ACM 模板

sy

2021 年 12 月 4 日

目录

第一章　基础算法

1.1　快速输入输出

</> 代码 1.1:　/基础算法/read

```
#include <bits/stdc++.h>
#define IOS ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
__int128 read() {
    __int128 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 * 10 + ch - '0'; ch = getchar(); }
    return x * f;
}
void write(__int128 x) {
    if (x < 0) { putchar('-'); x = -x; }
    if (x > 9) write(x / 10);
    putchar(x % 10 + '0');
}
```

1.2　排序

本场最佳 sort(begin(), end())

</> 代码 1.2:　/基础算法/sort

```
void quick_sort(int l, int  r) {
    if(l == r) return ;
    int k = (num[l] + num[r]) >> 1, i = l - 1, j = r + 1;
    while(i < j) {
        do i ++ ; while(num[i] < k);
        do j -- ; while(num[j] > k);
        if(i < j) swap(num[i], num[j]);
    }
    quick_sort(l, j); quick_sort(j + 1, r);
}
int num[M], tmp[M];
void merge_sort(int l, int r) {
```

```
    if(l >= r)return;
    int mid = (l + r)>>1;
    merge_sort(l, mid), merge_sort(mid+1, r);
    int i = l, j = mid + 1, k = 0;
    while(i <= mid && j <= r) {
        if(num[i] <= num[j]) tmp[k++] = num[i++];
        else tmp[k++] = num[j++];
    }
    while(i <= mid) tmp[k++] = num[i++];
    while(j <= r)   tmp[k++] = num[j++];
    for(int x = l, y = 0; x <= r;) num[x++] = tmp[y++];
}
```

1.3 求第 k 大数

</> 代码 1.3: /基础算法/kth-element

```
int kth_element(int l, int r, int k) {
    if(l == r) return num[r];
    int mid = (num[l] + num[r]) >> 1, i = l - 1, j = r + 1;
    while(i < j) {
        do i ++; while(num[i] < mid);
        do j --; while(num[j] > mid);
        if(i < j) swap(num[i], num[j]);
    }
    if(k <= j) return kth_element(l, j, k);
    return kth_element(j + 1, r, k);
}
// 与快排相比，求第K个数每次只需要递归一半，所以复杂度为 O(N);
// STL函数: nth_element(begin(), begin()+k-1, end());
```

1.4 求逆序对

</> 代码 1.4: /基础算法/inversion-pair

```
// 求逆序对 O(nlogn)
int num[M], tmp[M], res;
void merge_sort(int l, int r) {
    if(l >= r)return;
    int mid = (l + r)>>1;
    merge_sort(l, mid), merge_sort(mid+1, r);
    int i = l, j = mid + 1, k = 0;
    while(i <= mid && j <= r) {
        if(num[i] <= num[j]) tmp[k++] = num[i++];
```

```
        else {
            tmp[k++] = num[j++];
            res += mid - i + 1; //res的值就是逆序数
        }
    }
    while(i <= mid) tmp[k++] = num[i++];
    while(j <= r)   tmp[k++] = num[j++];
    for(int x = l, y = 0; x <= r;) num[x++] = tmp[y++];
}
// 归并排序可用来求逆序对
// 原理：因为在合并过程中，当num[i] > num[j]，在前半部分中比num[i]大的都 > num[j]，将num[j]
    放在 num[i]前面的话，逆序对需要加上 mid - i + 1;
// 相比较于树状数组求逆序对，归并排序不需要离散化。
```

1.5 C++ 高精度

</> 代码 1.5: /基础算法/high-accuracy

```
vector<int> add(vector<int> &a, vector<int> &b) {
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size() || i < b.size(); i++) {
        if (i < a.size()) 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;
}
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 < (int)a.size(); i++) { //保证a > b
        t = a[i] - t;
        if (i < (int)b.size()) t -= b[i];
        c.push_back((t + 10) % 10); //同时覆盖t>=0, t<0两种情况
        if (t < 0) t = 1;
```

```
27          else t = 0;
28      }
29      while (c.size() > 1 && c.back() == 0)
30          c.pop_back(); //去掉前导0;
31      return c;
32  }
33  vector<int> mul(vector<int> &a, int b) {
34      vector<int> c;
35      int t = 0;
36      for (int i = 0; i < (int)a.size() || t; i++) {
37          if (i < (int)a.size())
38              t += a[i] * b;
39          c.push_back(t % 10);
40          t /= 10;
41      }
42      while (c.size() > 1 && c.back() == 0) c.pop_back();
43      return c;
44  }
45  vector<int> div(vector<int> &a, int b, int &mod) {// r为余数
46      vector<int> c;
47      mod = 0;
48      for (int i = a.size() - 1; i >= 0; i--) {
49          mod = mod * 10 + a[i];
50          c.push_back(mod / b);
51          mod %= b;
52      }
53      reverse(c.begin(), c.end());
54      while (c.size() > 1 && c.back() == 0) c.pop_back();
55      return c;
56  }
57  void print_vec(vector<int> now) { //倒着输出vector
58      for (int i = now.size() - 1; ~i; i--)
59          cout << now[i];
60      cout << endl;
61  }
62  vector<int> read() {//读入字符串，转化为vector类型（里面是逆序）
63      string s; cin >> s;
64      vector<int> a;
65      for (int i = s.size() - 1; ~i; i--)
66          a.push_back(s[i] - '0');
67      return a;
68  }
69  int main() {
70      int op, num, mod;
```

```
    cin >> op;
    if (op == 1) {
        vector<int> a = read(), b = read();
        vector<int> c = add(a, b);
        print_vec(c);
    } else if (op == 2) {
        vector<int> a = read(), b = read();
        if (!cmp(a, b)) { //减法情况需要先判断两个数的大小，决定符号
            swap(a, b);
            printf("-");
        }
        vector<int> c = sub(a, b);
        print_vec(c);
    } else if (op == 3) {
        vector<int> a = read();
        cin >> num;
        vector<int> c = mul(a, num);
        print_vec(c);
    } else if (op == 4) {
        vector<int> a = read();
        cin >> num;
        vector<int> c = div(a, num, mod);//mod 为 余数
        cout << mod << endl;
        print_vec(c);
    }
    return 0;
}
```

1.6 python 读入

</> 代码 1.6: /基础算法/python-read

```
# python 应用
#strip() 去掉左右两端的空白符，返回 str
#slipt() 把字符串按空白符拆开，返回 [str]
#map 把 list 里面的值映射到指定类型，返回 [type]
# 有多组输入数据，但没有具体的告诉你有多少组，只是让你对应每组输入，应该怎样输出。
while True:
    try:
        a, b = map(int,input().strip().split())
        print (a+b)
    #  print ('\n') 每组输出后面加多加换行
    except EOFError:
        break
```

```python
# 输入一个整数，告诉我们接下来有多少组数据，然后在输入每组数据的具体值。
case = int(input().strip())
for i in range(case):
    a, b = map(int,input().strip().split())
    print (a+b)
# 有多组输入数据，没有具体的告诉你有多少组,但是题目却告诉你遇见什么结束
while True:
    a, b = map(int, input().strip().split())
    if a == 0 and b == 0:
        break
    print (a + b)
# 输入有多组，并却题目告诉你每组输入遇见什么结束，与第三种不同之处在于，每组输入都有相应的细化。
case = int(input())
for i in range(case):
    a, b = map(int, input().strip().split())
    if a == 0 and b == 0:
        break
    print (a + b)
# 这次的输入实现输入一个整数，告诉我们有多少行，在输入每一行。对于每一行的输入，有划分为第一个数和其他的数，第一个数代表那一组数据一共有多少输入。
case=int(input())
for i in range(case):
    data = list(map(int, input().split()))
    sum = 0
    for j in data:
        sum += j
    print (sum)
# 有多种输入数据，对于每组输入数据的第一个数代表该组数据接下来要输入数据量
while True:
    try:
        data = list(map(int, input().split()))
        n, array = data[0], data[1:]
        sum = 0
        for i in range(n):
            sum += array[i]
        print(sum)
    except EOFError:
        raise
# 这种类型的输出注意的就是换行，这类题目说在输出样例中，每组样例之间有什么什么，所以我们在对应输出的同时要判断一下是否是最后一组输出，如果不是，就 将题目所说的东西输出（一般是换行或空格），如果是，就直接结束。
while True:
    data = input()
```

```
    if data.isspace() == True:
        break
    else:
        data = list(map(int, input().split()))
        n, array = data[0], data[1:]
        sum = 0
        for i in range(n):
            sum += array[i]
        print (sum)
```

1.7 java 大数

</> 代码 1.7: /基础算法/java-rehigh-accuracyad

```
//！！！！提交JAVA的时候一定要去掉package
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Scanner;
        Scanner cin=new Scanner(System.in);
        BigInteger num1=new BigInteger("12345");
        BigInteger num2=cin.nextBigInteger();
        BigDecimal num3=new BigDecimal("123.45");
        BigDecimal num4=cin.nextBigDecimal();
//整数
import java.math.BigInteger;
public class Main {
    public static void main(String[] args) {
        BigInteger num1=new BigInteger("12345");
        BigInteger num2=new BigInteger("45");
        System.out.println(num1.add(num2));      //加法
        System.out.println(num1.subtract(num2)); //减法
        System.out.println(num1.multiply(num2)); //乘法
        System.out.println(num1.divide(num2));   //除法(相除取整)
        System.out.println(num1.mod(num2));      //取余
        System.out.println(num1.gcd(num2));      //最大公约数GCD
        System.out.println(num1.abs());          //取绝对值
        System.out.println(num1.negate());       //取反
        System.out.println(num1.max(num2));      //取最大值
        System.out.println(num1.min(num2));      //取最小值
        System.out.println(num1.equals(num2));   //是否相等
    }
}
//浮点数
import java.math.BigDecimal;
```

```java
public class Main {
    public static void main(String[] args) {
        BigDecimal num1=new BigDecimal("123.45");
        BigDecimal num2=new BigDecimal("4.5");
        System.out.println(num1.add(num2));        //加法
        System.out.println(num1.subtract(num2));   //减法
        System.out.println(num1.multiply(num2));   //乘法
        //除法 (在divide的时候就设置好要精确的小数位数和舍入模式)
        System.out.println(num1.divide(num2,10,BigDecimal.ROUND_HALF_DOWN);
        System.out.println(num1.abs());            //取绝对值
        System.out.println(num1.negate());         //取反
        System.out.println(num1.max(num2));        //取最大值
        System.out.println(num1.min(num2));        //取最小值
        System.out.println(num1.equals(num2));     //是否相等
        System.out.println(num2.compareTo(num1)); //判断大小( > 返回1, < 返回-1)
    }
}
```

第二章　字符串

2.1　前缀数组

pi 数组表示 $[0 \dots i]$ 中最长的相等的真前缀与真后缀的长度, 复杂度 $O(n)$
每一位 i 的值为 $i-1$ 的 $+1$ 、相等、减少，令 j 为当前匹配位置，则转移方程为 $j^{(n)} = \pi[j^{(n-1)} - 1]$

</> 代码 2.1:　/字符串/前缀数组

```
int pi[M];
void prefix_function(char s[]) {
    for(int i = 1; s[i]; i ++) {
        int j = pi[i - 1];
        while(j > 0 && s[j] != s[i]) j = pi[j - 1];
        if(s[i] == s[j]) j ++;
        pi[i] = j;
    }
}
```

2.2　最小表示法

用来找出字符串 S （或数组）的循环同构串中字典序最小 (大) 的一个。

</> 代码 2.2:　/字符串/最小表示法

```
int n, S[300009];
int Min_show() {
    int i = 0, j = 1, k = 0;
    while (i < n && j < n && k < n) {
        if (S[(i + k) % n] == S[(j + k) % n])
            k ++;
        else {
            if (S[(i + k) % n] > S[(j + k) % n])
                i += k + 1;
            else
                j += k + 1;
            if (i == j)
                i++;
            k = 0;
        }
```

```
    }
    return min(i, j); //返回最小表示
}
```

2.3 Hash

2.3.1 字符串 Hash

字符串 hash 相当于把字符串变成 base 进制的数。

</> 代码 2.3: /字符串/字符串 Hash

```
typedef unsigned long long ull;
const ull base = 131, M = 1e5 + 7, mod = 1e9 + 7;
char s[M];
ull myhash[M], power[M]; //myhash 里面存算到某位时的 Hash 值, power 存 base 的幂次 (子串查询才需要)
ull get_hash(char *s) {
    ull value = 0;
    power[0] = 1;
    for (int i = 1; s[i]; i++) {
        value = (value * base + s[i]) % mod;
        myhash[i] = value;
        power[i] = power[i - 1] * base % mod;
    }
    return value;//返回最终字符串的 Hash 值
}
//计算某段的 Hash 值
ull get_sub_hash(int l, int r) {
    return (myhash[r] - (myhash[l - 1] * power[r - l + 1]) % mod + mod) % mod;
}
/*扩展
字符串hash其实和进制进位有一些相似, 我们想到了这一点就可以线性的时间正着求前缀字符串的hash值和前缀字符串的倒转的hash值
当我们求正序前缀的hash的时候, 每一次操作就相当于在字符串的最前面插入一个值, 所以我们要首先处理出每一位的指数级ans,然后f[i] = f[i - 1] + str[i] * ans;
当我们求逆序前缀的hash值的时候, 每一次操作就相当于在字符串的最后插入一个值, 整个字符串就要左移一位然后把这个位塞进去 也就是f [i]= f[i - 1] * tmp + str[i];
*/
//例题 判断一个字符串的每一个前缀是否相等, abaehjsd -> aba(正反)
ULL ans = 1;
ULL f1[maxn],f2[maxn];
char str[maxn];
for(int i = 1 ; i <= l; i ++){
    if (str[i] >= '0' && str[i] <= '9')str[i] = str[i] - '0';
    else if (str[i] >= 'a' && str[i] <= 'z')str[i] = str[i] - 'a' + 10;
```

```
31      else str[i] = str[i] - 'A' + 36;
32      f1[i] = f1[i - 1] * 131 + str[i];
33      f2[i] = (f2[i - 1] + str[i] * ans);
34      ans *= 131;
35      if (f1[i] == f2[i]){
36          puts("i is equal");
37      }
38  }
```

2.3.2 图上 Hash

当一个点和他连到的点组成的集合和另一个点和他连到的点组成的集合完全相同的话，那么他们的 Hash 值相同

</> 代码 2.4: /字符串/图上 Hash

```
1  ULL Hash[maxn],id[maxn];
2  for(int i = 1; i <= N ; i ++ ) Hash[i] = id[i] = id[i - 1] * 3;
3  for(int i = 1; i <= M ; i ++ ){
4      int u,v; Sca2(u,v);
5      add(u,v); add(v,u);
6      Hash[u] += id[v]; Hash[v] += id[u];
7  }
```

2.4 KMP 算法

2.4.1 KMP

预处理转移数组，转移数组的本质是匹配字符串的前缀数组，复杂度 $O(n+m)$

</> 代码 2.5: /字符串/KMP

```
1  int ne[N]; //转移值
2  char s[N], p[N];
3  int main() {
4      int n, m;
5      scanf("%d%s%d%s", &n, p + 1, &m, s + 1);
6      for (int i = 2, j = 0; i <= n; i++) { //生成转移值数组
7          while (j && p[i] != p[j + 1]) j = ne[j];  //跳转
8          if (p[i] == p[j + 1]) j++;  //如果字符串中前后重复
9          ne[i] = j;                  //记录重复点位置
10     }
11     for (int i = 1, j = 0; i <= m; i++) {
12         while (j && s[i] != p[j + 1])
13             j = ne[j];                //如果匹配终端回到转移值得点重新开始匹配
14         if (s[i] == p[j + 1]) j++;    //匹配过程
15         if (j == n) {
16             printf("%d\n", i - n + 1);//输出下标(从 1 开始)
```

```
17              j = ne[j];                  //转移点
18          }
19      }
20      return 0;
21  }
```

2.4.2 EXKMP

</> 代码 2.6: /字符串/EXKMP

```
1   /* 拓展KMP算法
2   nxt[]:x串的每一个后缀与整个串的最长公共前缀，即x[i .. m - 1]与x[0...m - 1]的最长公共前缀
3   extend[]:y的每一个后缀与x的整个串的最长公共前缀，即y[i ... n - 1]与x[0 .. m - 1]的最长公共
        前缀
4   模板：求str2的后缀和str1的前缀的最长公共前缀
5   */
6   const int maxn = 1e6 + 10;
7   int nxt[maxn],extend[maxn];
8   char str1[maxn],str2[maxn];
9   void pre_EKMP(char x[],int m,int next[]){
10      next[0] = m;
11      int j = 0;
12      while(j + 1 < m && x[j] == x[j + 1]) j ++;
13      next[1] = j;
14      int k = 1;
15      for(int i = 2; i < m ; i ++){
16          int p = next[k] + k - 1;
17          int L = next[i - k];
18          if(i + L < p + 1) next[i] = L;
19          else{
20              j = max(0,p - i + 1);
21              while(i + j < m && x[i + j] == x[j]) j ++;
22              next[i] = j;
23              k = i;
24          }
25      }
26  }
27  void EKMP(char x[],int m,char y[],int n,int next[],int extend[]){
28      pre_EKMP(x,m,next);
29      int j = 0;
30      while(j < n && j < m && x[j] == y[j]) j++;
31      extend[0] = j;
32      int k = 0 ;
33      for(int i = 1; i < n ; i++){
34          int p = extend[k] + k - 1;
```

```
        int L = next[i - k];
        if(i + L < p + 1) extend[i] = L;
        else{
            j = max(0,p - i + 1);
            while(i + j < n && j < m && y[i + j] == x[j]) j ++;
            extend[i] = j;
            k = i;
        }
    }
}
int main(){
    while(~scanf("%s%s",str1,str2)){
        int l1 = strlen(str1),l2 = strlen(str2);
        EKMP(str1,l1,str2,l2,nxt,extend);
        int ans = 0;
        for(int i = 0 ; i < l2 ; i ++) if(extend[i] == l2 - i){
             ans = l2 - i;
             break;
        }
        if(ans) for(int i = 0; i < ans; i ++) putchar(str1[i]);
        if(ans) putchar(' ');
        Pri(ans);
    }
    return 0;
}
```

2.5 Manacher

通过构造 + 对称的方式使得寻找最大回文串的复杂度将为 $O(N)$

</> 代码 2.7: /字符串/Manacher

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2e7 + 10;
int chaArr[N], pArr[N];
char s[N];
int maxLcsplength(char str[]) {
    int len = (int)(strlen(str) * 2 + 1); //记录下manacher字符串的长度
    if (len == 0)
        return 0; //空字符串直接返回0
    for (int i = 0, idx = 0; i < len; i++)
        chaArr[i] = (i & 1) == 0 ? '#' : str[idx++]; //将字符串数组manacher化
    // R是最右回文边界，C是R对应的最左回文中心，maxn是记录的最大回文半径
    int R = -1, C = -1, ans = 0;
```

```
    for (int i = 0; i < len; i++) {
        //第一步直接取得可能的最短的回文半径，当i>R时，最短的回文半径是1，反之，最短的回文半径可能是i对应的i'的回文半径或者i到R的距离
        pArr[i] = R > i ? min(R - i, pArr[2 * C - i]) : 1;
        //取最小值后开始从边界暴力匹配，匹配失败就直接退出
        while (i + pArr[i] < len && i - pArr[i] > -1) {
            if (chaArr[i + pArr[i]] == chaArr[i - pArr[i]])
                pArr[i]++;
            else break;
        }//观察此时R和C是否能够更新
        if (i + pArr[i] > R) {
            R = i + pArr[i];
            C = i;
        } //更新最大回文半径的值
        ans = max(ans, pArr[i]);
    }
    return ans - 1;
}
int main() {
    scanf("%s", s);
    printf("%d\n", maxLcsplength(s));
    return 0;
}
```

2.6 Trie 树

2.6.1 字典树

快速插入查找字符串

</> 代码 2.8: /字符串/Trie 树

