(v);
        }
    }
    if(L.size() != n) cout << "有环\n"; // 输出L的内容即为拓扑排序的结果
}
```

```
void topo() {
```

```
  queue<int> q;
  for(int i = 1; i <= n; i++) {
    if(!in[i])
      q.push(i);
  }
  while(!q.empty()) {
    int u = q.front();
    q.pop();
    for(int v : ed[u]) {
      in[v]--;
      if(!in[v]) q.push(v);
    }
  }
}
```

7 同余最短路

处理「给定 n 个整数，求这 n 个整数能拼凑出多少的其他整数（n 个整数可以重复取）」，以及「给定 n 个整数，求这 n 个整数不能拼凑出的最小（最大）的整数」，或者「至少要拼几次才能拼出模 K 余 p 的数」的问题。

状态转移：f(i + y) = f(i) + y

```
// 例题：luogu跳楼机：给定 h，从 1 开始，可以加 x、y、z，问有多少组 abc，使得 1 < ax+by+cz <= h
#include <bits/stdc++.h>
#define rep(i, j, k) for(int i = (j); i <= (k); i++)
#define per(i, k, j) for(int i = (k); i >= (j); i--)
#define ll long long
using namespace std;
const int N = 1e5 + 5;
const ll llINF = 0x3f3f3f3f3f3f3f3f;
struct edge {
    ll v, w;
};
vector<edge> ed[N];
ll h, dis[N];
ll x, y, z;
bool vis[N];
void spfa() {
    dis[1] = 1;
    vis[1] = 1;
    queue<ll> q;
    q.push(1);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (auto e : ed[u]) {
            int v = e.v, w = e.w;
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                if (!vis[v]) {
                    q.push(v);
                    vis[v] = 1;
```

```
                }
            }
        }
    }
}
void solve()
{
    memset(dis, llINF, sizeof(dis));
    cin >> h >> x >> y >> z;
    if (x == 1 || y == 1 || z == 1) {
        cout << h << '\n';
        return;
    }
    rep(i, 0, x - 1) {
        ed[i].push_back({(i + y) % x, y}); // 加边
        ed[i].push_back({(i + z) % x, z});
    }
    spfa(); // 跑一边单源最短路
    ll ans = 0;
    rep(i, 0, x - 1) {
        if (h >= dis[i]) {
            ans += (h - dis[i]) / x + 1;
        }
    }
    cout << ans << '\n';
}
int main()
{
    int T_T = 1;
    // cin >> T_T;
    while (T_T--)
        solve();
    return 0;
}
```

8 传递闭包

floyd 判断连通性　模版: https://www.luogu.com.cn/problem/B3611

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
bool dis[105][105];
int n;
void floyd() {
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            if (dis[i][k])
                for (int j = 1; j <= n; j++)
                    if (dis[k][j])
                        dis[i][j] = 1;
}
int main()
```

```
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> dis[i][j];
    floyd();
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++)
            cout << dis[i][j] << " ";
        cout << endl;
    }
}
```

9 最小环问题

给出一个图，问其中的由 n 个节点构成的边权和最小的环 (n ≥ 3) 是多大。 https://www.luogu.com.cn/problem/P6175

记原图中 u,v 之间边的边权为 val[u] [v]。
我们注意到 Floyd 算法有一个性质：在最外层循环到点 k 时（尚未开始第 k 次循环），最短路数组 dis 中，dis[u][v] 表示的是从 u 到 v 且仅经过编号在 [1 , k) 区间中的点的最短路。
由最小环的定义可知其至少有三个顶点，设其中编号最大的顶点为 w，环上与 w 相邻两侧的两个点为 u,v，则在最外层循环枚举到 k = w 时，该环的长度即为 dis[u][v] + val[v][w] + val[w][u].
故在循环时对于每个 k 枚举满足 i < k, j < k 的 (i,j)，更新答案即可。

```
#include <bits/stdc++.h>
using namespace std;
const int INF = 0x2a2a2a2a; // 0x3f 会过大导致溢出WA
int dis[105][105], val[105][105];
int n, m;
int floyd() {
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= n; ++j) dis[i][j] = val[i][j];  //初始化最短路矩阵
    }
    int ans = INF;
    for (int k = 1; k <= n; ++k) {
        for (int i = 1; i < k; ++i) {
            for (int j = 1; j < i; ++j)
                ans = min(ans, dis[i][j] + val[i][k] + val[k][j]);  //更新答案
        }
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j)
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);  //floyd 更新最短路矩阵
        }
    }
    return ans;
}
int main()
{
    memset(val, INF, sizeof(val));
    cin >> n >> m;
    while (m--) {
```

```
        int u, v, w;
        cin >> u >> v >> w;
        val[u][v] = val[v][u] = w;
    }
    int ans = floyd();
    if (ans == INF) cout << "No solution.";
    else cout << ans;
}
```

10 Johnson 求全源最短路

P5905 【模板】全源最短路（Johnson） https://www.luogu.com.cn/problem/P5905
给定一个有向图，可能存在自环和重边，也不一定是连通图，求所有点对之间的最短路径

思路：

- 新建虚拟节点，向所有节点连接一条权值为 0 的边，跑一边spfa，得到 0 号结点到其他 n 个点的最短路，记为 h_i。
- 假设存在一条从 u 到 v 的权值为 w 的边，则将该边的权值从新设置为 $w + h_u - h_v$，目的是确保所有边的权值都非负，进而使用 dijkstra。
- 以每个点为起点跑 dijkstra 求最短路即可。最后得到的任意两点间最短路要减去 $h_u - h_v$