```
int son[N][26]; // 其中存放的是：子节点对应的idx。其中son数组的第一维是：父节点对应的idx，第第二维计数是：其直接子节点('a' - '0')的值为二维下标。
int cnt [N];    // 以“abc”字符串为例，最后一个字符--- ‘c’ 对应的idx作为cnt数组的下标。数组的值是该idx对应的个数。
int idx = 0;    // 将该字符串分配的一个树结构中，以下标来记录每一个字符的位置。方便之后的插入和查找。
char str[N];
void insert(char *str) {
    int p = 0;
    for (int i = 0; str[i]; i++) {
        int u = str[i] - 'a';
        if (!son[p][u]) son[p][u] = ++ idx;
        p = son[p][u];
    }// 此时的p就是str中最后一个字符对应的trie树的位置idx。
```

```
12      cnt[p]++;//结尾标记
13  }
14  int query(char *str) {
15      int p = 0;
16      for (int i = 0; str[i]; i++) {
17          int u = str[i] - 'a';
18          if (!son[p][u]) return 0;
19          p = son[p][u];
20      }
21      return cnt[p];//如果没有单词在此结尾过依旧会返回0
22  }
23  int main() {
24      int n;
25      scanf("%d", &n);
26      char op[2];
27      while (n--) {
28          scanf("%s%s", op, str);
29          if (op[0] == 'I') insert(str);
30          else printf("%d\n", query(str));
31      }
32      return 0;
33  }
34  //用时间换取空间的动态实现（跑的更慢空间更小）
35  int N,M,K;
36  struct node{
37      node* nxt[26];
38      int val;
39  }*root;
40  char str[100];
41  node* newnode(){
42      node *p = new node();
43      p->val = 0;
44      for(int i = 0 ; i < 26; i ++) p->nxt[i] = NULL;
45      return p;
46  }
47  void insert(char *s) {
48      node* p = root;
49      for(int i = 0; str[i]; i ++){
50          int id = str[i] - 'a';
51          if(p->nxt[id] == NULL) p->nxt[id] = newnode();
52          p = p->nxt[id];
53      }
54      p->val = 1;
55  }
```

```
void query(char *s) {
    node* p = root;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(p->nxt[id] == NULL){
            puts("WRONG");
            return;
        }
        p = p->nxt[id];
    }
    if(p->val == 0) puts("WRONG");
    else if(p->val == 1){
        puts("OK"); p->val = 2;
    }else{
        puts("REPEAT");
    }
}
void deal(node *p){
    for(int i = 0 ; i < 26; i ++) if(p->nxt[i]) deal(p->nxt[i]);
    free(p);
}
```

2.6.2 0-1 字典树

求一个数组中两数的最大异或和, 找出最短且满足异或和 $\geqslant k$ 的区间端点

</> 代码 2.9: /字符串/0-1 字典树

```
const int MAXN = 2e5 + 7, MAXBIT = 31;
struct Trietree{
    int tr[MAXN * 31][2], cnt, tag[MAXN * 31];
    void init() {
        fill(tr[0], tr[0] + MAXN * 2, 0);
        fill(tag, tag + MAXN, 0);
        cnt = 0;
    }
    void insert(int num) {
        int idx = 0;     // idx 与 cnt 的起始下标应当相同
        for(int i = MAXBIT; ~i; i--) {
            int bit = (num >> i) & 1;
            if(!tr[idx][bit])
                tr[idx][bit] = ++ cnt;
            idx = tr[idx][bit];
        }
        tag[idx] = num;
    }
```

```
19      int find_max(int num) {
20          int idx = 0;
21          for(int i = MAXBIT; ~i; i --) {
22              int bit = (num >> i) & 1;
23              if(tr[idx][bit ^ 1]) // 每一位尽量相反，则异或最大
24                  idx = tr[idx][bit ^ 1];
25              else
26                  idx = tr[idx][bit];
27          }
28          return tag[idx];
29      }
30  }Trie;
31  ------------------------------------------------------------
32  #include <iostream>
33  #include <algorithm>
34  using namespace std;
35  const int MAXN = 1e5 + 7, MAXBIT = 31;
36  struct Trietree {
37      int tr[MAXN * MAXBIT][2], cnt, tag[MAXN * MAXBIT];
38      void init() {  //
39          fill(tr[0], tr[0] + MAXN * 2 * MAXBIT, 0);
40          fill(tag, tag + MAXN * MAXBIT, 0);
41          cnt = 0;
42      }
43      void insert(int num, int id) {
44          int idx = 0;
45          for (int i = MAXBIT; ~i; i--) {
46              int bit = (num >> i) & 1;
47              if (!tr[idx][bit])
48                  tr[idx][bit] = ++cnt;
49              idx = tr[idx][bit];
50              tag[idx] = max(tag[idx], id);  // 记录当前节点有值的最右端点
51          }
52      }
53      int find_max(int num, int k) {
54          int idx = 0, res = -1;
55          for (int i = MAXBIT; ~i; i--) {
56              int bit = (num >> i) & 1;
57              if ((k >> i) & 1) {         //如果在这位上为0，则 此位异或起来为 1 才行
58                  idx = tr[idx][bit ^ 1];
59              } else {                    //如果在这位上为0，则 此位异或起来为 1 必定满足条
    件
60                  if (tr[idx][bit ^ 1])
61                      res = max(res, tag[tr[idx][bit ^ 1]]);
```

```
62                  idx = tr[idx][bit];
63              }
64              if (!idx) break;
65          }
66          if (idx) res = max(res, tag[idx]);
67          return res;
68      }
69  } Trie;
70  int a[MAXN], pre[MAXN];
71  int main() {
72      int t;
73      scanf("%d", &t);
74      while (t--) {
75          Trie.init();
76          int n, k;
77          scanf("%d%d", &n, &k);
78          for (int i = 1; i <= n; i++) {
79              scanf("%d", &a[i]);
80              pre[i] = a[i] ^ pre[i - 1];  //异或前缀和
81          }
82          int anl = -1, anr = n + 1;
83          for (int i = 0; i <= n; i++) { // 从 0 开始
84              int now = Trie.find_max(pre[i], k);   // 寻找满图条件的最大的左端点
85              if (now >= 0 && i - now < anr - anl)  // 存在符合条件的端点并且长度更小
86                  anl = now, anr = i;
87              Trie.insert(pre[i], i);  // 插入当前的前缀和
88          }
89          if (anl > -1)
90              printf("%d %d\n", anl + 1, anr);
91          else
92              printf("-1\n");
93      }
94      return 0;
95  }
```

2.7 AC 自动机

$O(N)$ 给定 n 个长度不超过 50 的由小写英文字母组成的单词，以及一篇长为 m 的文章。请问，有多少个单词在文章中出现了。本质 Tire + KMP 算法

</> 代码 2.10: /字符串/AC 自动机

```
1  const int N = 10010, S = 55, M = 1000010;
2  int n, tr[N * S][26], cnt[N * S], idx;
3  char str[M];
```

```
int q[N * S], ne[N * S];

void insert() {
    int p = 0;
    for (int i = 0; str[i]; i++) {
        int t = str[i] - 'a';
        if (!tr[p][t]) tr[p][t] = ++idx;
        p = tr[p][t];
    }
    cnt[p] ++;
}
void build() {
    int hh = 0, tt = -1;
    for (int i = 0; i < 26; i++)
        if (tr[0][i]) q[++tt] = tr[0][i];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = 0; i < 26; i++) {
            int p = tr[t][i];
            if (!p)  tr[t][i] = tr[ne[t]][i];
            else {
                ne[p] = tr[ne[t]][i];
                q[++tt] = p;
            }
        }
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) { //
        scanf("%s", str);
        insert();
    }
    build();
    scanf("%s", str);
    int res = 0;
    for (int i = 0, j = 0; str[i]; i++) {
        int t = str[i] - 'a';
        j = tr[j][t];
        int p = j;
        while (p) {
            res += cnt[p];
            cnt[p] = 0;
```

```
48              p = ne[p];
49          }
50      }
51      printf("%d\n", res);
52      return 0;
53  }
```

2.8 回文树

</> 代码 2.11: /字符串/回文树 p

```
1  /*回文树
2  一个节点表示一个回文串，每个节点之间表示的是本质不同的回文串
3  len[i]表示i结点表示的回文串的长度
4  cnt[i]表示i结点表示的回文串的个数,建树的时候cnt是不完全的，需要调用count()之后才是完全的
5  fail[i]表示结点i失配后跳转的结点，表示结点i表示的回文串的最长后缀回文串
6  next[i][c]表示结点i表示的字符串在两边添加字符c变成的回文串的编号
7  num[i]表示i表示的回文串的最右端点为回文串结尾的回文串个数
8  last指向新添加一个字母后形成的最长回文串标识的结点
9  S[i]表示第i次添加的字符(一开始设S[0] = -1,也可以是之后不会出现的任意字符)
10  */
11 const int maxn = 2e5 + 10;
12 const int maxc = 26;
13 int N,M,K;
14 struct Pal_T{
15     int next[maxn][maxc];
16     int fail[maxn],cnt[maxn],num[maxn],len[maxn],S[maxn];
17     int last,n,tot;
18     int newnode(int l){
19         for(int i = 0 ; i < maxc; i ++) next[tot][i] = 0;
20         cnt[tot] = num[tot] = 0;
21         len[tot] = l;
22         return tot++;
23     }
24     void init(){
25         tot = 0;
26         newnode(0); //偶数根节点
27         newnode(-1); //奇数根节点
28         last = n = 0;
29         S[n] = -1; fail[0] = 1;
30     }
31     int getfail(int x){
32         while(S[n - len[x] - 1] != S[n]) x = fail[x];
33         return x;
```

```
34      }
35      void add(int c){
36          c -= 'a';
37          S[++n] = c;
38          int cur = getfail(last);
39          if(!next[cur][c]){
40              int now = newnode(len[cur] + 2);
41              fail[now] = next[getfail(fail[cur])][c];
42              next[cur][c] = now;
43              num[now] = num[fail[now]] + 1;
44          }
45          last = next[cur][c];
46          cnt[last]++;
47      }
48      void count(){
49          for(int i = tot - 1; i >= 0; i --) cnt[fail[i]] += cnt[i];
50      }
51  }PT,PT2;
```

2.9 Shift-And

</> 代码 2.12: /字符串/Shift-And

```
1   /*Shift-and 算法
2   b[i][j]表示在字符i模式串中第j个位置出现
3   ans[i]表示当前字符串的后缀与模式串的长度为i的前缀匹配
4   每次加入字符的时候;
5   1.将ans左移一位2.ans[0]置1 3.ans与b[str[i]]进行与运算, str[i]为当前加入的字符
6   就更新完成了*/
7   //模板 hdu5972
8   /*给你N位数,接下来有N行,第i行先输入n,表示这个数的第i位上可以在接下来的n个数中挑选,然后i行再
        输n个数。
9   然后输入需要匹配的母串,让你输出母串中有多少个可行的N位子串并输出*/
10  char str[maxn];
11  bitset<1010>b[12],ans;
12  int main(){
13      while(~Sca(N)){
14          for(int i = 0 ; i < 10; i ++) b[i].reset();
15          for(int i = 0; i < N ; i ++){
16              int x; x = read();
17              while(x--) b[read()][i] = 1;
18          }
19          scanf("%s",str); ans.reset();
20          LL sum = 0;
```

```
        for(int i = 0;str[i]; i ++){
            ans <<= 1; ans[0] = 1;
            ans &= b[str[i] - '0'];
            if(ans[N - 1]){
                char tmp = str[i + 1];
                str[i + 1] = '\0';  printf("%s\n",str + (i - N + 1));
                str[i + 1] = tmp;
            }
        }
    }
    return 0;
}
```

2.10 序列自动机

</> 代码 2.13: /字符串/序列自动机

```
//序列自动机
//nxt[i][j]表示i后面第一个j出现的位置(不包括i)
//模板: 给一个长串, 查询N个字符串是否为长串的子序列
//做法: 直接跳next即可
const int maxn = 1e5 + 10;
char str[maxn],s[maxn];
int nxt[maxn][26]; //表示i后面第一个j出现的位置(不包括i)
int main(){
    scanf("%s",str + 1);
    int l = strlen(str + 1);
    for(int i = 0 ; i < 26; i ++) nxt[l][i] = -1; //后面不存在则为-1
    for(int i = l ; i >= 1; i --){   //构造序列自动机
        for(int j = 0; j < 26; j ++) nxt[i - 1][j] = nxt[i][j];
        nxt[i - 1][str[i] - 'a'] = i;
    }
    scanf("%d",&N);
    while(N--){
        scanf("%s",s);
        int now = 0;
        for(int i = 0 ;s[i] && ~now; i ++) now = nxt[now][s[i] - 'a'];
        if(~now) puts("YES");
        else puts("NO");
    }
    return 0;
}
//求子序列个数 f[i]表示以i起始的子序列个数
int dfs(int x) //main函数调用: dfs(0);
```

```
{
    if(f[x]) return f[x];
    for(int i=1;i<=a;i++)
        if(nxt[x][i]) f[x]+=dfs(nxt[x][i]);
    return ++f[x];
}
//求两串的公共子序列个数，两串都构造一下然后跑
LL dfs(LL x,LL y){
    if(f[x][y]) return f[x][y];
    for(LL i=1;i<=a;++i)
        if(nxt1[x][i]&&nxt2[y][i])
            f[x][y]+=Dfs(nxt1[x][i],nxt2[y][i]);
    return ++f[x][y];
}
//求字符串回文子序列的个数：正反都构造一遍然后跑
LL dfs(LL x,LL y){
    if(f[x][y]) return f[x][y];
    for(LL i=1;i<=a;++i)
        if(nxt1[x][i]&&nxt2[y][i]){
            if(nxt1[x][i]+nxt2[y][i]>n+1) continue;
            if(nxt1[x][i]+nxt2[y][i]<n+1) f[x][y]++;
            f[x][y]=(f[x][y]+Dfs(nxt1[x][i],nxt2[y][i]))%mod;
        }
    return ++f[x][y];
}
//求一个A,B的最长公共子序列S，使得C是S的子序列
//dfs(int x,int y,int z)，表示一匹配到C的z位
//需要改变C的构造方法
for(LL i=1;i<=a;++i) nxt[n][i]=n;
for(LL i=0;i<n;++i){
    for(LL j=1;j<=a;++j) nxt[i][j]=i;
    nxt[i][c[i+1]]=i+1;
}
```

2.11 fail 树

</> 代码 2.14: /字符串/fail 树

```
//fail树：将AC自动机上的所有fail指针反向，就形成了一颗fail树
//fail树所有的父节点是子节点的在所有单词中的最长后缀
//所以同一个AC自动机中，串x包含被其他所有串包含的次数就是x尾结点在fail树上的子树大小
//若要求x被y包含的次数，则仅将y到根节点的所有结点val值 ++ ，求x的子树大小即可
//模板 ： 洛谷P3966 求出每个单词在所有单词中出现的次数
const int maxm = 5e5 + 10;
```

```cpp
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
int head[maxm],tot;
struct Edge{
    int to,next;
}edge[maxm];
void init(int cnt){
    for(int i = 0 ; i <= cnt ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int nxt[maxm][26],fail[maxm],cnt[maxm];
int ttt,root;
int newnode(){
    for(int i = 0 ; i < 26; i ++) nxt[ttt][i] = -1;
    cnt[ttt] = 0;
    return ttt++;
}
void INIT(){
    ttt = 0;
    root = newnode();
}
int insert(char *str){
    int p = root;
    for(int i = 0; str[i]; i ++){
        int id = str[i] - 'a';
        if(nxt[p][id] == -1) nxt[p][id] = newnode();
        p = nxt[p][id];
        cnt[p]++;
    }
    return p;
}
void Build(){
    queue<int>Q;
    for(int i = 0; i < 26; i ++){
        if(nxt[root][i] == -1){
            nxt[root][i] = 0;
        }else{
            fail[nxt[root][i]] = root;
```

```
51              Q.push(nxt[root][i]);
52              add(root,nxt[root][i]);
53          }
54      }
55      while(!Q.empty()){
56          int u = Q.front(); Q.pop();
57          for(int i = 0 ; i < 26; i ++){
58              if(nxt[u][i] == -1){
59                  nxt[u][i] = nxt[fail[u]][i];
60              }else{
61                  fail[nxt[u][i]] = nxt[fail[u]][i];
62                  add(nxt[fail[u]][i],nxt[u][i]);
63                  Q.push(nxt[u][i]);
64              }
65          }
66      }
67  }
68  void dfs(int t,int la){
69      for(int i = head[t]; ~i ; i = edge[i].next){
70          int v = edge[i].to;
71          if(v == la) continue;
72          dfs(v,t);
73          cnt[t] += cnt[v];
74      }
75  }
76  int Index[maxn];
77  char str[1000010];
78  int main(){
79      Sca(N); INIT();
80      for(int i = 1; i <= N ; i ++){
81          scanf("%s",str);
82          Index[i] = insert(str);
83      }
84      init(ttt);
85      Build();
86      dfs(root,0);
87      for(int i = 1; i <= N ; i ++){
88          Pri(cnt[Index[i]]);
89      }
90      return 0;
91  }
```

2.12 后缀数组

</> 代码 2.15: /字符串/后缀数组

```
/*  后缀数组
(1)将字符串的所有后缀按照字典序从小到大排序
sa[i]表示排名第i的后缀的下标 ，rak[i]表示下标为i的后缀的排名
(2)Height数组
lcp(x,y)：字符串x与字符串y的最长公共前缀，在这里指x号后缀与与y号后缀的最长公共前缀
height[i]：lcp(sa[i],sa[i-1])，即排名为i的后缀与排名为i-1的后缀的最长公共前缀
H[i]：height[rak[i]]，即i号后缀与它前一名的后缀的最长公共前缀
应用：
1.两个后缀的最大公共前缀：lcp(x,y)=min(heigh[x-y])， 用rmq维护，O(1)查询
2.可重叠最长重复子串：Height数组里的最大值
3.不可重叠最长重复子串 POJ1743：首先二分答案x，对height数组进行分组，保证每一组的minheight
    都>=x
依次枚举每一组，记录下最大和最小长度，多sa[mx]-sa[mi]>=x那么可以更新答案
4.本质不同的子串的数量:枚举每一个后缀，第i个后缀对答案的贡献为len-sa[i]+1-height[i]
*/
#include <iostream>
#include <cstring>
using namespace std;
const int M = 1e6 + 7;
int n, m;
char s[M];
int sa[M], rk[M], height[M], x[M], y[M], cnt[M];
//sa 数组存各排名对应的后缀编号，rk 是各后缀编号的排名
//Height 记录相邻排名的最长公共前缀
// x 是第一关键字， y 是第二关键字， cnt 是计数数组
void get_sa() {
    //初始的基数排序
    for(int i = 1; i <= n; i ++) x[i] = s[i], cnt[x[i]] ++;
    for(int i = 2; i <= m; i++) cnt[i] += cnt[i-1];
    for(int i = n; i ; i--) sa[cnt[x[i]]--] = i;

    for (int k = 1; k <= n; k <<= 1) {
        int num = 0;
        for (int i = n - k + 1; i <= n; i++)
            y[++num] = i; //此处不足补充空格的排在前面
        for (int i = 1; i <= n; i++)
            if (sa[i] > k)
                y[++num] = sa[i] - k;
        //前k位被不足的子串占据，所以第二关键字后面排名的是第一关键字排序位置-k
        for (int i = 1; i <= m; i++) cnt[i] = 0;
        for (int i = 1; i <= n; i++) cnt[x[i]]++;
        for (int i = 2; i <= m; i++) cnt[i] += cnt[i - 1];
        for (int i = n; i; i--) sa[cnt[x[y[i]]]--] = y[i], y[i] = 0;
```

```
43
44          swap(x, y);
45          x[sa[1]] = 1, num = 1;
46          for (int i = 2; i <= n; i++) {
47              bool due = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] + k]);
48              //新的相邻排名，其位置的老排名相同，那它的新排名其实是并列的
49              if (!due) num++; //只有不并列，排名才会向前走
50              x[sa[i]] = num; //排名i的位置，其连续的排名是num，并列的相同
51          }
52          if (num == n) break;
53          m = num;
54      }
55  }
56  void get_height() {
57      for (int i = 1; i <= n; i++) rk[sa[i]] = i;
58      for (int i = 1, k = 0; i <= n; i++) {
59          if (rk[i] == 1) continue; //排名第1的不需计算，
60          if (k) k--;               //去掉前面已经比较的一个字符
61          int j = sa[rk[i] - 1]; //这个位置也许与前面比较的j位置不同，
62                                 //但是去掉前面一个字符后，相同部分不会少
63          while (i + k <= n && s[i + k] == s[j + k]) k ++;//从k位置开始比较
64          height[rk[i]] = k; //排名rk[i]的height
65      }
66  }
67  int main() {
68      scanf("%s", s + 1);
69      n = strlen(s + 1), m = 'z';
70      get_sa();
71      get_height();
72      for(int i = 1; i <= n; i ++)
73          printf("%d%c", sa[i], i == n ? '\n' : ' ');
74      for(int i = 1; i <= n; i ++)
75          printf("%d%c", height[i], i == n ? '\n' : ' ');
76      return 0;
77  }
```

2.13 后缀自动机

</> 代码 2.16: /字符串/后缀自动机

```
1  /* SAM 后缀自动机
2  时空复杂度O(n)
3  1.每个状态对应一个endpos的等价类，表示以该点结尾的子串的集合。
4  2.t0到任意节点之间的转移连接起来就是字符串的子串，任意子串对应一条t0出发的路径
```

```
3.通常一个结点表示的字符串是这个等价类集合中长度最长的字符串，集合中的所有其他字符串都是他的后缀
性质1.考虑一个endpos等价类，将类中的所有子串按长度非递增的顺序排序。每个子串都不会比它前一个子串长,与此同时每个子串也是它前一个子串的后缀。换句话说，对于同一等价类的任一两子串，较短者为较长者的后缀，且该等价类中的子串长度恰好覆盖整个区间[x,y] 。

后缀链接(parent树)
1.结点指向结点表示的最短字符串的去掉第一个字符的字符串
例如结点最短字符串表示abcabc，则指向结点的最长字符串bcabc
2.叶子节点都是主链上的节点，主链上的点不一定是叶子结点
3.在主链上的点，最长的子串都是原串的前缀
4.一个节点上的子串出现次数是一样的
5.对应同一状态v的所有子串在文本串T中的出现次数相同
6.parents上父亲上的子串出现次数，是儿子上的子串出现次数之和，如果父亲在主链上，就再加一
7.点i上面表示子串的数量为len[fa[i]]-len[i]。
8.两节点的最长公共后缀是他们在parent树上的LCA点表示的最长字符串
可以证明后缀自动机上至多有2 * n - 1个结点和3 * n - 4条边
*/
//tag:以下模板不自带初始化，son,fa,len,num等数组均未初始化，多样例请特别注意
/*模板1: 求字符串中出现次数不为1的子串中出现次数 * 长度最大的值
num[i]为当前状态点出现次数，len[i]为当前状态点最大字符串长度
根据parent树的性质6，主链上的结点肯定只出现一次，所以dfs之后统计一下儿子的和就可以知道每个结点出现的次数
采用桶排序模拟dfs的过程可以不必实质建出parent树*/
//tag:如果需要动态维护num，每次新加点之后对parent树的每个祖先都+1,同时创建nq的时候需要复制q的num值，下面有给出
int len[maxn << 1],fa[maxn << 1],son[maxn << 1][maxc];
LL num[maxn << 1];
int size,last;
void Init(){
    size = last = 1;
}
void insert(char c){
    int s = c - 'a';
    int p = last,np = ++size;last = np; num[np] = 1;
    //cout << np << endl;
    len[np] = len[p] + 1;
    for(;p && !son[p][s]; p = fa[p]) son[p][s] = np;
    if(!p) fa[np] = 1;
    else{
        int q = son[p][s];
        if(len[p] + 1 == len[q]) fa[np] = q;
        else{
            int nq = ++size; len[nq] = len[p] + 1;
```

```
            memcpy(son[nq],son[q],sizeof(son[q]));
            fa[nq] = fa[q]; fa[q] = fa[np] = nq;
            //num[nq] = num[q] if 需要动态维护num
            for(;son[p][s] == q && p;p = fa[p]) son[p][s] = nq;
        }
    }
}
void insert(char *s){
    Init();
    for(int i = 0; s[i] ; i ++) insert(s[i]);
}
char str[maxn];
int A[maxn << 1],tmp[maxn << 1];
void Qsort(){
    //排出一个按照len从小到大的结点序列
    //本质上是模拟dfs的加,len小的一定在len大的之前遍历
    for(int i = 1; i <= size; i ++) tmp[len[i]]++;
    for(int i = 1; i <= size; i ++) tmp[i] += tmp[i - 1];
    for(int i = 1; i <= size; i ++) A[tmp[len[i]]--] = i;
}
int main(){
    scanf("%s",str);
    insert(str);
    Qsort();
    for(int i = size; i >= 1; i --) num[fa[A[i]]] += num[A[i]];
    LL ans = 0;
    for(int i = 2; i <= size; i ++) if(num[i] != 1)ans = max(ans,len[i] * num[i]);
    Prl(ans);
    return 0;
}

/*应用，输出长度为1-n的子串出现最多次数的子串的次数
做法:利用模板1求出每个节点的出现次数，然后每个节点i将len[min]-len[max]的答案更新num[i]
模板:在模板1的基础上添加如下代码*/
for(int i = 2; i <= size; i ++) Max[len[i]] = max(Max[len[i]],num[i]);
for(int i = N; i >= 1 ; i --) Max[i] = max(Max[i],Max[i + 1]);
for(int i = 1; i <= N ; i ++) Prl(Max[i]);

/*应用：求s,t两串的最长公共子串(On)
做法：对S建SAM然后t开始跑,跑到失配点i就跳转到fa[i]*/
scanf("%s%s",s,t);
insert(s);
int ans = 0,tmp = 0;
int p = 1;
```

```
for(int i = 0;t[i]; i ++){
    int v = t[i] - 'a';
    while(p != 1 && !son[p][v]) p = fa[p],tmp = len[p];
    if(son[p][v]){
        p = son[p][v]; tmp++;
    }
    ans = max(ans,tmp);
}
Pri(ans);
/*应用：多个字符串的最长公共子串
做法：选取一个字符串建SAM，其它字符串如上跑一遍
每个状态对应每个字符串的答案为s1,s2,s3..sn,则每个状态的最终答案为min(s1,s2..sn),取最大即可
模板：dp[i]表示当前结尾的字符匹配上之后的最长匹配长度，显然一个结点匹配上之后他的父亲也匹配上
所以将parent树从叶子到跟再更行一遍*/
int dp[maxn << 1],Max[maxn << 1];
int A[maxn << 1],tmp[maxn << 1];
void ins(char* str){
    for(int i = 0 ; i <= size; i ++) dp[i] = 0;
    int l = 0,p = 1;
    for(int i = 0;str[i]; i ++){
        int v = str[i] - 'a';
        while(p != 1 && !son[p][v]) p = fa[p],l = len[p];
        if(son[p][v]){
            p = son[p][v]; l++;
            dp[p] = max(dp[p],l);
        }
    }
    for(int i = size; i >= 2; i --){
        dp[fa[A[i]]] = max(dp[fa[A[i]]],min(len[fa[A[i]]],dp[A[i]]));
    }
    for(int i = 1; i <= size; i ++) Max[i] = min(Max[i],dp[i]);
}
int main(){
    scanf("%s",s); int cnt = 1;
    while(~scanf("%s",t[cnt])) cnt++;
    cnt--;
    insert(s);
    for(int i = 1; i <= size; i ++) tmp[len[i]]++;
    for(int i = 1; i <= size; i ++) tmp[i] += tmp[i - 1];
    for(int i = 1; i <= size; i ++) A[tmp[len[i]]--] = i;
    for(int i = 1; i <= size; i ++) Max[i] = len[i];
    for(int i = 1; i <= cnt ; i ++) ins(t[i]);
    int ans = 0;
```

```
    for(int i = 2; i <= size; i ++) ans = max(ans,Max[i]);
    Pri(ans);
}
/*最小表示法，长度为N的循环字符串中找到字典序最小的字符串
模板:将原串倍增一遍之后SAM直接寻找长度为N的字典序最小的路径
数值大小不定，采用map建图，用时间换空间
*/
const int maxn = 6e5 + 10;
int N,M,K;
int len[maxn << 1],fa[maxn << 1];
map<int,int>son[maxn << 1];
int size,last;
void Init(){
    size = last = 1;
}
inline void insert(int c){
    int p = last,np = ++size; last = np;
    len[np] = len[p] + 1;
    for(;p && !son[p].count(c); p = fa[p]) son[p][c] = np;
    if(!p) fa[np] = 1;
    else{
        int q = son[p][c];
        if(len[p] + 1 == len[q]) fa[np] = q;
        else{
            int nq = ++size; len[nq] = len[p] + 1;
            son[nq] = son[q];
            fa[nq] = fa[q]; fa[q] = fa[np] = nq;
            for(;son[p][c] == q && p; p = fa[p]) son[p][c] = nq;
        }
    }
}
int a[maxn];
int main(){
    Sca(N); Init();
    for(int i = 1; i <= N ; i ++) insert(a[i] = read());
    for(int i = 1; i <= N ; i ++) insert(a[i]);
    int t = 1;
    for(int i = 1; i <= N ; i ++){
        printf("%d ",son[t].begin()->first);
        t = son[t].begin()->second;
    }
    return 0;
}
/*应用:检查字符串是否出现在文本中
```

```
175 做法：对文本建后缀自动机，能跑完模式串且不出现NULL结点就是出现在文本中了*/
176
177 /*应用:不同子串个数
178 做法：相当于图上从原点起有多少不同的路径，因为这是一个DAG图，所以直接DP一下就可以了
179 dp[i]表示从i点出发的路径数量
180 初始dp[i] = 1,然后从终点向起点累加，即dp[i] = 1 + Σdp[j](j为i的儿子)
181 最终不同子串的个数就是dp[1] - 1,因为需要减掉一个空串*/
182
183 /*模板：求第k小子串，选项0为不同位置的相同子串算一个，1为不同位置的相同子串算不同
184 做法：选项0:如同上述求不同字串个数的做法求出每个节点出发的路径数量，然后就如权值线段树找k大一
        般找第k大的路径
185 选项1:利用模板1求出不同结点表示的串的个数num[i],dp[i]的初始化从1变为num[i],即从i到i的路径变
        为了num[i]条,然后同上
186 */
187 const int maxn = 5e5 + 10;
188 char str[maxn];
189 int tmp[maxn << 1],A[maxn << 1];
190 int sum[maxn << 1];
191 void dfs(int t,int k){  //找出t出发的第k大路径
192     if(k <= num[t]) return;
193     k -= num[t];   //减去当前节点表示的路径数
194     for(int i = 0 ; i < 26; i ++){
195         if(!son[t][i]) continue;
196         int v = son[t][i];
197         if(sum[v] >= k){
198             printf("%c",i + 'a');
199             dfs(v,k);
200             return;
201         }
202         k -= sum[v];
203     }
204 }
205 int main(){
206     scanf("%s",str);
207     insert(str);
208     for(int i = 1; i <= size; i ++) tmp[len[i]]++;
209     for(int i = 1; i <= size; i ++) tmp[i] += tmp[i - 1];
210     for(int i = 1; i <= size; i ++) A[tmp[len[i]]--] = i;
211     for(int i = size; i >= 1; i --) num[fa[A[i]]] += num[A[i]];
212     int op = read(),k = read();
213     for(int i = 1; i <= size; i ++) sum[i] = op?num[i]:num[i] = 1;
214     sum[1] = num[1] = 0;
215     for(int i = size; i >= 1; i --){
216         for(int j = 0 ; j < 26; j ++){
```

```
            if(son[A[i]][j]) sum[A[i]] += sum[son[A[i]][j]];
        }
    }
    if(sum[1] < k) puts("-1");
    else dfs(1,k);
    return 0;
}
/*求区间不同字串个数
做法:一个很关键的性质: SAM每次插入字符后增加的不同字串的个数为len[last] - len[fa[last]]
所以对于每个左端点跑一边SAM, 然后插入的时候记录答案, 即可获得dp[i][j]表示区间不同字符串的个数*/
int dp[maxn][maxn];
int main(){
    int T = read();
    while(T--){
        scanf("%s",str + 1);
        for(int i = 1; str[i]; i ++){
            for(int i = 0; i <= size; i ++){
                fa[i] = len[i] = 0;
                for(int j = 0 ; j < 26; j ++) son[i][j] = 0;
            }
            Init();
            for(int j = i ; str[j] ; j ++){
                insert(str[j]);
                dp[i][j] = dp[i][j - 1] + len[last] - len[fa[last]];
            }
        }
        Sca(M);
        while(M--){
            int l = read(),r = read();
            Pri(dp[l][r]);
        }
    }
    return 0;
}
```

第三章　计算几何

3.1　基础准备

</> 代码 3.1:　/计算几何/基础准备

```
const double eps = 1e-8;
const double PI = acos(-1);
// 符号函数 (0 , -1, 1)
int sign(double x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;}
// 比较函数(== 0, < -1, > 1)
int cmp(double x, double y) { return fabs(x - y) < eps ? 0 : x < y ? -1 : 1;}
// 弧度转角度 与 角度转弧度
double rad_to_deg (double x) { return 180 / PI * x;}
double deg_to_rad (double x) { return PI / 180 * x;}
// floor(x) 向下取整函数 ceil(x) 向上取整函数 round(x) 四舍五入函数
const Point O = {0, 0};                          // 原点
const Line Ox = {O, {1, 0}}, Oy = {O, {0, 1}}; // 坐标轴
const double PI = acos(-1), eps = 1e-9;
```

3.2　点与向量

</> 代码 3.2:　/计算几何/点与向量

```
struct Point {     // 点
    double x, y;
    Point(double _x = 0.0, double _y = 0.0) : x(_x), y(_y) {}
    Point operator- (const Point& t) {return Point(x - t.x, y - t.y);}
    Point operator+ (const Point& t) {return Point(x + t.x, y + t.y);}
    Point operator* (double k) { return Point(k * x, k * y); } // 数乘
    Point operator/ (double k) { return Point(x / k, y / k); }
    double operator* (const Point& t) { return x * t.x + y * t.y; }// 点乘
    double operator^ (const Point& t) { return x * t.y - y * t.x; }// 叉乘
    Point r90a() { return Point(-y, x); } // 逆时针旋转90度
    Point r90c() { return Point(y, -x); } // 顺时针旋转90度
    Point rotate(double agl) {return Point(x * cos(agl) + y * sin(agl), -x * sin(agl) +
    y * cos(agl));} //弧度
```

```
    Point rotate(double agl, Point C) { return C + (*this - C).rotate(agl); }
                // DEPENDS ^1
    double len() { return sqrt(x * x + y * y); }
    double slope() { return y / x; } // 斜率,判不存在用paral_y
    Point norm() {return Point(x / this->len(), y / this->len());}
    Point pnorm() {return Point(x < 0 ? -1 : 1) / this->len()*(*this);}
    bool operator < ( const Point& t){ return x == t.x ? y < t.y : x < t.x; }
    bool operator==(const Point& t) { return cmp(x, t.x) == 0 && cmp(y, t.y) == 0;}
};
using Vec = Point; // 向量

double get_angle(Point a, Point b){ return acos(a * b / a.len() / b.len());} //求夹角(余弦弧度)
double area(Point a, Point b, Point c){return (b - a) ^ (c - a);} //求四边形面积(a为交点)
// 判断两向量的位置关系 (逆时针 1, 顺时针 -1, 重合 (相反) 0)
int cmp_order(Point a, Point b) {double res = a ^ b;return sign(res);}
// 判断两向量夹角大小 (锐角 1, 钝角 -1, 直角 0, 重合 2, 相反 -2)
int cmp_angle(Point a, Point b) {
    double res = a * b, res2 = a.len() * b.len();
    if (res == -res2) return -2;
    if (res == res2) return 2;
    return sign(res);
}
bool paral_x(Vec v) { return cmp(v.y, 0) == 0; }          // 向量是否与x轴平行
bool paral_y(Vec v) { return cmp(v.x, 0) == 0; }          // 向量是否与y轴平行
```

3.3 直线、线段与圆

</> 代码 3.3: /计算几何/直线、线段与圆

```
struct Line {    // 直线 (点向式)
    Point p; Vec v;
    Line (Point a, Point b) { p = a, v = b - a; }                 // 点向式
    Line (double k, double b) { p = Point(0, b), v = Vec(1, k); } // 斜截式
    Line (Point P, double k) { p = P, v = Vec(1, k); }            // 点斜式直线
    // Line (Seg l) { return {l.a, l.b - l.a}; }                  // 线段所在直线
    Line rotate(Line l, double rad, Point C = Point(0, 0)) { return Line(l.p.rotate(rad, C), l.v.rotate(rad));}
    Line operator+(Vec t) { return {p + t, v}; }          // 位移一个向量的距离
    void toString() {cout << "直线基点坐标为 " << p.x << " " << p.y << "  方向 : " << v.x << " " <<  v.y << endl;}
};
struct Seg {       // 线段 (存两个端点)
    Point a, b;
```

```
13      Seg(Point a, Point b) : a(a), b(b) {}
14      Seg rotate(Seg l, double rad, Point C = Point(0, 0)) { return Seg(l.a.rotate(rad, C)
        , l.b.rotate(rad, C));}
15      Seg operator+(Vec t) { return {a + t, b + t}; }      // 位移一个向量的距离
16      bool operator==(const Seg& t) { return (a == t.a && b == t.b) || (a == t.b && b == t
        .a); }
17      void toString() {cout << "两端点坐标为 :  " << a.x << " " << a.y << " ;  " << b.x <<
        " " <<  b.y << endl;}
18  };
19  struct Circle {    // 圆 (存圆心和半径)
20      Point O; double r;
21      Circle(Point _O, double _r) : O(_O), r(_r) {}
22  };
23  //给定直线的横坐标求纵坐标  // NOTE 请确保直线不与y轴平行
24  double at_x(Line l, double x) { return l.p.y + (x - l.p.x) * l.v.y / l.v.x; }
25  // 给定直线的纵坐标求横坐标// NOTE 请确保直线不与x轴平行
26  double at_y(Line l, double y) { return l.p.x - (l.p.y - y) * l.v.x / l.v.y; }
27  // 点到直线的垂足
28  Point pedal(Point P, Line l) { return l.p - l.v * ((l.p - P) * l.v / (l.v * l.v)); }
29  // 过某点作直线的垂线
30  Line perp(Line l, Point P) { return {P, l.v.r90c()}; }
31  // 角平分线
32  Line bisec(Point P, Vec u, Vec v) { return {P, u.norm() + v.norm()}; }
33  // 线段中点
34  Point midp(Seg l) { return Point((l.a.x + l.b.x) / 2, (l.a.y + l.b.y) / 2); }
35  // 线段中垂线
36  Line perp(Seg l) { return {midp(l), (l.b - l.a).r90c()}; }
37  // 点是否在直线上
38  bool on(Point P, Line l) { return cmp((P.x - l.p.x) * l.v.y, (P.y - l.p.y) * l.v.x) ==
        0; }
39  // 点是否在线段上
40  bool on(Point P, Seg l) { return cmp((P - l.a).len() + (P - l.b).len(), (l.a - l.b).len
        ()) == 0; }
41  // 两条直线是否重合
42  bool operator==(Line a, Line b) { return on(a.p, b) && on(a.p + a.v, b); }
43  // 直线与圆是否相切
44  bool tangency(Line l, Circle C) { return cmp(abs((C.O ^ l.v) - (l.p ^ l.v)), C.r * l.v.
        len()) == 0; }
45  // 圆与圆是否相切
46  bool tangency(Circle C1, Circle C2) { return cmp((C1.O - C2.O).len(), C1.r + C2.r) == 0;
         }
47  // 两点间的距离
48  double dis(Point A, Point B) { return (A - B).len(); }
49  // 点到直线的距离
```