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 3e3 + 5;
struct edge {
    int v;
    ll w;
};
struct node {
    ll dis;
    int u;
    bool operator>(const node& a) const { return dis > a.dis; }
};
vector<edge> ed[N];
int n, m;
ll h[N], dis[N][N];
int cnt[N];
bool vis[N];
void spfa(int s) {
    memset(h, 63, sizeof(h));
    queue<int> q;
    q.push(s);
    h[s] = 0;
    while (!q.empty()) {
        int u = q.front();
        vis[u] = 0;
        q.pop();
        for (auto e : ed[u]) {
            int v = e.v;
```

```
            ll w = e.w;
            if (h[v] > h[u] + w) {
                h[v] = h[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] > n) {
                    cout << -1;
                    exit(0);
                }
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
    }
}
void dijkstra(int s) {
    priority_queue<node, vector<node>, greater<node> > q;
    q.push({0, s});
    dis[s][s] = 0;
    while (!q.empty()) {
        int u = q.top().u;
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (auto e : ed[u]) {
            int v = e.v;
            ll w = e.w;
            if (dis[s][v] > dis[s][u] + w) {
                dis[s][v] = dis[s][u] + w;
                q.push({dis[s][v], v});
            }
        }
    }
}
int main()
{
    cin >> n >> m;
    while (m--) {
        int u, v, w;
        cin >> u >> v >> w;
        ed[u].push_back({v, w});
    }
    for (int i = 1; i <= n; i++) {
        ed[0].push_back({i, 0});
    }
    spfa(0);
    for (int i = 1; i <= n; i++) {
        for (auto& j : ed[i]) {
            j.w += h[i] - h[j.v];
        }
    }
    memset(dis, 0x3f, sizeof(dis));
```

```
    for (int i = 1; i <= n; i++) {
        memset(vis, 0, sizeof(vis));
        dijkstra(i);
    }
    for (int i = 1; i <= n; i++) {
        ll ans = 0;
        for (int j = 1; j <= n; j++) {
            if (dis[i][j] == 0x3f3f3f3f3f3f3f3f) {
                dis[i][j] = 1e9;
            }
            else {
                dis[i][j] -= h[i] - h[j];
            }
            ans += dis[i][j] * j;
        }
        cout << ans << endl;
    }
}
```

11 欧拉路

欧拉路定义：从图中某个点出发，遍历整个图，图中每条边经过且只经过一次

首先用 dfs 或并查集判断图的连通性，然后判断图是否存在欧拉路

无向连通图的判断条件：如果图中的点全都是偶点，则存在欧拉回路。如果只有两个奇点，则存在欧拉路，其中一个奇点是起点，另一个是终点

有向连通图的判断条件：如果所有点的读书都是 0 ，则存在欧拉回路。如果有两个点的度数分别是 1 和 -1 ，则存在欧拉路，度数为 1 的是起点，为 -1 的是终点。

12 二分图匹配

1. 匈牙利算法 O(nm)

```
int main(){
  for(i = 1;i<=n1;i++){
    memset(vis,0,sizeof vis);
    //如果匹配成功，则匹配总数+1
    if(find(i))ans++;
  }
  printf("%d",ans);
  return 0;
}

//对于第x个，能够匹配返回true，不能匹配返回false
bool find(int x){
  for(int i = h[x];i!=-1;i = e[i].ne){
    int v = e[i].v;
    //如果曾经查找过第v个点，但是没有成功
    if(vis[v] == 0){
      vis[v] = 1;
      //第v个没有匹配或者第v个的匹配对象可以匹配成功
```

```
      if(match[v] == 0 || find(match[v])){
        match[v] = x; return true;
      }
    }
  }
  return false;
}
```

2. KM算法，求二分图的最大带权匹配 O(n^3)

```
using namespace std;
typedef long long ll;
typedef pair<ll,ll> PII;
const int N = 605;
const int inf = 0x3f3f3f3f;
PII a[N];
ll n,b[N],c[N];
ll w[N][N], lx[N],ly[N];
ll linker[N], slack[N];
bool visy[N];
ll pre[N];
void bfs(ll k){
  int i,j;
    ll x,y = 0,yy = 0,delta;
    memset(pre,0,sizeof(pre));
    for(i =1;i<=n;i++) slack[i] = inf;
    linker[y] = k;
    while(1){
        x = linker[y];delta = inf;visy[y] = true;
        for(i=1;i<=n;i++){
            if(!visy[i]){
                if(slack[i] > lx[x] + ly[i] - w[x][i] ){
                    slack[i] = lx[x] + ly[i] - w[x][i];
                    pre[i] = y;
                }
                if(slack[i] < delta) delta = slack[i], yy = i;
            }
        }
        for(i = 0;i <= n;i++){
            if(visy[i]) lx[linker[i]] -= delta , ly[i] += delta;
            else slack[i] -= delta;
        }
        y = yy ;
        if(linker[y] == -1) break;
    }
    while(y) linker[y] = linker[pre[y]], y = pre[y];
}
ll KM(){
  int i,j;
    memset(lx,0,sizeof(lx));
    memset(ly,0,sizeof(ly));
    memset(linker,-1,sizeof(linker));
    for(i = 1;i<=n;i++){
```

```
        memset(visy,false, sizeof(visy));
        bfs(i);
    }
    ll res = 0 ;
    for(i = 1;i<=n;i++){
        if(linker[i]!=-1){
            res += w[linker[i]][i];
        }
    }
    return res;
}
int main(){
  int i,j;
  while(cin>>n){
    for(i = 1;i<=n;i++){
      for(j = 1;j<=n;j++){
        scanf("%d",&w[i][j]);
      }
    }
    cout<<KM()<<endl;
  }
  return 0;
}
```

13 欧拉回路