```cpp
double dis(Point P, Line l) { return abs((P ^ l.v) - (l.p ^ l.v)) / l.v.len(); }
//点P到线段AB距离
double dis_PtoSeg(Point p, Seg s){
    if(s.a == s.b) return (p - s.a).len();//AB重合
    Vec x = p - s.a, y = p - s.b, z = s.b - s.a;
    if(cmp(x * z, 0) == -1)return x.len();//P距离A更近
    if(cmp(y * z, 0) == -1)return y.len();//P距离B更近
    return abs( (x ^ z) / z.len());//面积除以底边长
}
// 平行直线间的距离(请确保两直线平行)
double dis(Line a, Line b) { return abs((a.p ^ a.v.pnorm()) - (b.p ^ b.v.pnorm())); }
//判断两线段是否相交
bool seg_intersection(Point a1, Point a2, Point b1, Point b2){
    double c1 = (a2 - a1) ^ (b1 - a1), c2 = (a2 - a1) ^ (b2 - a1);
    double c3 = (b2 - b1) ^ (a2 - b1), c4 = (b2 - b1) ^ (a1 - b1);
    return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;//不算端点
}
bool seg_intersection(Seg s1, Seg s2) {
    double c1 = (s1.b - s1.a) ^ (s2.a - s1.a), c2 = (s1.b - s1.a) ^ (s2.b - s1.a);
    double c3 = (s2.b - s2.a) ^ (s1.b - s2.a), c4 = (s2.b - s2.a) ^ (s1.a - s2.a);
    return sign(c1) * sign(c2) < 0 && sign(c3) * sign(c4) < 0;// 算端点
}
```

3.4 对称与交点

</> 代码 3.4: /计算几何/对称与交点

```cpp
// 对称，点和向量此处不同
// 关于点对称
Point reflect(Point A, Point P) { return {P.x * 2 - A.x, P.y * 2 - A.y}; }
Line reflect(Line l, Point P) { return {reflect(l.p, P), l.v}; }
Seg reflect(Seg l, Point P) { return {reflect(l.a, P), reflect(l.b, P)}; }
// 关于直线对称
// NOTE 向量和点在这里的表现不同，求向量关于某直线的对称向量需要用reflect_v
Point reflect(Point A, Line ax) { return reflect(A, pedal(A, ax)); }
Vec reflect_v(Vec v, Line ax) { return reflect(v, ax) - reflect(Point(0, 0), ax); }
Line reflect(Line l, Line ax) { return {reflect(l.p, ax), reflect_v(l.v, ax)}; }
Seg reflect(Seg l, Line ax) { return {reflect(l.a, ax), reflect(l.b, ax)}; }
// 交点
// 直线与直线交点
vector<Point> inter(Line a, Line b) {
    double c = a.v ^ b.v;
    if (cmp(c, 0) == 0) return {};
    Vec v = Vec{a.p ^ (a.p + a.v), b.p ^ (b.p + b.v)} * 1 / c;
```

```
    return {{v * Vec{-b.v.x, a.v.x}, v * Vec{-b.v.y, a.v.y}}};
}
// 直线与圆交点
vector<Point> inter(Line l, Circle C) {
    Point P = pedal(C.O, l);
    double h = (P - C.O).len();
    if (cmp(h, C.r) > 0) return {};
    if (cmp(h, C.r) == 0) return {P};
    double d = sqrt(C.r * C.r - h * h);
    Vec vec = l.v * (d / l.v.len());
    return {P + vec, P - vec};
}
// 圆与圆的交点
vector<Point> inter(Circle C1, Circle C2) {
    Vec v1 = C2.O - C1.O, v2 = v1.r90c();
    double d = v1.len();
    if (cmp(d, C1.r + C2.r) > 0 || cmp(abs(C1.r - C2.r), d) > 0) return {};
    if (cmp(d, C1.r + C2.r) == 0 || cmp(d, abs(C1.r - C2.r)) == 0) return {C1.O + v1 * (
    C1.r / d)};
    double a = ((C1.r * C1.r - C2.r * C2.r) / d + d) / 2;
    double h = sqrt(C1.r * C1.r - a * a);
    Vec av = v1 * (a / v1.len()), hv = v2 * (h / v2.len());
    return {C1.O + av + hv, C1.O + av - hv};
}
```

3.5 三角形

三角形面积

1. 两条边求叉积除二后的绝对值;
2. 海伦公式:

$$S = \sqrt{p(p-a)(p-b)(p-c)},\ p = \frac{a+b+c}{2}$$

$$S = \frac{ab\sin c}{2}$$

</> 代码 3.5: /计算几何/三角形

```
// 三角形的重心
Point barycenter(Point A, Point B, Point C){
    return {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3};
}
// 三角形的外心
// DEPENDS r90c, V*V, d*V, V-V, V+V
// NOTE 给定圆上三点求圆, 要先判断是否三点共线
Point circumcenter(Point A, Point B, Point C) {
    double a = A * A, b = B * B, c = C * C;
```

```
    double d = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y));
    return ((B - C) * a + (C - A) * b + (A - B) * c).r90c() * (1 / d);
}
// 三角形的内心
// DEPENDS len, d*V, V-V, V+V
Point incenter(Point A, Point B, Point C) {
    double a = (B - C).len(), b = (A - C).len(), c = (A - B).len();
    double d = a + b + c;
    return (A * a + B * b + C * c) * (1 / d);
}
// 三角形的垂心
// DEPENDS V*V, d*V, V-V, V^V, r90c
Point orthocenter(Point A, Point B, Point C) {
    double n = B * (A - C), m = A * (B - C);
    double d = (B - C) ^ (A - C);
    return ((C - B) * n - (C - A) * m).r90c() * (1 / d);
}
```

3.6 小技巧

</> 代码 3.6: /计算几何/小技巧

```
//给三点坐标求第四点坐标
//求矩形的第四个点的函数 //已知A(a.x,a.y),B(b.x,b.y),C(c.x,c.y), 求D(dx,dy)
double sqr(int x) {return x * x;}
Point get_4th(Point a, Point b, Point c) {
//ab表示AB^2,ac表示AC^2,BC表示BC^2
    int ab = sqr( a.x - b.x ) + sqr( a.y - b.y ),
        ac = sqr( a.x - c.x ) + sqr( a.y - c.y ),
        bc = sqr( b.x - c.x ) + sqr( b.y - c.y );
    int dx, dy;
//用勾股定理的逆定理, 判断谁是直角边,再根据矩形对边平行的性质, 算出第四个点的坐标
    if (ab + ac == bc) dx = b.x + c.x - a.x, dy = b.y + c.y - a.y;
    if (ab + bc == ac) dx = a.x + c.x - b.x, dy = a.y + c.y - b.y;
    if (ac + bc == ab) dx = a.x + b.x - c.x, dy = a.y + b.y - c.y;
    return Point(dx, dy);
}
//两圆相交面积
double AreaOfOverlap(Point c1, double r1, Point c2, double r2){
    double d = (c1 - c2).len();
    if(r1 + r2 < d + eps) return 0.0;
    if(d < fabs(r1 - r2) + eps){
        double r = min(r1, r2);
        return PI * r * r;
```

```
    }
    double x = (d * d + r1 * r1 - r2 * r2) / (2.0 * d);
    double p = (r1 + r2 + d) / 2.0;
    double t1 = acos(x / r1);
    double t2 = acos((d - x) / r2);
    double s1 = r1 * r1 * t1;
    double s2 = r2 * r2 * t2;
    double s3 = 2 * sqrt(p * (p - r1) * (p - r2) * (p - d));
    return s1 + s2 - s3;
}
//三点确定外接圆圆心坐标
Point Excenter(Point a, Point b, Point c){
    double a1 = b.x - a.x, b1 = b.y - a.y;
    double c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y;
    double c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    return Point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
}
```

3.7 极角排序

</> 代码 3.7: /计算几何/极角排序

```
#include<bits/stdc++.h>
using namespace std;
typedef long double ld;
const int M = 1e5 + 7;
struct Point {
    ld x, y;
    int id;
    Point(const ld &x = 0, const ld &y = 0) :x(x), y(y){}
    Point operator-(const Point& b){return Point(x - b.x, y - b.y);}
    ld get_length() {return sqrt(x * x + y * y);}
}p[M];
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++) {
        int x, y;
        scanf("%d%d", &x, &y);
        p[i] = Point(x, y);
    }

```

```
21      sort(p + 1, p + n + 1, [&](auto p1, auto p2) {
22          return atan2(p1.y, p1.x) < atan2(p2.y, p2.x);
23      });
24      for (int i = 1; i <= n; ++i)cout << '(' << p[i].x << ',' << p[i].y << ',' << atan2(p
        [i].y, p[i].x) <<')' << endl;
25      return 0;
26  }
27  ld cross(Point a, Point b) {
28      return a.x * b.y - a.y * b.x;
29  }
30  int qua(Point p) { return (p.y < 0) << 1 | (p.x < 0) ^ (p.y < 0);}    // 求象限(0, 1, 2,
        3)
31  bool cmp(Point a, Point b) { // cmp 写法
32      return qua(a) < qua(b) || qua(a) == qua(b) && (cross(a, b) > 0.0);
33  }
34  bool cmp2(const Point &a, const Point &b)//先按象限排序，再按极角排序，再按远近排序
35  {
36      if (a.y == 0 && b.y == 0 && a.x*b.x <= 0)return a.x>b.x;
37      if (a.y == 0 && a.x >= 0 && b.y != 0)return true;
38      if (b.y == 0 && b.x >= 0 && a.y != 0)return false;
39      if (b.y*a.y <= 0)return a.y>b.y;
40      return cross(a,b) > 0 || (cross(a,b) == 0 && a.y > b.y);
41  }
42  int main() {
43      int n;
44      scanf("%d", &n);
45      for(int i = 1; i <= n; i ++) {
46          int x, y;
47          scanf("%d%d", &x, &y);
48          p[i] = Point(x, y);
49      }
50      Point c = Point(0, 0); // 原点（可更改）
51      sort(p + 1, p + n + 1, [&](auto v1, auto v2) {
52          return qua(v1 - c) < qua(v2 - c) || qua(v1 - c) == qua(v2 - c) && (cross(v1 - c,
         v2 - c) > 0.0);
53      });
54      for (int i = 1; i <= n; ++i)cout << '(' << p[i].x << ',' << p[i].y <<')' << endl;
55      return 0;
56  }
```

3.8 多边形

任意凸多边形的外角和均为 $360°$

任意 n 凸多边形内角和为 $(n-2)180°$

3.8.1 点在多边形内

</> 代码 3.8: /计算几何/点在多边形内

```
//winding number 算法判断点是否在多边形内，若点在多边形(内 ：1，外 ： 0，上 ： -1)
int isPointInPolygon(Point p, vector<Point> poly){
    int wn = 0;
    int n = poly.size();
    for(int i = 0; i < n; ++i){
        if(on(p, Seg(poly[i], poly[(i+1)%n]))) return -1;
        int k = sign((poly[(i+1)%n] - poly[i]) ^ (p - poly[i]));
        int d1 = sign(poly[i].y - p.y), d2 = sign(poly[(i+1)%n].y - p.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn ++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn --;
    }
    if(wn != 0) return 1;
    return 0;
}
bool judge(Point a,Point L,Point R){//判断AL是否在AR右边
    return sign((L-a) ^ (R-a)) > 0;//必须严格以内
}
//二分判断点是否在凸多边形内，若点在多边形(内 ：1，外 ： 0，上 ： -1)
int isPointInConvexPolygon(Point a, vector<Point> poly){
    int n = poly.size();//点按逆时针给出(一定要满足这个条件)
    if(judge(poly[0], a, poly[1]) || judge(poly[0], poly[n - 1], a))return 0;//在P[0_1]
    或P[0_n - 1]外
    if(on(a, Seg(poly[0], poly[1])) || on(a, Seg(poly[0], poly[n - 1])))return -1;//在P
    [0_1]或P[0_n-1]上
    int l = 1,r = n - 2;
    while(l < r){//二分找到一个位置pos使得P[0]_A在P[1_pos],P[1_(pos+1)]之间
        int mid = (l + r) >> 1;
        if(judge(poly[1], poly[mid], a))l = mid;
        else r = mid - 1;
    }
    if(judge(poly[l], a, poly[l + 1]))return 0;//在P[pos_(pos+1)]外
    if(on(a, Seg(poly[l], poly[l + 1])))return -1;//在P[pos_(pos+1)]上
    return 1;
}
//【判断多边形A与多边形B是否相离】
int judge_PP(vector<Point> A, vector<Point> B){
    for(int i = 0; i < A.size(); ++ i) {
        for(int j = 0; j < B.size(); ++ j) {
            if(seg_intersection(Seg(A[i], A[(i + 1) % A.size()]), Seg(B[j], B[(j + 1) %
    B.size()])))return 0;//两线段相交
            if(isPointInConvexPolygon(A[i], B) || isPointInConvexPolygon(B[j], A))return
```

```
        0; //点包含在内
39          }
40      }
41      return 1;
42  }
```

若这个多边形为凸多边形，只需要判断点是否在所有边的左边即可（按逆时针顺序排列的顶点集）
textbf 判断线段是否在任意多边形内
1. 选段与多边形不相交
2. 两端点都在多边形内

3.8.2 多边形面积

</> 代码 3.9: /计算几何/多边形面积

```
1  double PolyArea(Point p[], int n) {
2      double res = 0.0;
3      for(int i = 0 ; i < n; i ++)
4          res += p[i] ^ p[(i + 1) % n];
5      return res / 2;
6  }
7  double PolyArea(Point p[], int n) {
8      double res = 0.0;
9      for(int i = 0 ; i < n - 1; i ++)
10         res += (p[i] - p[0]) ^ (p[(i + 1) % n] - p[i]);
11     return res / 2;
12 }
```

3.9 Pick 定理

计算顶点都在格点上的多边形的面积公式, a 表示多边形内的点数，b 表示多边形边界上的点数，S 表示多边形面积。

$$2S = 2a + b - 2,\ S = a + \frac{b}{a} - 1$$

给多边形顶点求内部有多少点

$$a = S - \frac{b}{2} + 1$$

3.10 凸包

3.10.1 二维凸包

</> 代码 3.10: /计算几何/二维凸包

```
1  const int M = 1e5 + 7;
2  int sta[M], n; // 单调栈
3  bool st[M];    // 标记此点是否被选取
```

```cpp
struct Point {
    double x, y;
    Point(double x = 0.0, double y = 0.0) : x(x), y(y) {}
    Point operator-(Point b){return Point(x - b.x, y - b.y);}
    bool operator<(Point b){return x < b.x;}
}p[M];
double cross(Point a, Point b) {return a.x * b.y - a.y * b.x;}
double distanc(Point a, Point b){
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
double solve(){
    int hh = 0; // 上凸壳
    for(int i = 1; i <= n; i ++) {
        // 通过叉积判断两个线段之间的位置关系
        while(hh >= 2 && cross(p[sta[hh]] - p[sta[hh - 1]], p[i] - p[sta[hh - 1]]) > 0)
            st[sta[hh --]] = false;
        st[i] = true;
        sta[++ hh] = i; // 新入栈
    }
    // 下凸壳
    st[1] = false; // 1 这个点也需要更新
    for(int i = n; i; i --) {
        if(st[i]) continue; // 求下凸壳时不影响上凸壳
        while(hh >= 2 && cross(p[sta[hh]] - p[sta[hh - 1]], p[i] - p[sta[hh - 1]]) > 0)
            hh --;
        sta[++ hh] = i;
    }
    double ans = 0.0;
    for(int i = 2; i <= hh; i ++)
        ans += distanc(p[sta[i]], p[sta[i - 1]]);
    return ans;
}
int main() {
    n = read();
    for(int i = 1; i <= n; i ++) {
        double x, y;
        scanf("%lf%lf", &x, &y);
        p[i] = Point(x, y);
    }
    sort(p + 1, p + n + 1); // 双关键字排序
    printf("%.2lf\n", solve());
    return 0;
}
```

3.10.2 三维凸包

</> 代码 3.11: /计算几何/三维凸包

```
#include <bits/stdc++.h>
using namespace std;
const int N = 210;
const double eps = 1e-12;
int n, m;
bool g[N][N];
double rand_eps() { return ((double)rand() / RAND_MAX - 0.5) * eps;}
struct Point {
    double x, y, z;
    void shake() { x += rand_eps(), y += rand_eps(), z += rand_eps();} // 扰动
    Point operator-(Point t) { return {x - t.x, y - t.y, z - t.z};}
    double operator^(Point t) { return x * t.x + y * t.y + z * t.z;}
    Point operator*(Point t) { return {y * t.z - t.y * z, z * t.x - x * t.z, x * t.y - y
     * t.x};}
    double len() { return sqrt(x * x + y * y + z * z);}
} q[N];
struct Plane {
    int v[3];
    Point norm(){ return (q[v[1]] - q[v[0]]) * (q[v[2]] - q[v[0]]);} // 法向量
    double area(){ return norm().len() / 2;} // 求面积
    bool above(Point a) { return ((a - q[v[0]]) ^ norm()) >= 0;}
} plane[N], np[N];
void get_convex_3d() {
    plane[m++] = {0, 1, 2};
    plane[m++] = {2, 1, 0};
    for (int i = 3; i < n; i++) {
        int cnt = 0;
        for (int j = 0; j < m; j++) {
            bool t = plane[j].above(q[i]);
            if (!t) np[cnt++] = plane[j];
            for (int k = 0; k < 3; k++)
                g[plane[j].v[k]][plane[j].v[(k + 1) % 3]] = t;
        }
        for (int j = 0; j < m; j++)
            for (int k = 0; k < 3; k++) {
                int a = plane[j].v[k], b = plane[j].v[(k + 1) % 3];
                if (g[a][b] && !g[b][a])
                    np[cnt++] = {a, b, i};
            }
        m = cnt;
        for (int j = 0; j < m; j++)
```

```
41              plane[j] = np[j];
42      }
43  }
44
45  int main() {
46      scanf("%d", &n);
47      for (int i = 0; i < n; i++) {
48          scanf("%lf%lf%lf", &q[i].x, &q[i].y, &q[i].z);
49          q[i].shake();
50      }
51      get_convex_3d();
52      double res = 0;
53      for (int i = 0; i < m; i++)
54          res += plane[i].area();
55      printf("%lf\n", res);
56      return 0;
57  }
```

3.10.3 动态凸包

</> 代码 3.12: /计算几何/动态凸包

```
1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  ll n;
5  map<ll, ll> top, down;
6  bool check_top(ll x, ll y) {
7      auto i = top.lower_bound(x);
8      if (i == top.end()) return 0;
9      if (i->first == x) return y <= i->second;
10     if (i == top.begin()) return 0;
11     auto j = i; j--;
12     return (i->first - j->first) * (y - j->second) - (i->second - j->second) * (x - j->
       first) <=0;
13 }
14 bool check_down(ll x, ll y) {
15     auto i = down.lower_bound(x);
16     if (i == down.end()) return 0;
17     if (i->first == x) return y >= i->second;
18     if (i == down.begin()) return 0;
19     auto j = i;
20     j--;
21     return (i->first - j->first) * (y - j->second) -(i->second - j->second) * (x - j->
       first) >= 0;
```

```
}
bool delete_top(map<ll,ll>::iterator i) {
    if (i == top.begin()) return 0;
    auto j = i, k = i;
    j--, k++;
    if (k == top.end()) return 0;
    if ((i->first - j->first) * (k->second - j->second) - (i->second - j->second) * (k->
    first - j->first) >= 0) {
        top.erase(i);
        return 1;
    }
    return 0;
}
bool delete_down(map<ll,ll>::iterator i) {
    if (i == down.begin()) return 0;
    auto j = i, k = i;
    j--, k++;
    if (k == down.end()) return 0;
    if ((i->first - j->first) * (k->second - j->second) - (i->second - j->second) * (k->
    first - j->first) <= 0) {
        down.erase(i);
        return 1;
    }
    return 0;
}
void insert_top(ll x, ll y) {
    if (check_top(x, y)) return;
    top[x] = y;
    auto i = top.find(x), j = i;
    if (i != top.begin()) {
        j--;
        while (delete_top(j++)) j--;
    }
    if (++j != top.end())
        while (delete_top(j--)) j++;
}
void insert_down(ll x, ll y) {
    if (check_down(x, y)) return;
    down[x] = y;
    auto i = down.find(x), j = i;
    if (i != down.begin()) {
        j--;
        while (delete_down(j++)) j--;
    }
```

```
    if (++j != down.end())
        while (delete_down(j--)) j++;
}
int main() {
    scanf("%lld", &n);
    for (ll i = 1; i <= n; i++) {
        ll op , x , y ;
        scanf("%lld%lld%lld", &op, &x, &y);
        if (op == 1) {
            insert_top(x, y); //插入上凸包
            insert_down(x, y); //插入下凸包
        } else {
            if (check_top(x, y) && check_down(x, y))
                puts("YES");
            else
                puts("NO");
        }
    }
    return 0;
}
```

3.11 旋转卡壳

3.11.1 最远点对

</> 代码 3.13: /计算几何/最远点对

```
#include <bits/stdc++.h>
#define x first
#define y second

using namespace std;
typedef pair<int, int> PII;
const int N = 50010;
int n, stk[N], top;
PII q[N];
bool used[N];
PII operator-(PII a, PII b) { return {a.x - b.x, a.y - b.y}; }
int operator*(PII a, PII b) { return a.x * b.y - a.y * b.x; }
int area(PII a, PII b, PII c) { return (b - a) * (c - a); }
int get_dist(PII a, PII b) {
    int dx = a.x - b.x;
    int dy = a.y - b.y;
    return dx * dx + dy * dy;
}
```

```cpp
void get_convex() {
    sort(q, q + n);
    for (int i = 0; i < n; i++) {
        while (top >= 2 && area(q[stk[top - 2]], q[stk[top - 1]], q[i]) <= 0) {
            if (area(q[stk[top - 2]], q[stk[top - 1]], q[i]) < 0)
                used[stk[--top]] = false;
            else top--;
        }
        stk[top++] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; i--) {
        if (used[i]) continue;
        while (top >= 2 && area(q[stk[top - 2]], q[stk[top - 1]], q[i]) <= 0)
            top--;
        stk[top++] = i;
    }
    top--; // 首尾重复
}
// 旋转卡壳
int rotating_calipers() {
    if (top <= 2)
        return get_dist(q[0], q[n - 1]);
    int res = 0;
    for (int i = 0, j = 2; i < top; i++) {
        auto d = q[stk[i]], e = q[stk[i + 1]];
        while (area(d, e, q[stk[j]]) < area(d, e, q[stk[j + 1]]))
            j = (j + 1) % top;
        res = max(res, max(get_dist(d, q[stk[j]]), get_dist(e, q[stk[j]])));
    }
    return res;
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d", &q[i].x, &q[i].y);
    get_convex();
    printf("%d\n", rotating_calipers());
    return 0;
}
```

3.11.2 最小矩形覆盖

</> 代码 3.14: /计算几何/最小矩形覆盖

```cpp
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
typedef pair<double, double> PDD;
const int N = 50010;
const double eps = 1e-12, INF = 1e20;
const double PI = acos(-1);
int n;
PDD q[N];
PDD ans[N];
double min_area = INF;
int stk[N], top;
bool used[N];
int sign(double x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;}
int cmp(double x, double y) { return fabs(x - y) < eps ? 0 : x < y ? -1 : 1;}
PDD operator+(PDD a, PDD b) { return {a.x + b.x, a.y + b.y}; }
PDD operator-(PDD a, PDD b) { return {a.x - b.x, a.y - b.y}; }
PDD operator*(PDD a, double t) { return {a.x * t, a.y * t}; }
PDD operator/(PDD a, double t) { return {a.x / t, a.y / t}; }
double operator*(PDD a, PDD b) { return a.x * b.y - a.y * b.x; }
double operator&(PDD a, PDD b) { return a.x * b.x + a.y * b.y; }
double area(PDD a, PDD b, PDD c) { return (b - a) * (c - a); }
double get_len(PDD a) { return sqrt(a & a); }
double project(PDD a, PDD b, PDD c) { return ((b - a) & (c - a)) / get_len(b - a); }
PDD norm(PDD a) { return a / get_len(a); }
PDD rotate(PDD a, double b) { return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y *
     cos(b)}; }
void get_convex() {
    sort(q, q + n);
    for (int i = 0; i < n; i++) {
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) >= 0)
            used[stk[--top]] = false;
        stk[top++] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; i--) {
        if (used[i]) continue;
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) >= 0)
            top--;
        stk[top++] = i;
    }
```

```
    reverse(stk, stk + top);
    top--;
}
void rotating_calipers() {
    for (int i = 0, a = 2, b = 1, c = 2; i < top; i++) {
        auto d = q[stk[i]], e = q[stk[i + 1]];
        while (cmp(area(d, e, q[stk[a]]), area(d, e, q[stk[a + 1]])) < 0)
            a = (a + 1) % top;
        while (cmp(project(d, e, q[stk[b]]), project(d, e, q[stk[b + 1]])) < 0)
            b = (b + 1) % top;
        if (!i) c = a;
        while (cmp(project(d, e, q[stk[c]]), project(d, e, q[stk[c + 1]])) > 0)
            c = (c + 1) % top;
        auto x = q[stk[a]], y = q[stk[b]], z = q[stk[c]];
        auto h = area(d, e, x) / get_len(e - d);
        auto w = ((y - z) & (e - d)) / get_len(e - d);
        if (h * w < min_area) {
            min_area = h * w;
            ans[0] = d + norm(e - d) * project(d, e, y);
            ans[3] = d + norm(e - d) * project(d, e, z);
            auto u = norm(rotate(e - d, -PI / 2));
            ans[1] = ans[0] + u * h;
            ans[2] = ans[3] + u * h;
        }
    }
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%lf%lf", &q[i].x, &q[i].y);
    get_convex();
    rotating_calipers();
    int k = 0;
    for (int i = 1; i < 4; i++)
        if (cmp(ans[i].y, ans[k].y) < 0 ||!cmp(ans[i].y, ans[k].y) && cmp(ans[i].x, ans[
    k].x))
            k = i;
    printf("%.5lf\n", min_area);
    for (int i = 0; i < 4; i++, k++) {
        auto x = ans[k % 4].x, y = ans[k % 4].y;
        if (!sign(x)) x = 0;
        if (!sign(y)) y = 0;
        printf("%.5lf %.5lf\n", x, y);
    }
```

```
86      return 0;
87  }
```

3.12 最小圆覆盖

</> 代码 3.15: /计算几何/最小圆覆盖

```
1   #include <bits/stdc++.h>
2   #define x first
3   #define y second
4   using namespace std;
5   typedef pair<double, double> PDD;
6   const int N = 100010;
7   const double eps = 1e-12;
8   const double PI = acos(-1);
9   int n;
10  PDD q[N];
11  struct Circle {
12      PDD p;
13      double r;
14  };
15  int sign(double x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;}
16  int cmp(double x, double y) { return fabs(x - y) < eps ? 0 : x < y ? -1 : 1;}
17  PDD operator-(PDD a, PDD b) { return {a.x - b.x, a.y - b.y}; }
18  PDD operator+(PDD a, PDD b) { return {a.x + b.x, a.y + b.y}; }
19  PDD operator*(PDD a, double t) { return {a.x * t, a.y * t}; }
20  PDD operator/(PDD a, double t) { return {a.x / t, a.y / t}; }
21  double operator*(PDD a, PDD b) { return a.x * b.y - a.y * b.x; }
22  PDD rotate(PDD a, double b) { return {a.x * cos(b) + a.y * sin(b), -a.x * sin(b) + a.y *
        cos(b)}; }
23  double get_dist(PDD a, PDD b) {
24      double dx = a.x - b.x;
25      double dy = a.y - b.y;
26      return sqrt(dx * dx + dy * dy);
27  }
28  PDD get_line_intersection(PDD p, PDD v, PDD q, PDD w) {
29      auto u = p - q;
30      double t = w * u / (v * w);
31      return p + v * t;
32  }
33  pair<PDD, PDD> get_line(PDD a, PDD b) {
34      return {(a + b) / 2, rotate(b - a, PI / 2)};
35  }
36  Circle get_circle(PDD a, PDD b, PDD c) {
```

```
    auto u = get_line(a, b), v = get_line(a, c);
    auto p = get_line_intersection(u.x, u.y, v.x, v.y);
    return {p, get_dist(p, a)};
}
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%lf%lf", &q[i].x, &q[i].y);
    random_shuffle(q, q + n); // 随机打乱
    Circle c({q[0], 0});
    for (int i = 1; i < n; i++)
        if (cmp(c.r, get_dist(c.p, q[i])) < 0) {
            c = {q[i], 0};
            for (int j = 0; j < i; j++)
                if (cmp(c.r, get_dist(c.p, q[j])) < 0) {
                    c = {(q[i] + q[j]) / 2, get_dist(q[i], q[j]) / 2};
                    for (int k = 0; k < j; k++)
                        if (cmp(c.r, get_dist(c.p, q[k])) < 0)
                            c = get_circle(q[i], q[j], q[k]);
                }
        }
    printf("%.10lf\n", c.r);
    printf("%.10lf %.10lf\n", c.p.x, c.p.y);
    return 0;
}
```

3.13 半平面交

求多边形的面积并

</> 代码 3.16: /计算几何/半平面交

```
using namespace std;
const double INF = 1e12;
const double pi = acos(-1.0);
const double eps = 1e-8;
int sign(double x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;}
struct Point {
    double x, y;
    Point(double x = 0.0, double y = 0.0) : x(x), y(y) {}
    Point operator+(Point B) {return Point(x + B.x, y + B.y);}
    Point operator-(Point B) {return Point(x - B.x, y - B.y);}
    Point operator*(double k) {return Point(x * k, y * k);}
};
typedef Point Vec;
```

```
14 double Cross(Vec A, Vec B) { return A.x * B.y - A.y * B.x; }//叉积
15 struct Line {
16     Point p; Vec v;
17     double ang;
18     Line(Point p=Point(), Vec v=Point()) : p(p), v(v) {ang = atan2(v.y, v.x);}
19     bool operator<(Line &L) {return ang < L.ang; }//用于极角排序
20 };
21 //点p在线L左边，即点p在线L在外面：
22 bool OnLeft(Line L, Point p) {return sign(Cross(L.v, p - L.p)) > 0;}
23 Point Cross_point(Line a, Line b) {//两直线交点
24     Vec u = a.p - b.p;
25     double t = Cross(b.v, u) / Cross(a.v, b.v);
26     return a.p + a.v * t;
27 }
28 vector<Point> HPI(vector<Line> L) {//求半平面交，返回图多边形
29     int n = L.size();
30     sort(L.begin(), L.end()); //将所有半平面按照极角排序。
31     int first, last;    //指向双端队列的第一个和最后一个元素
32     vector<Point> p(n); //两个相邻半平面的交点
33     vector<Line> q(n);  //双端队列
34     vector<Point> ans;  //半平面交形成的凸包
35     q[first = last = 0] = L[0];
36     for (int i = 1; i < n; i++) {
37         //情况1：删除尾部的半平面
38         while (first < last && !OnLeft(L[i], p[last - 1]))
39             last--;
40         //情况2：删除首部的半平面：
41         while (first < last && !OnLeft(L[i], p[first]))
42             first++;
43         q[++last] = L[i]; //将当前的半平面加入双端队列尾部
44         //极角相同的两个半平面，保留左边：
45         if (fabs(Cross(q[last].v, q[last - 1].v)) < eps) {
46             last--;
47             if (OnLeft(q[last], L[i].p)) q[last] = L[i];
48         }
49         //计算队列尾部半平面交点：
50         if (first < last) p[last - 1] = Cross_point(q[last - 1], q[last]);
51     }
52     //情况3：删除队列尾部的无用半平面
53     while (first < last && !OnLeft(q[first], p[last - 1])) last--;
54     if (last - first <= 1) return ans;    //空集
55     p[last] = Cross_point(q[last], q[first]); //计算队列首尾部的交点。
56     for (int i = first; i <= last; i++)
57         ans.push_back(p[i]); //复制。
```

```
    return ans;                 //返回凸多边形
}
double Polygon_area(vector<Point> p) {
    double area = 0;
    for (int i = 0; i < (int)p.size(); i++)
        area += Cross(p[i], p[(i + 1) % (int)p.size()]);
    return area / 2; //面积有正负，不能简单地取绝对值
}
int main() {
    int n, m;
    vector<Line> L; // 半平面
    cin >> n;
    while (n--) { //点坐标逆时针给出
        cin >> m;
        vector<Point> tmp; //点集
        for (int i = 0; i < m; i++) {
            double a, b;
            scanf("%lf%lf", &a, &b);
            tmp.push_back(Point(a, b));
        }
        for (int i = 0; i < m; i++) {
            L.push_back(Line(tmp[i], tmp[(i + 1) % m] - tmp[i]));
        }
    }
    vector<Point> ans = HPI(L); //得到凸多边形
    printf("%.3f", Polygon_area(ans));
    return 0;
}
```

3.14 辛普森积分

3.14.1 自适应辛普森积分

对于一个定积分

$$\int_a^b f(x)\mathrm{d}x = F(b) - F(a)$$

但是如果原函数求导困难，可以使用二次函数逼近

$$\int_a^b f(x)\mathrm{d}x \approx \frac{(b-a)}{6}(f(a) + 4 \times f(\frac{2}{a+b}) + f(b))$$

主要用于求不规则的图形面积，以及简单函数求导（面积）

</> 代码 3.17: /计算几何/自适应辛普森积分

```
double simpson(double l,double r){
    double mid = ( l + r ) / 2;
    return (r-l) * (f(l) + 4 * f(mid) + f(r)) / 6;
```

```
} // f 为被积函数
double asr(double l, double r, double s){
    double mid = ( l + r ) / 2;
    double L = simpson(l, mid), R = simpson(mid, r);
    if(fabs(s - L - R ) < eps) return L + R; // 根据设置的 eps 自适应拟合精度
    return asr(l, mid, L) + asr(mid, r, R);
}
printf("%.6lf\n", asr(l, r, simpson(l, r)));
```

3.14.2 二重自适应辛普森积分

$$\int_a^b \int_c^d f(x,y)\mathrm{d}x\mathrm{d}y = \int_a^b F_x(y)\mathrm{d}y \approx \frac{b-a}{6}(F_x(a) + 4F_x(\frac{a+b}{2}) + F_x(b)))$$

$$F_x(y) = \int_c^d f(x,y)\mathrm{d}x \approx \frac{d-c}{6}(f(c,y) + 4f(\frac{d+c}{2}, y) + f(d,y))$$

</> 代码 3.18: /计算几何/二重自适应辛普森积分

```
#include <iostream>
#include <stdlib.h>
using namespace std;
double f(double x, double y) {return x * x + y;}
double simpsonX2(double a, double b, double y) { // a,d为积分下、上限，y为被固定的y值
    double c = (b + a) / 2.0;
    return (f(a, y) + 4 * f(c, y) + f(b, y)) * (b - a) / 6.0; //对x的辛普森近似
}
double adspX2(double a, double b, double y, double eps, double S) { //对x的自适应辛普森递归
    double c = (b + a) / 2.0;
    double L = simpsonX2(a, c, y), R = simpsonX2(c, b, y);
    if (abs(L + R - S) <= 15.0 * eps)  return L + R + (L + R - S) / 15.0;
    return adspX2(a, c, y, eps / 2.0, L) + adspX2(c, b, y, eps / 2.0, R);
}
double inte2(double a, double b, double y, double eps) { //固定y后，对x的积分
    return adspX2(a, b, y, eps, simpsonX2(a, b, y));
}
double simpsonY2(double xa, double xb, double ya, double yb, double eps) {
    double yc = (ya + yb) / 2.0;
    return (inte2(xa, xb, ya, eps) + 4 * inte2(xa, xb, yc, eps) + inte2(xa, xb, yb, eps)
    ) * (yb - ya) / 6.0; //对y的辛普森近似
}
double adspY2(double xa, double xb, double ya, double yb, double eps, double S) { //对y的自适应辛普森递归
    double L = simpsonY2(xa, xb, ya, (ya + yb) / 2, eps);
    double R = simpsonY2(xa, xb, (ya + yb) / 2, yb, eps);
```

```
25      if (abs(L + R - S) <= 15.0 * eps) return L + R + (L + R - S) / 15.0;
26      return adspY2(xa, xb, ya, (ya + yb) / 2.0, eps / 2.0, L) + adspY2(xa, xb, (ya + yb)
        / 2.0, yb, eps / 2.0, R);
27  }
28  double intergation2(double xa, double xb, double ya, double yb, double eps) { //求二重积
        分
29      return adspY2(xa, xb, ya, yb, eps, simpsonY2(xa, xb, ya, yb, eps));
30  }
31  int main() {
32      cout << intergation2(0, 1, 0, 1, 0.00001) << endl;
33      return 0;
34  }
```

3.14.3 圆的面积并

</> 代码 3.19: /计算几何/圆的面积并

```
1   #include <bits/stdc++.h>
2   using namespace std;
3   typedef pair<double, double> PDD;
4   const int N = 1010;
5   const double eps = 1e-8;
6   struct Circle{
7       double x, y, r;
8       Circle(double _x = 0.0, double _y = 0.0, double _r = 0.0) : x(_x), y(_y), r(_r) {}
9   } cir[N];
10  int n;
11  PDD lr[N * 2];
12  int sign(double x) { return fabs(x) < eps ? 0 : x < 0 ? -1 : 1;}
13  int cmp(double x, double y) { return fabs(x - y) < eps ? 0 : x < y ? -1 : 1;}
14  double f(double x) {
15      int idx = 0;
16      for(int i = 0; i < n; i ++) {
17          double dlt = fabs(cir[i].x - x);
18          if(cir[i].r > dlt) {
19              double h = sqrt(cir[i].r * cir[i].r - dlt * dlt);
20              lr[idx ++] = {cir[i].y - h, cir[i].y + h};
21          }
22      }
23      if(idx == 0) return 0; // 因为这个数组没清空过所以要判
24      sort(lr, lr + idx);
25      double tot = 0.0, st = lr[0].first, en = lr[0].second;
26      for(int i = 1; i < idx; i ++) {
27          if(lr[i].first < en)
28              en = max(en, lr[i].second);
```

```
29          else
30              tot += en - st, st = lr[i].first, en = lr[i].second;
31      }
32      return tot + en - st;
33  }
34  double simposon(double l, double r) {
35      double mid = (l + r) / 2;
36      return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
37  }
38  double asr(double l, double r, double s) {
39      double mid = (l + r) / 2;
40      double L = simposon(l, mid), R = simposon(mid, r);
41      if(fabs(s - L - R) < eps) return L + R;
42      return asr(l, mid, L) + asr(mid, r, R);
43  }
44  int main() {
45      scanf("%d", &n);
46      for(int i = 0 ; i <= n; i ++) {
47          scanf("%lf%lf%lf", &cir[i].x, &cir[i].y, &cir[i].r);
48      }
49      double l = -2000, r = 2000;
50      printf("%.3lf\n", asr(l, r, simposon(l, r)));
51      return 0;
52  }
```

3.15 闵可夫斯基和

给两个凸包 A, B, q 次询问，每次给一个移动量 w 来移动 B 凸包，问 A 与 B 是否有交点令 $a \in A, b \in B$, 则移动向量 w 使得存在 $b + w = a$, 那么 w 需要满足 $w = a - b$ 构造闵可夫斯基合 $C = a + (-b)$，只要判断移动向量是否在凸包 C 内。

</> 代码 3.20: /计算几何/闵可夫斯基和