```
//找出欧拉回路，并将欧拉回路的每条边存入栈中
//type==1,为无向图，type==2为有向图
void dfs(int u){
    int &i = h[u],j;
    for(;i!=-1;){
        int v = e[i].v;
        if(vis[i]){
            i = e[i].ne;
            continue;
        }
        vis[i] = 1;//标记为走过
        int t = i;
        if(type == 1){
            //偶数边为正，奇数边为负
            vis[i^1] = 1;
            t = (t+2)/2;
            if(i%2 == 1)    t = -t;
        }else{
            t = t+1;
        }
        i = e[i].ne;
        dfs(v);
        stac.push(t);
    }
}
```

四 字符串

1 manacher 求最长回文子串

```
int n, P[N << 1];
char a[N], S[N << 1];
void change()
{
    n = strlen(a);
    int k = 0; S[k++] = '$'; S[k++] = '#';
    for (int i = 0; i < n; i++) { S[k++] = a[i]; S[k++] = '#'; }
    S[k++] = '&';
    n = k;
}
void manacher()
{
    int R = 0, c;
    for (int i = 1; i < n; i++) {
        if (i < R) P[i] = min(P[(c << 1) - i], P[c] + c - i);
        else P[i] = 1;
        while (S[i + P[i]] == S[i - P[i]]) P[i]++;
        if (P[i] + i > R) {
            R = P[i] + i;
            c = i;
        }
    }
}
void solve()
{
    cin >> a; change();
    manacher();
    int ans = 1;
    for (int i = 0; i < n; i++) ans = max(ans, P[i]);
    cout << ans - 1 << endl;
    return;
}
```

2 KMP

```
const int N = 1e5 + 5;
int nex[N];
char s1[N]; //文本串
char s2[N]; //模式串
int n, m;//文本串和模式串的长度
//下标从1开始
void getnex() {
    nex[1] = 0;
    int i, j = 0;
    for (i = 2; i <= m; i++) {
        while (j > 0 && s2[j + 1] != s2[i])j = nex[j];
        //如果回跳到第一个字符就不 用再回跳了
```

```
        if (s2[j + 1] == s2[i])j++;
        nex[i] = j;
    }
}
int kmp() {
    int i, j = 0;
    getnex();
    for (i = 1; i <= n; i++) {
        while (s1[i] != s2[j + 1] && j > 0)j = nex[j];
        //如果失配 ，那么就不断向回跳，直到可以继续匹配
        if (s1[i] == s2[j + 1])j++;
        //如果匹配成功，那么对应的模式串位置++
        if (j == m) {
            //输出所有出现位置的起始下标
            printf("%d ", i - m + 1);
            //注意下标，是从0还是1
            j = nex[j];//继续匹配
        }
    }
}
```

五 树

1 树的直径

1. 两次 DFS （不能有负权边）

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e4 + 5;
int n, c, d[N]; // c为最深点
vector<int> ed[N];
void dfs(int u, int fa) {
    for (int v : ed[u]) {
        if (v == fa) continue;
        d[v] = d[u] + 1;
        if (d[v] > d[c]) c = v;
        dfs(v, u);
    }
}
int main()
{
    cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        ed[u].push_back(v);
        ed[v].push_back(u);
    }
    dfs(1, 0);
    d[c] = 0;
    dfs(c, 0); // 从直径的一端一路向上
```

```
    cout << d[c] << endl;
}
```

2. 树形 DP（允许负权）

```
// 记录当 1 为树的根时，每个节点作为子树的根向下，所能延伸的最长路径长度 d1 与次长路径（与最长路径无公共边）长度 d2
// 那么直径就是对于每一个点，该点 d1 + d2 能取到的值中的最大值。
#include <bits/stdc++.h>
using namespace std;
const int N = 1e4 + 5;
int n, d = 0;
int d1[N], d2[N];
vector<int> E[N];
void dfs(int u, int fa) {
    d1[u] = d2[u] = 0;
    for (int v : E[u]) {
        if (v == fa) continue;
        dfs(v, u);
        int t = d1[v] + 1;
        if (t > d1[u])
            d2[u] = d1[u], d1[u] = t;
        else if (t > d2[u])
            d2[u] = t;
    }
    d = max(d, d1[u] + d2[u]);
}
int main() {
    cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        E[u].push_back(v), E[v].push_back(u);
    }
    dfs(1, 0);
    cout << d << endl;
}

// 或
// 对于树的直径，实际上是可以通过枚举从某个节点出发不同的两条路径相加的最大值求出。因此，在 DP 求解的过程中，我们只需要在更新 dp[u] 之前，计算 d = max(d, dp[u] + dp[v] + w(u, v)) 即可算出直径 d。
int n, d = 0;
int dp[N];
vector<int> E[N];
void dfs(int u, int fa) {
  for (int v : E[u]) {
    if (v == fa) continue;
    dfs(v, u);
    d = max(d, dp[u] + dp[v] + 1);
    dp[u] = max(dp[u], dp[v] + 1);
  }
}
```

性质 若树上所有边权为正，则树的所有直径的中点相同

2 最近公共祖先 LCA（Lowest Common Ancestor）

1. 倍增法 O(n) 预处理，O(logn) 查询

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 5e5 + 5;
int n, m, s;
int fa[N][20], d[N]; // fa[x][i]表示x的第(2^i)个父亲 d[i]为深度
vector<int> ed[N];
void dfs(int u, int f) {
    d[u] = d[f] + 1;
    fa[u][0] = f;
    for (int i = 1; (1 << i) <= d[u]; i++)
        fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int v : ed[u]) {
        if (v != f)
            dfs(v, u);
    }
}
int LCA(int x, int y) {
    if (d[x] < d[y]) swap(x, y);
    for (int i = 19; i >= 0; i--) { // 把x和y提到相同高度
        if (d[x] - (1 << i) >= d[y])
            x = fa[x][i];
    }
    if (x == y) return x;
    for (int i = 19; i >= 0; i--) {
        if (fa[x][i] != fa[y][i]) {
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m >> s;
    for (int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        ed[u].push_back(v);
        ed[v].push_back(u);
    }
    dfs(s, 0);
    while (m--) {
        int a, b;
        cin >> a >> b;
```