```
1  #include <bits/stdc++.h>
2  #define ll long long
3  using namespace std;
4  const ll N = 1e5 + 10;
5  struct Point {
6      ll x, y;
7      Point operator-(Point A) { return (Point){x - A.x, y - A.y}; }
8      Point operator+(Point A) { return (Point){x + A.x, y + A.y}; }
9      ll operator*(Point A) const { return x * A.y - y * A.x; } // 叉积
10     ll len() const { return x * x + y * y; }
11 } A[N], C1[N], C2[N], s1[N], s2[N], bs;
12 ll cmp1(const Point &A, const Point &B) { return A.y < B.y || (A.y == B.y && A.x < B.x)
```

```
    ;}
ll cmp2(const Point &A, const Point &B) { return A * B > 0 || (A * B == 0 && A.len() < B.len()); }
ll n, m, sta[N], top, q, tot;
void Convex(Point *A, ll &n) { // 求凸包
    sort(A + 1, A + n + 1, cmp1);
    bs = A[1];
    sta[top = 1] = 1;
    for (ll i = 1; i <= n; i++)
        A[i] = A[i] - bs;
    sort(A + 2, A + n + 1, cmp2);
    for (ll i = 2; i <= n; sta[++top] = i, i++)
        while (top >= 2 && (A[i] - A[sta[top - 1]]) * (A[sta[top]] - A[sta[top - 1]]) >= 0)
            top--;
    for (ll i = 1; i <= top; i++)
        A[i] = A[sta[i]] + bs;
    n = top;
    A[n + 1] = A[1];
}
void Minkowski() { // 闵可夫斯基合
    for (ll i = 1; i < n; i++) s1[i] = C1[i + 1] - C1[i];
    s1[n] = C1[1] - C1[n];
    for (ll i = 1; i < m; i++) s2[i] = C2[i + 1] - C2[i];
    s2[m] = C2[1] - C2[m];
    A[tot = 1] = C1[1] + C2[1];
    ll p1 = 1, p2 = 1;
    while (p1 <= n && p2 <= m) ++tot, A[tot] = A[tot - 1] + (s1[p1] * s2[p2] >= 0 ? s1[p1++] : s2[p2++]);
    while (p1 <= n) ++tot, A[tot] = A[tot - 1] + s1[p1++];
    while (p2 <= m) ++tot, A[tot] = A[tot - 1] + s2[p2++];
}
ll in(Point a) { // 判断点是否在凸包内
    if (a * A[1] > 0 || A[tot] * a > 0) return 0;
    ll ps = lower_bound(A + 1, A + tot + 1, a, cmp2) - A - 1;
    return (a - A[ps]) * (A[ps % tot + 1] - A[ps]) <= 0;
}
int main() {
    cin >> n >> m >> q;
    for (ll i = 1; i <= n; i++)
        scanf("%lld%lld", &C1[i].x, &C1[i].y);
    Convex(C1, n); // A 凸包
    for (ll i = 1; i <= m; i++) {
        scanf("%lld%lld", &C2[i].x, &C2[i].y);
```

```
53          C2[i].x = -C2[i].x; // 反向存
54          C2[i].y = -C2[i].y;
55      }
56      Convex(C2, m); // B 凸包
57      Minkowski(); // 闵可夫斯基合
58      Convex(A, tot); // 求闵可夫斯基合凸包
59      bs = A[1];
60      for (ll i = tot; i >= 1; i--) A[i] = A[i] - A[1];
61      while (q--) {
62          scanf("%lld%lld", &A[0].x, &A[0].y);
63          printf("%lld\n", in(A[0] - bs));// 判断输入量是否在闵可夫斯基合内
64      }
65      return 0;
66  }
```

第四章　数论

4.1　乘法加速系列

4.1.1　快速乘

处理超范围的取模乘法

</> 代码 4.1:　/数论/快速乘

```
ll qmul(ll a, ll b, ll p) {
    ll ans = 0;
    while(b){
        if(b&1) ans = (ans + a) % p;
        a = a * 2 % p;
        b >>= 1;
    }
    return ans;
}
```

4.1.2　快速幂

</> 代码 4.2:　/数论/快速幂

```
ll qp(ll a, ll b, ll p) {
    ll ans = 1;
    while(b) {
        if(b & 1)ans = (ans * a) %p;
        a = (a * a)%p;
        b >>= 1;
    }
    return ans;
}
```

4.1.3　虚数快速幂

</> 代码 4.3:　/数论/虚数快速幂

```
ll n, mod = 998244353;
struct ty{
```

```
    ll x, y; //复数形如x+yi
    ty operator* (const ty &b){
        ty ans;
        ans.x = ((x * b.x % mod - y *b.y % mod) % mod + mod % mod);
        ans.y = (x *b.y % mod + y * b.x % mod) % mod;
        return ans;
    }
} a, b;
ll ksm1(ll  a,ll  b){
    ll ans = 1, t = a;
    while(b){
        if (b & 1) ans =ans * t% mod;
        t = t * t % mod;
        b>>=1;
    }
    return ans;
}

ll ksm2(ty a,ll b) {//此处不反回ty类型因为(1 + i) ^ n + (1 - i) ^ n 虚数都是奇次被抵消了;
    ty ans, t = a; ans.x = 1;  ans.y =0;
    while(b){
        if (b & 1) ans = ans * t;
        t = t * t;
        b>>=1;
    }
    return ans.x % mod;
}

int main(){
    a.x = 1, a.y = 1;
    b.x = 1, b.y = mod-1;
    cin >> n;
    ll ans = ((ksm1(2,n)+ ksm2(a, n)) % mod + ksm2(b, n) %mod) * ksm1(4, mod-2)/*乘法逆元
    */ %mod;
    cout <<(ans+mod )%mod;
    return 0;
}
```

4.1.4 矩阵快速幂

</> 代码 4.4: /数论/矩阵快速幂

```
//以斐波那契数列为模板
const int M = 2;
struct ma{
```

```
    int a[M][M];//含有一个M*M的矩阵
    ma(){ memset(a,0,sizeof(a));}
    void unit(){
        a[0][0]=a[1][1]=1;//需符合矩阵乘法与递推关系
        a[0][1]=a[1][0]=0;
    }
    ma operator*(const ma &b)const {//定义矩阵乘法，并给操作符*
        ma ans;
        for(int i=0; i<M; i++)
            for(int j=0; j<M; j++)
                for(int k=0; k<M; k++)
                    ans .a[i][j]+=a[i][k]*b.a[k][j];
        return ans;
    }
    ma operator^(int n)const {//定义矩阵幂，并给操作符^
        ma ans; ans.unit();
        ma A=*this;
        while(n) {
            if(n&1)ans = ans*A;
            A = A*A;
            n>>=1;
        }
        return ans;
    }
    void show() {//打印
        for(int i=0; i<M; i++){
            for(int j=0; j<M; j++)
                cout<<a[i][j]<<" ";
            cout<<endl;
        }
    }
};
int main() {
    ma A;
    A.a[0][0]= 0;
    A.a[0][1]=A.a[1][0]=A.a[1][1]=1;

    ma F;
    F.a[0][0]=F.a[1][0]=1;
    F.a[0][1]=F.a[1][1]=0;

    int n = 8;
    ma ans=(A^n)*F;//矩阵快速幂
    ans.show();
```

```
48      return 0;
49  }
```

4.2 素数判断

4.2.1 试除法

</> 代码 4.5: /数论/试除法

```
1  bool is_prime(int n){
2      if(n < 2) return 0;
3      for(int i=2; i <= n / i; i++){
4          if(n%i == 0) return 0;
5      }
6      return 1;
7  }
```

4.2.2 Miller-Rabin

Miller-Rabin 素性测试 $O(k \log^3 n)$

</> 代码 4.6: /数论/Miller-Rabin

```
1  // 2, 7, 61||2,325,9375,28178,450775,9780504,1795265022
2  //但这几个数本身需要特判
3  ll qp(ll n, ll x, ll mod) {
4      ll res = 1;
5      while (x) {
6          if (x & 1)
7              res = res * n % mod;
8          n = n * n % mod;
9          x >>= 1;
10     }
11     return res;
12 }
13
14 bool millerRabbin(ll n) {
15     if (n < 3 || n % 2 == 0) return n == 2;
16     ll a = n - 1, b = 0;
17     while (a % 2 == 0)
18         a /= 2, b ++;
19     ll test_time = 10; //可以使用特定底数
20     for (int i = 1, j; i <= test_time; i ++) {//fermat判断
21         ll x = rand() % (n - 2) + 2, v = qp(x, a, n);
22         if (v == 1)  continue;
23         for (j = 0; j < b; ++j) {//二次探测定理
```

```
24              if (v == n - 1) break;
25              v = v * v % n;
26          }
27          if (j >= b) return false;
28      }
29      return true;
30  }
```

4.3 欧拉

4.3.1 欧拉函数

$\varphi(i)$ 表示第 i 个欧拉函数的值，表示从 1 到 i 中与 i 互质的数的个数
公式：

$$\varphi(i) = x \times \prod_{i=1}^{n}(1 - \frac{1}{p_i})(P_i\ x \quad)$$

eg: $\varphi(8) = 8 \times (1 - \frac{1}{2}) = 4$
如果 n, m 互质 $\varphi(n \times m) = \varphi(n) * \varphi(m)$
如果 p 是素数, 且 n 是 p 的倍数，$\varphi(n \times p) = \varphi(n) * p$
如果 n 是素数 $\varphi(n) = n - 1$
如果 n 是奇数 $\varphi(2 \times n) = \varphi(n)$
如果 $n \geq 2$ 是素数 $\varphi(n)$ 为偶数
如果 p 是素数 $\varphi(p^q) = p^q - p^{q-1}$

$$\sum_{d|n} \varphi(d) = n$$

$$\sum_{i=1}^{n} [\gcd(n, i) = 1] = \varphi(n)$$

$$\sum_{i=1}^{n} i * [\gcd(n, i) = 1] = \left\lceil \frac{\varphi(n) * n}{2} \right\rceil$$

</> 代码 4.7: /数论/欧拉函数

```
1  int phi(int x){
2      int ans = x;
3      for(int i = 2; i*i <= x; i++){
4          if(x % i == 0){
5              ans = ans / i * (i-1);
6              while(x % i == 0) x /= i;
7          }
8      }
9      if(x > 1) ans = ans / x * (x-1);
10     return ans;
11 }
```

4.3.2 欧拉定理

$$a^{\varphi(m)} \equiv 1 \ (mod \ m)(\gcd(a,m)=1)$$

欧拉降幂

$$a^b \equiv \begin{cases} a^{b \ mod \ \varphi(p)} \gcd(a,p)=1 \\ a^b \gcd(a,p) \neq 1, b \leqslant \varphi(p)(mod \ p) \\ a^{b \ mod \ \varphi(p)+\varphi(p)} \gcd(a,p) \neq 1, b \geqslant \varphi(p) \end{cases}$$

递归求

$$2^{2^{2^{\cdots}}} \mod p$$

</> 代码 4.8: /数论/递归欧拉降幂

```
ll dfs(ll mod){
    if(mod == 1) return 0;
    return quick_pow(2, dfs(phi[mod]) + phi[mod], mod);
}
```

4.4 威尔逊定理

当 n 为质数时有:

$$(n-1)! = -1 \ (mod \ n)$$
$$(n-2)! = 1 \ (mod \ n)$$

4.5 费马定理

费马小定理:

$$a^m \equiv a(mod \ m), a^{m-1} \equiv 1(mod \ m)$$

费马小定理降幂

$$a^p \equiv a^{p\%m}(mod \ m)$$

求逆元

$$inv(a) = a^{m-2}$$

费马大定理:

$$x^n + y^n = z^n$$

在 $n > 2$ 时无解，当 $n = 2$ 时，当 a 为奇数 $a, \frac{a^2}{2}, \frac{a^2}{2}+1$，当 a 为偶数 $a, \frac{a^2}{4}-1, \frac{a^2}{4}+1$

4.6 素数

4.6.1 反素数

求因子个数一定的最小的数

</> 代码 4.9: /数论/因子数固定的最小的数

```
ull p[16] = {
    2,  3,  5,  7,  11, 13, 17, 19,
    23, 29, 31, 37, 41, 43, 47, 53}; //根据数据范围可以确定使用的素数最大为53
ull ans, n;
// depth: 当前在枚举第几个素数
// temp: 当前因子数量为 num的时候的数值
// num: 当前因子数
// up: 上一个素数的幂，这次应该小于等于这个幂次
void dfs(ull depth, ull temp, ull num, ull up) {
    if (num > n || depth >= 16) return;     //边界条件
    if (num == n && ans > temp) { //取最小的ans
        ans = temp;
        return;
    }
    for (int i = 1; i <= up; i++) {
        if (temp * p[depth] > ans)
            break; //剪枝：如果加一个这个乘数的结果比ans要大，则必不是最佳方案
        //取一个该乘数，进行对下一个乘数的搜索
        dfs(depth + 1, temp = temp * p[depth], num * (i + 1), i);
    }
}
int main() {
    scanf("%llu", &n); // 输入的 n 为因数个数
    ans = ~(ull)0;
    dfs(0, 1, 1, 64);
    printf("%llu\n", ans);
    return 0;
}
```

4.6.2 因子数最多的数

</> 代码 4.10: /数论/因子数最多的数

```
ull p[16] = {
    2,  3,  5,  7,  11, 13, 17, 19,
    23, 29, 31, 37, 41, 43, 47, 53}; //根据数据范围可以确定使用的素数最大为53
ull ans, n, ans_num;

void dfs(ull depth, ull temp, ull num, ull up) {
    if (depth >= 16 || temp > n)
        return;
    if (num > ans_num) { //更新答案
        ans = temp; ans_num = num;
    }
```

```
12      if (num == ans_num && ans > temp)
13          ans = temp; //更新答案
14      for (int i = 1; i <= up; i++) {
15          if (temp * p[depth] > n)
16              break; //剪枝：如果加一个这个乘数的结果比ans要大，则必不是最佳方案
17          dfs(depth + 1, temp *= p[depth], num * (i + 1),
18              i); //取一个该乘数，进行对下一个乘数的搜索
19      }
20      return;
21  }
22  int main() {
23      while (scanf("%llu", &n) != EOF) {
24          ans_num = 0;
25          dfs(0, 1, 1, 60);
26          printf("%llu\n", ans);
27      }
28      return 0;
29  }
```

4.6.3 Pollard-Rho 算法

可以判断一个数是否为质数（Miller-Rabin），如果不是质数可以求出最大的质因子，$O(N^{\frac{1}{4}})$

</> 代码 4.11: /数论/Pollard-Rho

```
1   #include <bits/stdc++.h>
2   using namespace std;
3
4   typedef long long ll;
5
6   int t;
7   ll max_factor, n;
8
9   ll gcd(ll a, ll b) {
10      return b ? gcd(b, a % b) : a;
11  }
12
13  ll quick_pow(ll x, ll p, ll mod) { //快速幂
14      ll ans = 1;
15      while (p) {
16          if (p & 1)
17              ans = (__int128)ans * x % mod;
18          x = (__int128)x * x % mod;
19          p >>= 1;
20      }
21      return ans;
```

```
}

bool Miller_Rabin(ll p) { //判断素数
    if (p < 2)  return 0;
    if (p == 2) return 1;
    if (p == 3) return 1;
    ll d = p - 1, r = 0;
    while (!(d & 1)) ++r, d >>= 1; //将d处理为奇数
    for (ll k = 0; k < 10; ++k) {
        ll a = rand() % (p - 2) + 2;
        ll x = quick_pow(a, d, p);
        if (x == 1 || x == p - 1) continue;
        for (int i = 0; i < r - 1; ++i) {
            x = (__int128)x * x % p;
            if (x == p - 1) break;
        }
        if (x != p - 1) return 0;
    }
    return 1;
}

ll Pollard_Rho(ll x) {
    ll s = 0, t = 0;
    ll c = (ll)rand() % (x - 1) + 1;
    int step = 0, goal = 1;
    ll val = 1;
    for (goal = 1;; goal *= 2, s = t, val = 1) { //倍增优化
        for (step = 1; step <= goal; ++step) {
            t = ((__int128)t * t + c) % x;
            val = (__int128)val * abs(t - s) % x;
            if ((step % 127) == 0) {
                ll d = gcd(val, x);
                if (d > 1)
                    return d;
            }
        }
        ll d = gcd(val, x);
        if (d > 1)
            return d;
    }
}

void fac(ll x) {
    if (x <= max_factor || x < 2)
```

```
66          return;
67      if (Miller_Rabin(x)) {                  //如果x为质数
68          max_factor = max(max_factor, x); //更新答案
69          return;
70      }
71      ll p = x;
72      while (p >= x)
73          p = Pollard_Rho(x); //使用该算法
74      while ((x % p) == 0)
75          x /= p;
76      fac(x), fac(p); //继续向下分解x和p
77  }
78
79  int main() {
80      scanf("%d", &t);
81      while (t--) {
82          srand((unsigned)time(NULL));
83          max_factor = 0;
84          scanf("%lld", &n);
85          fac(n);
86          if (max_factor == n) //最大的质因数即自己
87              printf("Prime\n");
88          else
89              printf("%lld\n", max_factor);
90      }
91      return 0;
92  }
```

4.7 筛法

4.7.1 埃拉托斯特尼

$O(n \log \log n)$ 可以生成一个筛树，并且可以筛一段区间 $[l, r]$ 内的质数

</> 代码 4.12: /数论/埃拉托斯特尼

```
1  void get_primes(int n) {
2      for(int i = 2; i <= n; i++) {
3          if(!st[i]){
4              prime[cnt++] = i;
5              for(int j = i; j <= n; j += i)
6                  st[j] = true;
7          }
8      }
9  }
```

4.7.2 分块筛选

计算 n 内质数的个数，时间复杂度接近欧拉筛。(分块大小在 10^4 到 10^5 之间时效率最好)

</> 代码 4.13: /数论/分块筛选

```
int count_primes(int n) {
  const int S = 10000;
  vector<int> primes;
  int nsqrt = sqrt(n);
  vector<char> is_prime(nsqrt + 1, true);
  for (int i = 2; i <= nsqrt; i++) {
    if (is_prime[i]) {
      primes.push_back(i);
      for (int j = i * i; j <= nsqrt; j += i) is_prime[j] = false;
    }
  }
  int result = 0;
  vector<char> block(S);
  for (int k = 0; k * S <= n; k++) {
    fill(block.begin(), block.end(), true);
    int start = k * S;
    for (int p : primes) {
      int start_idx = (start + p - 1) / p;
      int j = max(start_idx, p) * p - start;
      for (; j < S; j += p) block[j] = false;
    }
    if (k == 0) block[0] = block[1] = false;
    for (int i = 0; i < S && start + i <= n; i++) {
      if (block[i]) result++;
    }
  }
  return result;
}
```

4.7.3 线性筛

每次都是该数的 **最小质因子**把它筛去，所有每个数只会被筛一次，复杂度 $O(n)$

</> 代码 4.14: /数论/线性筛

```
int get_prime(int x) {
    for (int i = 2; i <= x; i++) {
        if (!st[i]) prime[cnt++] = i;
        for (int j = 0; prime[j] * i <= x; j++) {
            st[prime[j] * i] = true;
            if (i % prime[j] == 0) break;
        }
```

```
    }
    return cnt;
}
```

4.7.4 线性筛欧拉函数

欧拉函数：1 n 中与 n 互质的数的个数，由于欧拉函数是积性函数（非完全），所有可以用线性筛来加速，复杂度 $O(n)$

</> 代码 4.15: /数论/线性筛欧拉函数

```
void get_euler(int n) {
    euler[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!st[i]) {
            prime[idx++] = i;
            euler[i] = i - 1;
        }
        for (int j = 0; prime[j] * i <= n; j ++) {
            int t = prime[j] * i;
            st[t] = 1;
            if (i % prime[j] == 0) {
                euler[t] = euler[i] * prime[j];
                break;
            }
            euler[t] = euler[i] * euler[prime[j]];
        }
    }
}
```

4.7.5 线性筛莫比乌斯函数

欧拉函数：1 n 中与 n 互质的数的个数，由于欧拉函数是积性函数（非完全），所有可以用线性筛来加速，复杂度 $O(n)$

</> 代码 4.16: /数论/线性筛莫比乌斯函数