```
        cout << LCA(a, b) << '\n';
    }
}
```

2. tarjan离线处理 O(n + m)

```
#include <bits/stdc++.h>
#define ll long long
#define PII pair<int, int>
using namespace std;
const int N = 5e5 + 5;
int n, m, s;
int fa[N], ans[N];
vector<int> ed[N];
vector<PII> query[N];
bool vis[N];
int find_set(int x) {
    int root = fa[x];
    while (root != fa[root]) root = fa[root];
    while (x != root) {
        int t = fa[x];
        fa[x] = root;
        x = t;
    }
    return root;
}
void tarjan(int u) {
    vis[u] = true;
    for (int v : ed[u]) {
        if (!vis[v]) {
            tarjan(v);
            fa[v] = u;
        }
    }
    for (auto it : query[u]) {
        int v = it.first;
        if (vis[v])
            ans[it.second] = find_set(v);
    }
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    cin >> n >> m >> s;
    for (int i = 1; i < n; i++) {
        fa[i] = i; // 并查集初始化
        int u, v;
        cin >> u >> v;
        ed[u].push_back(v);
        ed[v].push_back(u);
    }
    fa[n] = n;
```

```
    for (int i = 1; i <= m; i++) {
        int a, b;
        cin >> a >> b;
        query[a].push_back({b, i});
        query[b].push_back({a, i});
    }
    tarjan(s);
    for (int i = 1; i <= m; i++)
        cout << ans[i] << '\n';
}
```

3 树的重心

定义

如果在树中选择某个节点并删除，这棵树将分为若干棵子树，统计子树节点数并记录最大值。取遍树上所有节点，使此最大值取到最小的节点被称为整个树的重心。

性质

- 树的重心如果不唯一，则至多有两个，且这两个重心相邻。
- 以树的重心为根时，所有子树的大小都不超过整棵树大小的一半。
- 树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么到它们的距离和一样。
- 把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。
- 在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。

求解

在 DFS 中计算每个子树的大小，记录「向下」的子树的最大大小，利用总点数 - 当前子树（这里的子树指有根树的子树）的大小，
得到「向上」的子树的大小，然后就可以依据定义找到重心了。

```
const int N = 1e4 + 5;
int n;
int Size[N];  // 这个节点的「大小」（所有子树上节点数 + 该节点）
int weight[N];  // 这个节点的「重量」，即所有子树「大小」的最大值
int centroid[2];  // 用于记录树的重心（存的是节点编号）
vector<int> ed[N];
void GetCentroid(int cur, int fa) {  // cur 表示当前节点 (current)
    Size[cur] = 1;
    weight[cur] = 0;
    for (int v : ed[cur]) {
        if (v != fa) {  // v 表示这条有向边所通向的节点。
            GetCentroid(v, cur);
            Size[cur] += Size[v];
            weight[cur] = max(weight[cur], Size[v]);
        }
    }
    weight[cur] = max(weight[cur], n - Size[cur]);
    if (weight[cur] <= n / 2) {  // 依照树的重心的定义统计
        centroid[centroid[0] != 0] = cur;
```

```
    }
}
```

六 数据结构

1 Trie 字典树

```
template <typename Key, typename Value>
class TrieNode {
public:
    map<Key, TrieNode*> children; // 子节点字典树
    Value data; // 节点数据
    bool isEndOfWord; // 是否为单词结尾
};
template <typename Key, typename Value>
class Trie {
private:
    TrieNode<Key, Value>* root; // 根节点
public:
    Trie() {
        root = new TrieNode<Key, Value>(); // 初始化根节点
    }
    // 插入一个单词到字典树中
    void insert(string word) {
        TrieNode<Key, Value>* node = root;
        for (int i = 0; i < word.length(); i++) {
            Key key = word[i]; // 取单词的每个字符作为 key
            if (node->children.find(key) == node->children.end()) { // 如果子节点不存在，则创建新的子节点
                node->children[key] = new TrieNode<Key, Value>();
            }
            node = node->children[key]; // 移动到子节点处
        }
        node->data = word; // 将单词的值保存到节点中
        node->isEndOfWord = true; // 标记单词结尾
    }
    // 查找一个单词是否在字典树中
    bool search(string word) {
        TrieNode<Key, Value>* node = root;
        for (int i = 0; i < word.length(); i++) {
            Key key = word[i]; // 取单词的每个字符作为 key
            if (node->children.find(key) == node->children.end()) { // 如果子节点不存在，则返回 false
                return false;
            }
            node = node->children[key]; // 移动到子节点处
        }
        return (node != NULL && node->isEndOfWord); // 如果找到了单词且单词结尾标记为 true，则返回 true，否则返回 false
    }
};
```

2 并查集