```
void prework(){
    mu[1] = 1;
    for (int i = 2; i < N; i ++ )
    {
        if (!st[i]) prime[cnt ++ ] = i, mu[i] = -1;
        for (int j = 0; prime[j] * i < N; j ++ )
        {
            st[prime[j] * i] = true;
            if (i % prime[j] == 0) break;
            mu[prime[j] * i] = -mu[i];
```

```
        }
    }
}
```

4.8 欧几里得算法

4.8.1 GCD

</> 代码 4.17: /数论/GCD

```
/*
1.lcm(S/a, S/b) = S/gcd(a, b)
2.gcd(a,b) = gcd(a,a - b) => gcd(a1,a2,a3...an) = gcd(a1,a2 - a1,a3 - a2,a4 - a3...an - an - 1);
3.gcd(a,lcm(b,c)) = lcm(gcd(a,b),gcd(a,c))
4.lcm(a,gcd(b,c)) = gcd(lcm(a,b),lcm(a,c))
5.在坐标里，将点(0, 0)和(a, b)连起来，通过整数坐标的点的数目（除了(0, 0)一点之外）就是gcd(a, b)。
6.gcd(a,b)=gcd(b, a mod b)
7.gcd(ab,m)=gcd(a,m) * gcd(b,m)
 */
int gcd(int a, int b){
    return b ? gcd(b, a % b) : a;
}
int gcd(int a, int b){while(b^=a^=b^=a%=b);return a;}
int lcm(int a, int b){
    return (ll)a * b / gcd(a, b);
}
```

4.8.2 EXGCD

</> 代码 4.18: /数论/EXGCD

```
//求出ax + by = c的一组可行解
//有解的前提 gcd(a,b)|c ,否则无解
// (x % b + b) % x 可得最小正整数解
int exgcd(int a, int b, int &x, int &y) {
    if(a == 0 && b == 0) return -1;
    if(!b) {
        x = 1, y = 0;
        return a;
    }
    int g = exgcd(b, a % b, y, x);
    y -=  a / b * x;
    return g;
```

```
}
```

4.8.3 中国剩余定理

</> 代码 4.19: /数论/中国剩余定理

```cpp
ll CRT(int n, ll m[], ll Mi[]) {
    ll mul = 1, res = 0;
    for (int i = 1; i <= n; i ++) mul *= m[i];
    for (int i = 1; i <= n; i ++) {
        Mi[i] = mul / m[i];
        ll x = 0, y = 0;
        exgcd(Mi[i], m[i], x, y);
        res += r[i] * Mi[i] * (x < 0 ? x + m[i] : x);
    }
    return res % mul;
}

//判断是否有解

ll mod(ll a, ll b){ return ((a % b) + b) % b;}
int main(){
    int n;
    scanf("%d", &n);
    ll a1, m1, a2, m2, k1, k2;
    scanf("%lld%lld", &a1, &m1);
    for(int i = 1; i < n; i++){
        scanf("%lld%lld", &a2, &m2);
        ll d = exgcd(a1, -a2, k1, k2);
        if((m2 - m1) % d){ puts("-1"); return 0; } // 判断是否有解
        k1 = mod(k1 * (m2 - m1) / d, abs(a2 / d));
        m1 = k1 * a1 + m1;
        a1 = abs(a1 / d * a2);
    }
    printf("%lld\n", m1);
    return 0;
}
```

4.8.4 逆元

求任意 n 个数的逆元首先计算 n 个数的前缀积，记为 s_i，然后使用快速幂或扩展欧几里得法计算 s_n 的逆元，记为 sv_n。因为 sv_n 是 n 个数的积的逆元，所以当我们把它乘上 a_n 时，就会和 a_n 的逆元抵消，于是就得到了 a_1 到 a_{n-1} 的积逆元，记为 sv_{n-1}。同理我们可以依次计算出所有的 sv_i，于是 a_i^{-1} 就可以用 $s_{i-1} \times sv_i$ 求得。所以我们就在 $O(n + \log p)$ 的时间内计算出了 n 个数的逆元。

</> 代码 4.20: /数论/Manacher/01.cpp

```cpp
//逆元 (a / b) % mod = a * inv(b) % mod
//拓展欧几里得定理求inv(b)
int mod = 1e9 + 7;
int N;
ll exgcd(ll a,ll b,ll &x,ll &y){
    if(a == 0 && b == 0) return -1;
    if(b == 0){
        x = 1;y = 0;
        return a;
    }
    ll d = exgcd(b,a % b,y,x);
    y -= a / b * x;
    return d;
}
ll inv(ll a,ll n){
    ll x, y;
    ll d = exgcd(a,n,x,y);
    if(d == 1) return (x % n+ n) % n;
    else return -1;
}
//当mod为质数时, 可以利用费马小定理, b的逆元为n ^ (mod - 2) % mod
//当我们需要求连续一段数在mod p下的逆元的时候, 我们可以使用逆元的递推公式
//模板: 求1到N的所有逆元
ll mod = 1e9 + 7;
ll N, inv[maxn];
int main(){
    scanf("%lld%lld",&N,&mod);
    inv[1] = 1;
    for(int i = 2; i <= N; i ++) inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    return 0;
}

s[0] = 1;
for (int i = 1; i <= n; ++i) s[i] = s[i - 1] * a[i] % p;
sv[n] = qpow(s[n], p - 2);
// 当然这里也可以用 exgcd 来求逆元,视个人喜好而定.
for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] * a[i] % p;
for (int i = 1; i <= n; ++i) inv[i] = sv[i] * s[i - 1] % p;
```

4.9 高斯消元

</> 代码 4.21: /数论/高斯消元

```cpp
//高斯消元,时间复杂度O(n ^ 3)
/*
高斯消元是求解线性方程组的一种做法。将线性方程组列为矩阵的形式
x1 + 2x2 - x3 = -6
2x1 + x2 - 3x3 = -9
-x1 - x2 + 2x3 = 7
可以写成
1 2 -1 -6
0 -3 -1 3
-1 -1 2 7
这么一个矩阵,然后利用初等行变换化为最简求每个变量的值
 */
 //模板，给出像上述增广矩阵求每个变量的值
#include <bits/stdc++.h>
using namespace std;
const int M = 110;
const double eps = 1e-6; //防止浮点精度偏差
double a[M][M];
int n;
int gauss() {
    int r, c; // r是row（行），c是col（列）
    for (r = 0, c = 0; c < n; c++) {
        int idx = r;
        for (int i = r; i < n; i++) {
            if (fabs(a[idx][c]) < fabs(a[i][c]))
                idx = i; //寻找该列最大值（最大值可以使得答案精度较好）
        }
        if (fabs(a[idx][c]) < eps)
            continue; //如果最大值已经为0，则进入下一次循环（行不变，列++）
        for (int i = c; i <= n; i++)
            swap(a[idx][i], a[r][i]); //找出的开头最大列与原列交换
        for (int i = n; i >= c; i--)
            a[r][i] /= a[r][c]; //从行末向前更新，使得开头为1
        for (int i = r + 1; i < n; i++) {//更新该行下面的矩阵，使其靠近最简形矩阵
            if (fabs(a[i][c]) > eps) //如果该行的队头是0，则不执行；
                for (int j = n; j >= c; j--)
                    a[i][j] -= a[r][j] * a[i][c]; //从后向前更新该行的值
        }
        r++; //完整的一次操作后， 行 ++
    }
    if (r < n){ //判断矩阵的秩，确定解的个数，如果r == n，矩阵有解
        for (int i = r; i < n; i++) {
            if (a[i][n] > eps)
                return 2; //判断是否存在 0 = ! 0的情况
```

```
        }
        return 1; //否则无穷多解，相当于矩阵的秩 < n
    }
//从下往上，只对b i进行操作，因为最后b i的值便对应x直接输出
    for (int i = n - 1; i >= 0; i--)
        for (int j = i + 1; j < n; j++)
            a[i][n] -= a[j][n] * a[i][j];
    return 0;
}
int main() {
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n + 1; j++)
            cin >> a[i][j];
    int flag = gauss();
    if (flag == 1) {
        puts("Infinite group solutions");
    } else if (flag == 2) {
        puts("No solution");
    } else {
        for (int i = 0; i < n; i++)
            printf("%.2lf\n", a[i][n]);
    }
    return 0;
}

/*
高斯消元也可求解一类开关问题
所有开关的的开闭状态从一个指定状态变为src另一个指定状态dst的方法
改变一个开关j会改变另一个开关i的状态aij,可列出方程
a1,1 ^ x1 ^ a1,2 ^ x2 .... ^ a1n ^ xn = src1 ^ dst1
a2,1 ^ x1 ^ a2,2 ^ x2 .....^ a2n ^ xn = src2 ^ dst2
a为已知系数，src和dst为已知系数，直接高斯消元求解可行种数
种数就是(1 << 自由元个数)
n小的话可以用状态压缩,大的话可以用bitset
 */
 //POJ1830
//有N个相同的开关，每个开关都与某些开关有着联系，每当你打开或者关闭某个开关的时候，其他的与此开关相关联的开关也会相应地发生变化，即这些相联系的开关的状态如果原来为开就变为关，如果为关就变为开。你的目标是经过若干次开关操作后使得最后N个开关达到一个特定的状态。对于任意一个开关，最多只能进行一次开关操作。你的任务是，计算有多少种可以达到指定状态的方法。（不计开关操作的顺序）
#include <bits/stdc++.h>
using namespace std;
```

```cpp
const int N = 35;
int n;
int a[N][N];
int gauss() {
    int r, c;
    for (r = 1, c = 1; c <= n; c++) {
        // 找主元
        int t = r;
        for (int i = r + 1; i <= n; i++)
            if (a[i][c])
                t = i;
        if (!a[t][c]) continue;
        // 交换
        for (int i = c; i <= n + 1; i++) swap(a[t][i], a[r][i]);
        // 消
        for (int i = r + 1; i <= n; i++)
            for (int j = n + 1; j >= c; j--)
                a[i][j] ^= a[i][c] & a[r][j];
        r++;
    }
    int res = 1;
    if (r < n + 1) {
        for (int i = r; i <= n; i++) {
            if (a[i][n + 1])
                return -1; // 出现了 0 == !0, 无解
            res *= 2;
        }
    }
    return res;
}
int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        memset(a, 0, sizeof a);
        scanf("%d", &n);
        for (int i = 1; i <= n; i++)
            scanf("%d", &a[i][n + 1]);
        for (int i = 1; i <= n; i++) {
            int t;
            scanf("%d", &t);
            a[i][n + 1] ^= t;
            a[i][i] = 1;
        }
```

```
129         int x, y;
130         while (scanf("%d%d", &x, &y), x || y)
131             a[y][x] = 1;
132         int t = gauss();
133         if (t == -1)
134             puts("Oh,it's impossible~!!");
135         else
136             printf("%d\n", t);
137     }
138     return 0;
139 }
```

4.10 组合数学

4.10.1 排列数与组合数

$$A_n^m = \frac{n!}{(n-m)!}$$

$$C_n^m = \binom{n}{m} = \frac{A_n^m}{m!} = \frac{n!}{m!(n-m)!}$$

二项式定理

$$(a+b)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} b^i$$

不相邻的排列 $1 \sim n$ 这 n 个自然数中选 k 个，这 k 个数中任意两个数都不相邻的组合有

$$\binom{n-k+1}{k}$$

绝对错排

$$f(n) = (n-1)(f(n-1) + f(n-2))$$

前几项为 ‘0,1,2,9,44,265‘。

圆排列 n 个人全部来围成一圈，所有的排列数记为 Q_n^n。考虑其中已经排好的一圈，从不同位置断开，又变成不同的队列。所以有

$$\mathrm{Q}_n^n \times n = \mathrm{A}_n^n \Longrightarrow \mathrm{Q}_n = \frac{\mathrm{A}_n^n}{n} = (n-1)!$$

由此可知部分圆排列的公式：

$$\mathrm{Q}_n^r = \frac{\mathrm{A}_n^r}{r} = \frac{n!}{r \times (n-r)!}$$

组合数性质 | 二项式推论

$$\binom{n}{m} = \binom{n}{n-m}$$

相当于将选出的集合对全集取补集，故数值不变。（对称性）

$$\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$$

由定义导出的递推式。

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$
$$\binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$$

组合数的递推式（杨辉三角的公式表达）。我们可以利用这个式子，在 $O(n^2)$ 的复杂度下推导组合数。

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = \sum_{i=0}^{n} \binom{n}{i} = 2^n$$

这是二项式定理的特殊情况。取 $a = b = 1$ 就得到上式。

$$\sum_{i=0}^{n} (-1)^i \binom{n}{i} = [n = 0]$$

二项式定理的另一种特殊情况，可取 $a = 1, b = -1$。式子的特殊情况是取 $n = 0$ 时答案为 1。

$$\sum_{i=0}^{m} \binom{n}{i} \binom{m}{m-i} = \binom{m+n}{m} \quad (n \geq m)$$

拆组合数的式子，在处理某些数据结构题时会用到。

$$\sum_{i=0}^{n} \binom{n}{i}^2 = \binom{2n}{n}$$

这是 (6) 的特殊情况，取 $n = m$ 即可。

$$\sum_{i=0}^{n} i \binom{n}{i} = n2^{n-1}$$

带权和的一个式子，通过对 (3) 对应的多项式函数求导可以得证。

$$\sum_{i=0}^{n} i^2 \binom{n}{i} = n(n+1)2^{n-2}$$

与上式类似，可以通过对多项式函数求导证明。

$$\sum_{l=0}^{n} \binom{l}{k} = \binom{n+1}{k+1}$$

可以通过组合意义证明，在恒等式证明中较常用。

$$\binom{n}{r} \binom{r}{k} = \binom{n}{k} \binom{n-k}{r-k}$$

通过定义可以证明。

$$\sum_{i=0}^{n} \binom{n-i}{i} = F_{n+1}$$

其中 F 是斐波那契数列。

$$\sum_{l=0}^{n} \binom{l}{k} = \binom{n+1}{k+1}$$

通过组合分析——考虑 $S = a_1, a_2, \cdots, a_{n+1}$ 的 $k+1$ 子集数可以得证。

4.10.2 递推 + 预处理

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

适用场景 $n \le 2000, m \le 2000$ 询问次数 $t \le 1e5$ f[n][m] 就是答案

</> 代码 4.22: /数论/递推 + 预处理

```
ll f[2010][2010];
void Cinit() {
    for (int i = 0; i <= 2000; i++) {
        for (int j = 0; j <= i; j++) {
            if (!j) f[i][j] = 1;
            else f[i][j] = (f[i - 1][j - 1] + f[i - 1][j]) % mod;
        }
    }
}
```

4.10.3 逆元 + 预处理

$$\binom{n}{m} = \frac{n!}{(n-m)!m!} = (fac[n] * INV(fac[n-m]) * INV(fact[m])) \mod (prime)$$

适用场景 $n \le 2e5, m \le 2e5$ 询问次数 $t \le 1e5$, **模数为质数** C(a, b) 为答案

</> 代码 4.23: /数论/逆元 + 预处理

```
const int MAXN = 2e5 + 3;
ll fac[MAXN], inv[MAXN]; // 阶乘 阶乘的逆元
ll qp(ll a, ll b, ll mod) {
    ll res = 1, tmp = a % mod;
    while (b) {
        if (b & 1) res = (res * tmp) % mod;
        tmp = tmp * tmp % mod;
        b >>= 1;
    }
    return res;
}
void init(ll mod) {
    fac[0] = 1;
    for (int i = 1; i < MAXN; i++)
        fac[i] = fac[i - 1] * i % mod;
    inv[MAXN - 1] = qp(fac[MAXN - 1], mod - 2, mod);
    for (int i = MAXN - 2; i >= 0; i--)
        inv[i] = inv[i + 1] * (i + 1) % mod;
}
ll C(ll a, ll b) {
    if (b > a) return 0;
```

```
22      if (b == 0) return 1;
23      return fac[a] * inv[b] % mod * inv[a - b] % mod;
24  }
```

4.10.4 Lucas 定理

适用场景 $n, m \leq 1e18$ 模数为质数且 $p \leq 1e5$, 询问次数 $t \leq 20$ 复杂度 $O(p + \log_p m)$ lucas(a, b, p) 为答案

</> 代码 4.24: /数论/Lucas 定理

```
1   //卢卡斯定理,计算C(n,m) % p, 当n, m很大(1e18)且p比较小(1e5)的情况
2   //C(n, m) % p  =  C(n / p, m / p) * C(n%p, m%p) % p
3   ll qp(ll a, ll b, ll mod) {
4       ll res = 1, tmp = a % mod;
5       while (b) {
6           if (b & 1) res = (res * tmp) % mod;
7           tmp = tmp * tmp % mod;
8           b >>= 1;
9       }
10      return res;
11  }
12  ll C(ll a, ll b, ll p) { // 求组合数基础方法 O(n)
13      if (b > a) return 0;
14      ll res = 1;
15      for (ll i = 1, j = a; i <= b; i++, j--) {
16          res = (ll)res * j % p;
17          res = (ll)res * qp(i, p - 2, p) % p;
18      }
19      return res;
20  }
21  ll lucas(ll a, ll b, ll p) {
22      if (a < p && b < p) return C(a, b, p);
23      return (ll)C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
24  }
25  //例题 hdu5446
26  /*
27  给你三个数n, m, k
28  第二行是k个数, p1,p2,p3...pk
29  所有p的值不相同且p都是质数
30  求C(n, m) % (p1*p2*p3*...*pk)的值
31  范围: 1 m n 1e18, 1 k 10, pi 1e5, 保证p1*p2*p3*...*pk 1e18
32  我们知道题目要求C(n, m) % (p1*p2*p3*...*pk)的值
33  其实这个就是中国剩余定理最后算出结果后的最后一步求余
34  那C(n, m)相当于以前我们需要用中国剩余定理求的值
35  然而C(n, m)太大, 我们只好先算出
36  C(n, m) % p1 = r1
```

```
C(n, m) % p2 = r2
C(n, m) % p3 = r3
.
C(n, m) % pk = rk
用Lucas, 这些r1,r2,r3...rk可以算出来
然后又是用中国剩余定理求答案
 */
const int maxn = 110;
const int maxp = 1e5 + 10;
const int INF = 0x3f3f3f3f;
ll N,M,K;
ll F[maxp],Finv[maxp],inv[maxp];
void init(ll MOD){
    inv[1] = 1;
    for(int i = 2; i < maxp; i ++){
        inv[i] = (MOD - MOD / i) * 1ll * inv[MOD % i] % MOD;
    }
    F[0] = Finv[0] = 1;
    for(int i = 1; i < maxp; i ++){
        F[i] = F[i - 1] * 1ll * i % MOD;
        Finv[i] = Finv[i - 1] * 1ll * inv[i] % MOD;
    }
}
ll quick_power(ll a,ll b,ll p){
    ll ans = 1;
    while(b){
        if(b & 1) ans = ans * a % p;
        b >>= 1;
        a = a * a % p;
    }
    return ans;
}
void ex_gcd(ll a, ll b, ll &x, ll &y, ll &d){
    if (!b) {d = a, x = 1, y = 0;}
    else{
        ex_gcd(b, a % b, y, x, d);
        y -= x * (a / b);
    }
}
ll Inv(ll t, ll p){//如果不存在, 返回-1
    ll d, x, y;
    ex_gcd(t, p, x, y, d);
    return d == 1 ? (x % p + p) % p : -1;
}
```