```
int fa[N];
int find(int x)
{
    int root = fa[x];
    while (root != fa[root]) root = fa[root];
    while (x != root) {
        int t = fa[x];
        fa[x] = root;
        x = t;
    }
    return root;
}
void solve()
{
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) fa[i] = i;
    for (int i = 0; i < m; i++) {
        int x, y, z;
        cin >> z >> x >> y;
        if (z == 1) {
            int u = find(x);
            int v = find(y);
            fa[u] = fa[v];
        }
        else {
            if (find(x) == find(y)) {
                cout << "Y\n";
            }
            else {
                cout << "N\n";
            }
        }
    }
}
```

3 树状数组

单点修改、区间求和

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 5e5 + 5;
int tr[N], n;
int lowbit(int x) { return x & -x; }
void add(int x, int y) {
    for(; x <= n; x += lowbit(x))
        tr[x] += y;
}
```

```
int query(int x) {
    int ans = 0;
    for(; x; x -= lowbit(x))
        ans += tr[x];
    return ans;
}

int main()
{
    int q;
    cin >> n >> q;
    for(int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        add(i, x);
    }
    while(q--) {
        int op;
        cin >> op;
        if(op == 1) {
      int x, k;
            cin >> x >> k;
            add(x, k);
        }
        else {
            int x, y;
            cin >> x >> y;
            cout << query(y) - query(x - 1) << '\n';
        }
    }
    return 0;
}
```

区间修改, 单点查询

```
#include <bits/stdc++.h>
using namespace std;
const int N = 5e5 + 5;
int a[N];
int b[N], c[N];
int n, m;
int lowbit(int x) {
    return x & (-x);
}
void add(int x, int y) {
    int i, j;
    for (i = x; i <= n; i += lowbit(i))
        c[i] += y;
}
int sum(int x) {
    int i, res = 0;
    for (i = x; i > 0; i -= lowbit(i))
        res += c[i];
```

```
    return res;
}
int main() {
    ios::sync_with_stdio(0), cin.tie(0);
    int i, j;
    cin >> n >> m;
    for (i = 1; i <= n; i++) {
        cin >> a[i];
        b[i] = a[i] - a[i - 1];
        add(i, b[i]);
    }
    for (i = 1; i <= m; i++) {
        string s;
        cin >> s;
        if (s[0] == 'C') {
            int l, r, d;
            cin >> l >> r >> d;
            add(l, d); add(r + 1, -d);
        }
        else {
            int t;
            cin >> t;
            cout << sum(t) << endl;
        }
    }
    return 0;
}
```

区间修改, 区间查询

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 5e5 + 5;
int n, m;
ll b[N], a[N], c1[N], c2[N];
int lowbit(int x) {
    return x & (-x);
}
void add(ll x, ll y) {
    for (int i = x; i <= n; i += lowbit(i)) {
        c1[i] += y;
        c2[i] += y * x;
    }
}
ll ask(ll x) {
    ll i, res = 0;
    for (i = x; i > 0; i -= lowbit(i))
        res += (x + 1) * c1[i] - c2[i];
    return res;
}
int main() {
    ios::sync_with_stdio(0);
```

```
    int i, j;
    cin >> n >> m;
    for (i = 1; i <= n; i++) {
        cin >> a[i];
        b[i] = a[i] - a[i - 1];
        add(i, b[i]);
    }
    for (i = 1; i <= m; i++) {
        string s; ll l, r, d;
        cin >> s;
        if (s[0] == 'C') {//区间修改
            cin >> l >> r >> d;
            add(l, d);
            add(r + 1, -d);
        }
        else {//区间查询
            cin >> l >> r;
            cout << ask(r) - ask(l - 1) << endl;
        }
    }
    return 0;
}
```

4 线段树

线段树模板

```
#include <bits/stdc++.h>
#define lson (x << 1)
#define rson (x << 1 | 1)
#define ll long long
const int N = 1e5 + 5;
int n, m;
ll a[N], d[N << 2], lazy[N << 2];
void build(int l, int r, int x) {
    if (l == r) { d[x] = a[l]; return; }
    int mid = ((r - l) >> 1) + l;
    build(l, mid, lson);
    build(mid + 1, r, rson);
    d[x] = d[lson] + d[rson];
}
void update(int l, int r, int x, ll k, int L, int R) {
    if (l <= L && R <= r) {
        d[x] += k * (R - L + 1);
        lazy[x] += k;
        return;
    }
    int mid = ((R - L) >> 1) + L;
    if (lazy[x] && L != R) {
        d[lson] += lazy[x] * (mid - L + 1);
        d[rson] += lazy[x] * (R - mid);
        lazy[lson] += lazy[x];
        lazy[rson] += lazy[x];
```

```
        lazy[x] = 0;
    }
    if (l <= mid) update(l, r, lson, k, L, mid);
    if (r >= mid + 1) update(l, r, rson, k, mid + 1, R);
    d[x] = d[lson] + d[rson];
}
ll get(int l, int r, int x, int L, int R) {
    if (l <= L && R <= r) {
        return d[x];
    }
    int mid = ((R - L) >> 1) + L;
    ll res = 0;
    if (lazy[x]) {
        d[lson] += lazy[x] * (mid - L + 1);
        d[rson] += lazy[x] * (R - mid);
        lazy[lson] += lazy[x];
        lazy[rson] += lazy[x];
        lazy[x] = 0;
    }
    if (l <= mid) res += get(l, r, lson, L, mid);
    if (r >= mid + 1) res += get(l, r, rson, mid + 1, R);
    return res;
}
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0);
    std::cin >> n >> m;
    for (int i = 1; i <= n; i++) std::cin >> a[i];
    build(1, n, 1);
    while (m--) {
        int op, l, r;
        std::cin >> op >> l >> r;
        if (op == 1) {
            ll k;
            std::cin >> k;
            update(l, r, 1, k, 1, n);
        }
        else {
            std::cout << get(l, r, 1, 1, n) << std::endl;
        }
    }
}
```

5 分块

区间加法、单点查值

```
int n, par; // par为区块长度
int belong[N]; // 记录所属区块
int l[MAXPAR], r[MAXPAR];
int part[MAXPAR];
void build() {
    par = sqrt(n);
```

```
    int cnt = n / par;
    int ptop = 1;
    for(int i = 1; i <= cnt; i++) {
        l[i] = ptop;
        for(int j = 1; j <= par; j++)
            belong[ptop++] = i;
        r[i] = ptop - 1;
    }
    while(ptop <= n) // 最右侧多余的部分
        belong[ptop++] = cnt;
    r[cnt] = n;
}
void add(int L, int R, int c) {
    if(belong[L] == belong[R]) {
        for(int i = L; i <= R; i++)
            a[i] += c;
    }
    else {
        for(int i = L; i <= r[belong[L]]; i++)
            a[i] += c;
        for(int i = l[belong[R]]; i <= R; i++)
            a[i] += c;
        for(int i = belong[L] + 1; i <= belong[R] - 1; i++)
            part[i] += c;
    }
}
```

6 主席树(区间第k小模版)

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e5 + 5;
const int M = 2e5 + 5;
int n, m;
int a[N], idx;
vector<int> nums;//去重数组
struct Node {
    int l, r;//左右子节点编号
    int cnt;
}tr[N * 4 + N * 18];//开N*4+N*lgN个空间
int root[N];//一共需要保存N个历史记录
int find(int x) {
    int l = 0, r = nums.size() - 1;
    while (l != r) {
        int mid = (l + r) / 2;
        if (nums[mid] >= x)  r = mid;
        else    l = mid + 1;
    }
    return l;
}
int build(int l, int r) {//建树，cnt都为0
    int q = ++idx;
```

```
    if (l == r)  return q;
    int mid = (l + r) / 2;
    tr[q].l = build(l, mid);
    tr[q].r = build(mid + 1, r);
    return q;
}
int insert(int p, int l, int r, int x) {
    int q = ++idx;
    tr[q] = tr[p];
    if (l == r) {
        tr[q].cnt++;
        return q;
    }
    int mid = (l + r) / 2;
    //应该更新哪一个值域区间
    if (x <= mid)
        tr[q].l = insert(tr[p].l, l, mid, x);
    else
        tr[q].r = insert(tr[p].r, mid + 1, r, x);
    tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
    return q;
}
int query(int q, int p, int l, int r, int k) {
    if (l == r)  return r;
    //有多少个数落在值域为[l,mid]中
    int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;
    int mid = (l + r) / 2;
    if (k <= cnt)
        return query(tr[q].l, tr[p].l, l, mid, k);
    else
        return query(tr[q].r, tr[p].r, mid + 1, r, k - cnt);
}
int main() {
    int i, j;
    cin >> n >> m;
    for (i = 1; i <= n; i++) {
        cin >> a[i];
        nums.push_back(a[i]);
    }
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    root[0] = build(0, nums.size() - 1);//下标从0开始
    for (i = 1; i <= n; i++) {
        root[i] = insert(root[i - 1], 0, nums.size() - 1, find(a[i]));
        //建立可持久话线段树
    }
    for (i = 1; i <= m; i++) {
        int l, r, k;
        cin >> l >> r >> k;
        cout << nums[query(root[r], root[l - 1], 0, nums.size() - 1, k)] << endl;
    }
    return 0;
}
```

七 组合数学

1 斯特林数

第二类斯特林数：S(n,k), 把 n 个不同的球放到 k 个相同的盒子里，不能有空盒子，有多少种分法

- 求 S(n, m)

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const ll N = 2e5 + 5;
const ll inf = 1ll << 62;
const ll mod = 1e9 + 7;
ll fac[100010], inv[100010];
void add(ll& a, ll b) {
    a += b;
    if (a >= mod) a -= mod;
}
ll gcd(ll a, ll b) {
    return b ? gcd(b, a % b) : a;
}
ll qmi(ll a, ll b) {
    ll res = 1;
    for (; b; b >>= 1) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
    }
    return res;
}
ll f(string s, string t) {
    ll p = 0;
    for (ll i = 0; i < s.size(); i++)
        if (p < t.size() && s[i] == t[p]) {
            p++;
        }
    return (p == t.size());
}
void sub(ll& a, ll b) {
    a -= b;
    if (a < 0) a += mod;
}
ll C(ll a, ll b) {
    return fac[a] * inv[b] % mod * inv[a - b] % mod;
}
void solve() {
    ll n, m; cin >> n >> m;
    if (m > n)
        cout << 0 << endl;
    else {
        ll ans = 0;
        for (int i = 0; i <= m; i++) {
            if ((m - i) % 2 == 0) {
```

```
                add(ans, qmi(i, n) * inv[i] % mod * inv[m - i] % mod);
            }
            else {
                sub(ans, qmi(i, n) * inv[i] % mod * inv[m - i] % mod);
            }

        }
        cout << ans << endl;
    }
}
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);
    fac[0] = inv[0] = 1;
    for (int i = 1; i <= 100000; i++)
        fac[i] = fac[i - 1] * i % mod;
    inv[100000] = qmi(fac[100000], mod - 2);
    for (int i = 100000; i >= 1; i--)
        inv[i - 1] = inv[i] * i % mod;
    solve();
}
```

2 卡特兰数

$$f(n) = C_{2n}^{n} - C_{2n}^{n-1} \tag{4}$$

$$f(n) = \sum_{i=0}^{n-1} f(i) * f(n-i-1) \tag{5}$$

$$h(n) = \frac{C_{2n}^{n}}{n+1} \tag{6}$$

3 错排公式

假设有 n 个元素，n 个位置，每个元素都有自己唯一的正确位置，问所有元素都处在错误位置有多少可能?

$$D(n) = (n-1) * (D(n-1) + D(n-2)), D(1) = 0, D(2) = 1 \tag{7}$$

八 动态规划

1 01背包

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int main()
{
    int f[1005] = {0};
    int n, m;
    cin >> n >> m;
    vector<int> w(n);
    vector<int> v(n);
```

```
    for (int i = 0; i < n; i++) cin >> w[i] >> v[i];
    for (int i = 0; i < n; i++) {
        for (int j = m; j >= w[i]; j--) {
            f[j] = max(f[j], f[j - w[i]] + v[i]);
        }
    }
    cout << f[m];
}
```

2 完全背包

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int f[1005];
int main()
{
    int n, m;
    cin >> n >> m;
    vector<int> w(n), v(n);
    for (int i = 0; i < n; i++) cin >> w[i] >> v[i];
    for (int i = 0; i < n; i++) {
        for (int j = w[i]; j <= m; j++) {
            f[j] = max(f[j], f[j - w[i]] + v[i]);
        }
    }
    cout << f[m];
}
```

3 多重背包-二进制优化

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int f[2005];
int main()
{
    int n, m;
    cin >> n >> m;
    vector<int> w, v, s;
    int a, b, c;
    for (int i = 0; i < n; i++) {
        cin >> a >> b >> c;
        int k = 1;
        while (k <= c) {
            w.push_back(k * a);
            v.push_back(k * b);
            c -= k;
            k *= 2;
        }
        if (c) {
```

```
            w.push_back(c * a);
            v.push_back(c * b);
        }
    }
    n = w.size();
    for (int i = 0; i < n; i++) {
        for (int j = m; j >= w[i]; j--) {
            f[j] = max(f[j], f[j - w[i]] + v[i]);
        }
    }
    cout << f[m];
}
```

4 状压DP

我们用一个整数表示一个状态，它的二进制形式记录了某个物品是否被选择了

如状态`10`在二进制中为`1010`, 表示选择了第 1 个和第 3 个物品（下标从 0 开始，二进制位从右至左）. 若要在`10`的基础上选择物品 3, 则状态变为`1110`, 即`14`.

状态的转移可以通过位运算来实现，即`dp[x | (1 << j)] -> dp[y]`.

例题: P1433 吃奶酪

房间里放着 n 块奶酪。一只小老鼠要把它们都吃掉，问至少要跑多少距离？老鼠一开始在 (0,0) 点处。

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 2e5 + 5;
int n;
double x[20], y[20], dis[20][20];
double dp[(1 << 16) + 5][20]; // dp[i][x] 表示经过点的状态为 i，最后一个点是 j
int main()
{
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> x[i] >> y[i];
    }
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            dis[i][j] = dis[j][i] = sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
        }
    }
    for (int i = 0; i < (1 << n); i++) { // 初始化
        for (int j = 0; j < n; j++) {
            dp[i][j] = 1e7;
        }
    }
    for (int i = 0; i < n; i++) { // 初始化原点到点 i 的距离
        dp[1 << i][i] = sqrt(x[i] * x[i] + y[i] * y[i]);
    }
    for (int i = 0; i < (1 << n); i++) { // 状压 dp
```

```
        for (int j = 0; j < n; j++) {
            if ((i & (1 << j)) == 0) continue; // 状态 i 没有经过 j
            for (int k = 0; k < n; k++) {
                if (j == k) continue;
                if ((i & (1 << k)) == 0) continue; // 状态 i 没有经过 k
                // 从 j 到 k 和从 k 到 j 取最小
                dp[i][k] = min(dp[i][k], dp[i - (1 << k)][j] + dis[j][k]);
            }
        }
    }
    double ans = 1e9;
    for (int i = 0; i < n; i++)
        ans = min(ans, dp[(1 << n) - 1][i]);
    cout << fixed << setprecision(2) << ans;
}
```

5 数位DP

P2602 ZJOI2010 数字计数

给定两个正整数 a 和 b，求在 [a,b] 的所有整数中，每个数码各出现了多少次

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
ll a, b, dp[15], ten[15];
// dp[i] 表示在 i 位数中每个数出现多少次，如 1、2、3 位数中每个数分别出现 1、20、300 次
// ten[i] 表示 10 的 i 次方
ll cnta[15], cntb[15];
void dfs(ll* cnt, ll x) {
    vector<int> v;
    v.push_back(0);
    ll t = x;
    while (t) {
        v.push_back(t % 10);
        t /= 10;
    }
    int n = v.size() - 1;
    for (int i = n; i >= 1; i--) { // 从高位到低位依次处理
        // 以 324 为例
        // v[3] = 3，先计算 0xx、1xx、2xx，即 dp[2]*v[3]=20*3
        for (int j = 0; j <= 9; j++) cnt[j] += dp[i - 1] * v[i];
        // 再计算从 1 到 v[i]-1 在第 i 位出现的次数，如 100-199 中 1 出现了 100 次
        for (int j = 0; j < v[i]; j++) cnt[j] += ten[i - 1];
        // 再计算 300-324，不完整的 i 位数（完整的为 00-99）中 3 出现的次数
        ll num = 0;
        // 得到 24
        for (int j = i - 1; j >= 1; j--) num = num * 10 + v[j];
        // 还有 300 贡献的一个 3
        cnt[v[i]] += num + 1;
        cnt[0] -= ten[i - 1]; // 去掉前导 0
    }
}
```

```
int main()
{
    cin >> a >> b;
    ten[0] = 1;
    for (int i = 1; i <= 13; i++) { // init
        dp[i] = ten[i - 1] * i;
        ten[i] = ten[i - 1] * 10;
    }
    dfs(cnta, a - 1);
    dfs(cntb, b);
    for (int i = 0; i <= 9; i++) {
        cout << cntb[i] - cnta[i] << " ";
    }
}
```

P2657 SCOI2009 windy 数

不含前导零且相邻两个数字之差至少为 2 的正整数被称为 windy 数。windy 想知道，在 a 和 b 之间，包括 a 和 b ，总共有多少个 windy 数?

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
int a, b;
int ub[12], len; // upper_bound 表示第 i 为能取的最大数，从 1 开始是最高位
int dp[15][15];
// flag 表示上一位是否达到上限，若 flag=1 则当前为也只能取到 upper bound
int dfs(int pos, int pre, bool flag) { // pre = 11 表示前一位是前导 0
    if (pos <= 0) return 1;
    int max_num;
    if (!flag && dp[pos][pre] != -1) {
        return dp[pos][pre];
    }
    if (flag) max_num = ub[pos];
    else max_num = 9;
    int res = 0;
    for (int i = 0; i <= max_num; i++) {
        if (abs(i - pre) >= 2) {
            if (pre == 11 && i == 0) // 当前仍然属于前导 0
                res += dfs(pos - 1, pre, flag && (i == ub[pos]));
            else
                res += dfs(pos - 1, i, flag && (i == ub[pos]));
        }
    }
    if (!flag) dp[pos][pre] = res;
    return res;
}
int solve(int x) {
    memset(dp, -1, sizeof(dp));
    len = 0;
    int t = x;
    while (t) {
        ub[++len] = t % 10;
```

```
        t /= 10;
    }
    return dfs(len, 11, true);
}
int main()
{
    cin >> a >> b;
    int x = solve(a - 1);
    int y = solve(b);
    cout << y - x << endl;
}
```

STL

1 set

```
s.earse(it1,it2);删除[it1,it2) 这个区间（迭代器）对应位置的元素
iterator lower_bound( const key_type &key ); 返回一个迭代器，指向键值 >= key的第一个元素。
iterator upper_bound( const key_type &key ); 返回一个迭代器，指向键值 > key的第一个元素

Set的定义：
struct T1{
  int key,value;
  bool operator < (const T1 &a) const {
    return key<a.key;//按照升序排列
  }
};
struct T2{
  int key,value;
};
struct T2cmp{
  bool operator () (const int &a,const int &b){
    if(abs(a-b)<=k)
      return false;
    return a<b;
  }
};
int main(){
  int i,j;
  set<T1> s1;
  set<T2,T2cmp> s2;
  set<string> ss1;//等于set<string,less<int> > ss1;从小到大
  set<string,greater<string> > ss2;//从大到小
  set<string,greater<string> > ::iterator itsey;
  //set的遍历
  set<string> :: iterator it;
  for(it = ss1.begin();it!=ss1.end();it++){
    cout<<*it<<endl;
  }
  return 0;
}
Multiset
```

c.erase(elem) 删除与elem相等的所有元素，返回被移除的元素个数。
c.erase(pos) 移除迭代器pos所指位置元素，无返回值。

2 map

mp.find(key):在map中查找key并返回其iterator,找不到的话返回mp.end() O(logn)
mp.count(key):在map中找key的数量，由于每个key都是唯一的，只会返回0或1
mp[key] 可以直接访问到键值队key---value中的value，如果不存在，则mp[key]返回的是value类型默认构造器所构造的值，并将该键值对插入到map中
哈希表:

```
#include <unordered_map>
unordered_map
```

3 pair

```
#include <utility>头文件中
pair<T1, T2> p1; //创建一个空的pair对象（使用默认构造），它的两个元素分别是T1和T2类型，采用值初始化。
pair<T1, T2> p1(v1, v2); //创建一个pair对象，它的两个元素分别是T1和T2类型，其中first成员初始化为v1，
second成员初始化为v2。
make_pair(v1, v2); // 以v1和v2的值创建一个新的pair对象，其元素类型分别是v1和v2的类型。
p1 < p2; // 两个pair对象间的小于运算，其定义遵循字典次序
// 如 p1.first < p2.first 或者 !(p2.first < p1.first) && (p1.second < p2.second) 则返回true。
p1 == p2// 如果两个对象的first和second依次相等，则这两个对象相等；该运算使用元素的==操作符。
```

4 vector

```
v.resize(n); // 把v的长度重新设定为n个元素
v.erase(unique(v.begin(), v.end()), v.end()); // 去重
```

5 priority_queue

pq.top():获取优先级最高的元素O(1)
优先队列的定义:

```
priority_queue<int> q1; //默认从大到小，大顶堆
priority_queue<int ,vector<int>,less<int> > q2; //降序队列，大顶堆
priority_queue<int ,vector<int>,greater<int> > q3; //升序队列，小顶堆
```

对于结构体定义:

```
struct T1{//法一 强烈推荐
  int x,y;
  friend bool operator < (T1 a,T1 b){
    return a.x<b.x;
  }
};
priority_queue<T1> q1;

struct T1{//法二
  int x,y;
  bool operator < (const T1 &a) const{
```

```
    return x<a.x; //大顶堆
  }
};
priority_queue<T1> q1;
```

其他

lower_bound(begin,end,num)：二分查找第一个大于或等于num的数字，找到返回该数字的地址，不存在则返回end
upper_bound(begin,end,num)：二分查找第一个大于num的数字
在从大到小的排序数组中，重载lower_bound()和upper_bound()
lower_bound(begin,end,num,greater()):二分查找第一个小于或等于num的数字
upper_bound(begin,end,num,greater()):二分查找第一个小于num的数字