```cpp
ll fact(int n, ll p){//n的阶乘求余p
    ll ret = 1;
     for (int i = 1; i <= n ; i ++) ret = ret * i % p ;
    return ret ;
}
ll comb(int n,int m,ll p){
    if(m < 0 || m > n) return 0;
    return fact(n,p) * Inv(fact(m,p),p) % p * Inv(fact(n - m,p),p) % p;
}
ll Lucas(ll n,ll m,ll p){
    return m?Lucas(n / p,m / p,p) * comb(n % p,m % p,p) % p : 1;
}

ll mul(ll a,ll b,ll p){
    ll ans = 0;
    while(b){
        if(b & 1) ans = (ans + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return ans;
}
ll china(int n,ll *a,ll *m){
    ll M = 1,ret = 0;
    for(int i = 1; i <= n; i ++) M *= m[i];
    for(int i = 1; i <= n; i ++){
        ll w = M / m[i];
        ret = (ret + mul(w * Inv(w,m[i]),a[i],M)) % M;
    }
    return (ret + M) % M;
}
ll a[maxn],b[maxn];
int main(){
    int T; Sca(T);
    while(T--){
        scanf("%lld%lld%lld",&N,&M,&K);
        for(int i = 1; i <= K ; i ++){
            Scl(a[i]);
            b[i] = Lucas(N,M,a[i]);
        }
        Prl(china(K,b,a));
    }
    return 0;
}
```

4.10.5 高精度求组合数

对组合数公式的分子分母都有的因子进行处理，最后得到组合数为各个质数的幂次的乘积

$$C_n^m = p_1^{a1} \times p_2^{a2} \times p_3^{a3} \times \cdots \times p_n^{an}$$

适用场景 $n, m \leq 5000$

</> 代码 4.25: /数论/高精度求组合数

```
#include <iostream>
#include <vector>
using namespace std;
const int N = 5010;
int sum[N]; //每一个质数的次数
int primes[N], cnt;
bool st[N];
void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] * i <= n; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
vector<int> mul(vector<int> a, int b) {
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i++) {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while (t) { c.push_back(t % 10); t /= 10; }
    return c;
}
int get(int n, int p) {
    int res = 0;
    while (n) {
        res += n / p;
        n /= p;
    }
    return res;
}
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
```

```
39      get_primes(n);
40      //遍历每一个质数,求每个质数的次数
41      for (int i = 0; i < cnt; i++) {
42          int p = primes[i];
43          sum[i] = get(n, p) - get(n - m, p) - get(m, p); //约掉因子
44      }
45      vector<int> res;
46      res.push_back(1);
47      for (int i = 0; i < cnt; i++)         //遍历每一个质数因子
48          for (int j = 0; j < sum[i]; j++) //开始乘以它的个数
49              res = mul(res, primes[i]);
50      for (int i = res.size() - 1; i >= 0; i--)
51          printf("%d", res[i]);
52      return 0;
53  }
```

4.10.6 卡特兰数

递归公式 1

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

递归公式 2

$$f(n) = \frac{f(n-1) * (4 * n - 2)}{n+1}$$

组合公式 1

$$f(n) = \frac{C_{2n}^{n}}{n+1}$$

组合公式 2!!!

$$f(n) = C_{2n}^{n} - C_{2*n}^{n-1}$$

递推公式

$$f[n] = \sum_{i=0}^{n-1} f[i] * f[n-i-1]$$

</> 代码 4.26: /数论/卡特兰数

```
1  /*卡特兰数
2  1.前30项
3  [1,1,2,5,14,42,132,429,1430,4862,16796,58786,
4   208012,742900,2674440,9694845,35357670,129644790,
5   477638700,1767263190,6564120420,24466267020,
6   91482563640,343059613650,1289904147324,
7   4861946401452,18367353072152,69533550916004,
8   263747951750360,1002242216651368,3814986502092304]
9  2.在比赛的时候很可能不能证明一个数列是卡特兰数, 一般手算几项发现对上了卡特兰数就冲了
10 */
11 //卡特兰数取模模板
```

```
ll Catalan[maxn],inv[maxn];
inline void Catalan_init(int n,ll mod){
    inv[1] = 1;
    for(int i = 2; i <= n + 1; i ++){
        inv[i] = (mod - mod / i) * inv[mod % i] % mod;
    }
    Catalan[0] = Catalan[1] = 1;
    for(int i = 2; i <= n ; i ++){
        Catalan[i] = Catalan[i - 1] * (4 * i - 2) % mod * inv[i + 1] % mod;
    }
}
int main(){
    Catalan_init(10000,mod);
    while(~Sca(N)) Prl(Catalan[N]);
    return 0;
}
//卡特兰数大数模板
#include<bits/stdc++.h>
using namespace std;
const int C_maxn = 100 + 10;///项数
int Catalan_Num[C_maxn][1000];///保存卡特兰大数、第二维为具体每个数位的值
int NumLen[C_maxn];///每个大数的数长度、输出的时候需倒序输出
void catalan() {//求卡特兰数

    int i, j, len, carry, temp;
    Catalan_Num[1][0] = NumLen[1] = 1;
    len = 1;
    for(i = 2; i < 100; i++){
        for(j = 0; j < len; j++) //乘法
        Catalan_Num[i][j] = Catalan_Num[i-1][j]*(4*(i-1)+2);
        carry = 0;
        for(j = 0; j < len; j++) {//处理相乘结果

            temp = Catalan_Num[i][j] + carry;
            Catalan_Num[i][j] = temp % 10;
            carry = temp / 10;
        }
        while(carry) {//进位处理

            Catalan_Num[i][len++] = carry % 10;
            carry /= 10;
        }
        carry = 0;
        for(j = len-1; j >= 0; j--) {//除法
```

```
            temp = carry*10 + Catalan_Num[i][j];
            Catalan_Num[i][j] = temp/(i+1);
            carry = temp%(i+1);
        }
        while(!Catalan_Num[i][len-1]) //高位零处理
        len --;
        NumLen[i] = len;
    }
}
int main(void){
    catalan();
    for(int i=1; i<=30; i++){
        for(int j=NumLen[i]-1; j>=0; j--){
            printf("%d", Catalan_Num[i][j]);
        }puts("");
    }
    return 0;
}
```

4.10.7 康拓展开

康托展开可以用来求一个 $1 \sim n$ 的任意排列的排名。康托展开可以在 $O(n^2)$ 的复杂度内求出一个排列的排名，在用到树状数组优化时可以做到 $O(n \log n)$。

$$X = a_n(n-1)! + a_{n-1}(n-2) + \cdots + a_1 \times 0!$$

</> 代码 4.27: /数论/康拓展开

```
//如果要求当前全排列是第几个排列，则需要把数字 + 1
ll cantor(int a[], int n) {
    ll res = 0; // 树状数组优化
    for(int i = n; i >= 1; i --) {
        int smaller = sum(a[i]); // 在当前位之后小于其的个数
        add(a[i], 1);            // 改为 大于 则逆序
        res = (res + fac[n - i] * smaller % mod) % mod;
    }  // fac[] 为预处理的阶乘数组
    return (res + 1) % mod;
}
```

4.10.8 逆康拓展开

由于康托展开是双射，所以可逆，从而通过排名求原组合，可以用线段树维护区间第 t 小值来优化到 $O(nlog(n))$

</> 代码 4.28: /数论/逆康拓展开

```
vector<int> decantor(int x, int n) {
    x --;
```

```
    vector<int> num(n + 1, 0), res(n + 1);
    for (int i = 1; i <= n; i++) {
        int t = x / fac[n - i], j;
        for(j = 1; i <= n; j ++)
            if(!num[j]) {
                if(!t) break;
                t --;
            }
        res[i] = j;
        num[j] = 1;
        x %= fac[n - i];
    }
    return res; // res 的 [1 ~ n] 为答案
}
```

4.11 0/1 分数规划

如一个物品有成本 bi 和价值 ai 的时候, 我们需要一种方案使得取 K 的物品之后价值和成本的比例最高

$$R = \sum \frac{a_i * x_i}{b_i * x_i}$$

形如式子中选定每个 xi 的值为 0/1, 最终使得 R 最大
解: 二分出答案 t 按来代入式子取 check, 每次选取 ai - bi * t 最大的 K 个物品, 如果最终 ai - bi * t >= 0, 说明答案 t 可行

4.12 SG 函数

</> 代码 4.29: /数论/SG 函数

```
/* SG函数
必败态: 当前状态不管怎么选择, 只要对手采取最优策略就败
必胜态: 当前状态只要你采取最优策略就胜
1.当某个局面导向的所有状态都是必胜态时候, 当前状态必败
2.当某个局面导向有一个状态是必败的时候, 当前状态必胜
3.SG(x) = mex{SG(y1),SG(y2),SG(y3)...SG(yk)} 其中y1..yk是x导向的所有状态
当SG(x) > 0时说明当前状态必胜, SG(x) < 0时说明当前状态必败
4.多个有向图游戏的和SG函数值等于他包含的各个子游戏的异或和
SG(G) = SG(G1) ^ SG(G2) ^ SG(G3)..... ^ SG(Gm)
SG可以通过递推或者记忆化搜索等方式求出
*/
//模板:总共n堆石子, 每个人只能抓2的幂次(1,2,4,8...)个石子, 轮流抓取抓不到石头的输
const int maxn = 1010;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
```

```cpp
int sg[maxn];
int step[maxn];  //step存储可以走的步数，step[0]存储有多少种走法
bool vis[maxn];
void getsg(){   //直接递推
    sg[0] = 0;
    for(int i = 1; i <= 1000; i ++){
        for(int j = 0 ; j <= i ; j ++) vis[j] = 0;
        for(int j = 1; j <= step[0]; j ++){
            if(step[j] > i) break; //step需要从小到大排序
            vis[sg[i - step[j]]] = 1;
        }
        for(int j = 0; j <= i ; j ++){
            if(vis[j]) continue;
            sg[i] = j; break;
        }
    }
}

int main(){
    step[0] = 10; step[1] = 1;
    for(int i = 2; i <= 10 ; i ++) step[i] = step[i - 1] * 2;
    getsg();
    while(~Sca(N)){
        if(sg[N]) puts("Kiki");  //先手胜
        else puts("Cici");   //后手胜
    }
    return 0;
}
```

4.13 bsgs

</> 代码 4.30: /数论/bsgs

```cpp
//baby step,giant step 算法
//给定整数a,b,p,其中a,p互质，求一个非负整数x，使得a ^ x  b (mod p)
LL bsgs(LL a,LL b,LL p){
    map<LL,LL> hash; hash.clear();
    b %= p;
    int t = (int)sqrt(p) + 1;
    for(int j = 0 ; j < t; j ++){
        int val = (LL)b * quick_power(a,j,p) % p;
        hash[val] = j;
    }
    a = quick_power(a,t,p);
```

```
12      if(a == 0) return b == 0?1:-1;
13      for(int i = 0 ; i <= t; i ++){
14          int val = quick_power(a,i,p);
15          int j = hash.find(val) == hash.end() ? -1:hash[val];
16          if(j >= 0 && i * t - j >= 0) return i * t - j;
17      }
18      return -1;
19  }
```

4.14 母函数

</> 代码 4.31: /数论/母函数

```
1   //母函数
2   /*
3   求用1分、2分、3分的邮票贴出不同数值的方案数: (每张邮票的数量是无限的)
4   那么
5   1分: (1+x^1+x^2+x^3+x^4+......)
6   2分: (1+x^2+x^4+x^6+x^8+......)
7   3分: (1+x^3+x^6+x^9+x^12+......)
8   然后这3个乘起来
9   对于这种无限的, 计算到最大的范围就可以了
10   */
11   #include<cstdio>
12  typedef long long LL;
13  const int maxn = 100 + 5;//假如题目只问到100为止
14  const int MAX = 3;//题目只有1,2,3这3种邮票
15  LL c1[maxn], c2[maxn];//c2是临时合并的多项式, c1是最终合并的多项式
16  int n;
17  void init(){
18      c1[0] = 1;//一开始0的情况算一种
19      for(int i = 1; i <= MAX; i ++){//把1分到MAXN的邮票合并, 变成一个多项式
20          for(int j = 0; j < maxn; j += i){//i分的邮票, 步长是i
21              for(int k = 0; j + k < maxn; k ++){//从x^0到x^N遍历一遍
22                  c2[j + k] += c1[k];//因为j的所有项系数为1, 所以c1[k]可以看成c1[k]*1;
23              }
24          }
25          for(int j = 0; j < maxn; j ++){//把c2的数据抄到c1, 清空c2
26              c1[j] = c2[j];
27              c2[j] = 0;
28          }
29      }
30  }
31  int main(){
```

```
    init();
    while(scanf("%d", &n) != EOF){
        printf("%I64d\n", c1[n]);
    }
}
//样例2，问一个数字n能够拆成多少种数字的和
/* 比如n = 4
比如n=4
4 = 4;4 = 3 + 1;
4 = 2 + 2;4 = 2 + 1 + 1;
4 = 1 + 1 + 1 + 1;
有5种，那么答案就是5*/
//相当于把所有数当成邮票看能组成多少
const int maxn = 130;
const int MAX = 120;
int N,M,K;
LL c1[maxn],c2[maxn];
void init(){
    c1[0] = 1;
    for(int i = 1; i <= MAX; i ++){
        for(int j = 0 ; j < maxn; j += i){
            for(int k = 0 ; j + k < maxn; k ++){
                c2[j + k] += c1[k];
            }
        }
        for(int j = 0 ; j < maxn; j ++){
            c1[j] = c2[j];
            c2[j] = 0;
        }
    }
}
int main(){
    init();
    while(~Sca(N)) Prl(c1[N]);
    return 0;
}
```

4.15 线性空间

</> 代码 4.32: /数论/线性空间

```
/*
线性空间
定义
```

```
线性空间是一个关于一下两个运算封闭的向量集合:
1.向量加法a+b，其中a,b为向量
2.标量乘法k*a，其中a为向量，k为常数
基础概念
1.给定若干个向量a1,a2,...,an，若向量b能够通过a1,a2,...,an经过向量加法和标量乘法得到，则称向量b能够通过a1,a2,...,an表出。
2.对于向量a1,a2,...,an能够表出的所有向量所构成了一个线性空间，称a1,a2,...,an为这个线性空间的生成子集。
3.在一个线性空间中选取若干个向量，若一个向量能够被其他向量表出，则称这些向量线性相关，反之，称这些向量线性无关。
4.线性无关的生成子集成为线性空间的基底，简称基。
4 .另外地，线性空间的极大线性无关子集也称为线性空间的基底，简称基。
5.一个线性空间的所有基包含的向量个数都相等，这个个数称为线性空间的维数。

高斯消元和线性基
对于一个n*m的矩阵，我们可以把它看做n个长度为m的向量，即它的维数为m。那么如果将这个矩阵看做系数矩阵进行高斯消元，则消元后得到的矩阵中所有非0行代表的向量线性无关。
因为初等行变换就是对每行的向量之间进行向量加法和标量乘法，所以高斯消元的操作并不改变n个向量所能表出的线性空间，则消元后的矩阵的所有非0行所代表的向量就是原线性空间的一个基。
综上所述，我们可以利用高斯消元算法对若干个向量求解其构成线性空间的一个基。
 */
//BZOJ4004
//每个向量有一个权值，求给定矩阵的秩和组成这些秩的向量的最小权值和
//这次不需要解线性方程组了，高斯消元的目的是化简矩阵
const int maxn = 510;
const double eps = 1e-5;  //这个精度可能要调，看AC情况
int N,M,K;

long double MAP[maxn][maxn]; //long double提升精度
int c[maxn];
void gauss(){
    int num = 0,ans = 0;
    for(int i = 1; i <= min(N,M) ; i ++){
        int r = i;
        for(int j = i + 1; j <= N ; j ++){
            if(fabs(MAP[j][i]) < eps) continue;
            if(fabs(MAP[r][i]) < eps || c[r] > c[j]) r = j;
        } //每次选取代价最小的基底
        if(fabs(MAP[r][i]) < eps) continue;
        num++; ans += c[r];
        swap(MAP[r],MAP[i]); swap(c[r],c[i]);
        long double div = MAP[i][i];
        for(int j = i; j <= M; j ++) MAP[i][j] /= div;
        for(int j = i + 1; j <= N ; j ++){
```

```
            div = MAP[j][i];
            for(int k = i ; k <= M; k ++) MAP[j][k] -= MAP[i][k] * div;
        }
    }
    printf("%d %d",num,ans);
}
int main(){
    Sca2(N,M);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            cin >> MAP[i][j];
        }
    }
    for(int i = 1; i <= N; i ++) Sca(c[i]);
    gauss();
    return 0;
}
```

4.16 线性基

</> 代码 4.33: /数论/线性基

```
/*    线性基: 处理许多数之间的异或问题
对于一个数集V, 它的线性基 是它的一个子集, 满足 中所有数互相异或得到的集合等价于V中所有数互相异或得到的集合。也就是说, 可以看成是V的压缩。
线性基有一些性质:
1.线性基的异或集合中不存在0。也就是说, 是V中线性无关的极大子集。(这些概念以后再补吧。。)
2.线性基中每个元素的异或方案唯一, 也就是说, 线性基中不同的异或组合异或出的数都是不一样的。这个与性质1其实是等价的。
3.线性基的二进制最高位互不相同。
4.如果线性基是满的, 它的异或集合为[1,2n-1]。
5.线性基中元素互相异或, 异或集合不变。
*/
const int maxn = 110;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
LL a[maxn],p[100],d[100];
//建立线性基 (插入操作)
inline void add(LL x){
    for(int i = 62; i >= 0; i --){
        if(!(x & (1LL << i))) continue; //注意1LL
        if(!p[i]){p[i] = x;break;}      //注意break
        x ^= p[i];                      //注意是x变动
```

```
    }
}
//查询异或最小值：线性基中 位数最低的数字
//查询异或最大值
inline LL getmax(){
    LL ans = 0;
    for(LL i = 62; i >= 0; i --){
        if(ans < (ans ^ p[i])) ans ^= p[i];
    }
    return ans;
}
//check是否存在子集合可以异或为这个数
bool check(LL x){  // true存在,false不存在
    for(int i = 62; i >= 0 ; i --){
        if(!(x & (1LL << i))) continue;
        if(!p[i]) return false;
        x ^= p[i];
    }
    return x == 0;
}
//查询集合所有可异或数中K小的数
//先将线性基rebuild, 然后可以O(logn)的查找
void rebuild(){
    for(LL i = 62; i >= 0 ; i --){
        for(LL j = i - 1; j >= 0; j --){
            if(p[i] & (1LL << j)) p[i] ^= p[j];
        }
    }
    for(int i = 0 ; i <= 62; i ++) if(p[i]) d[cnt++] = p[i];
    return ;
}
LL query(LL k){
    if(!k) return 0;
    if(k >= (1LL << cnt)) return -1;
    LL ans = 0;
    for(int i = 62; i >= 0; i --){
        if(k & (1LL << i)) ans ^= d[i];
    }
    return ans;
}
//main函数：查找线性基异或数的K小
int main(){
    int T = read();
    int CASE = 1;
```

```
65      while(T--){
66          for(int i = 0 ; i <= 62; i ++) p[i] = 0;
67          Sca(N);
68          for(int i = 1; i <= N ; i ++){
69              Scl(a[i]); add(a[i]);
70          }
71          cnt = 0; rebuild();
72          Sca(Q);
73          int flag = (cnt != N);
74          for(int i = 1; i <= Q; i ++){
75              LL k; scanf("%lld",&k);
76              Prl(query(k - flag));
77          }
78      }
79      return 0;
80  }
81
82  //例题: hdu6579
83  /*给出一个序列.
84  操作1在序列尾增加一个数 操作2.求区间异或最大值
85  做法:维护每个前缀的线性基(上三角形态)
86  对于每个线性基，讲出现为止靠右的数字尽可能放在高位，也就是说在插入新的数字的时候，要同时记录
        对应位置上数字出现位置，
87  并且再找到可以插入的位置的时候，如果新数字比原来位置上的数字更靠右，就将该位置上原来的数字向
        地位推。
88  在求最大值的时候，从高位向低位遍历，如果该位上数字出现在询问中左端点的右侧且可以使答案变大，
        就异或进答案。
89  对于线性基的每一位，与它异或过的线性基更高位置上的数字肯定都出现在它优策
90   */
91  const int maxn = 1e6 + 10;
92  int N,M,K;
93  struct xj{
94      int a[32],pos[32];
95      inline void init(){
96          for(int i = 0; i <= 30; i ++) a[i] = pos[i] = 0;
97      }
98      inline void add(int x,int p){
99          for(int i = 30; i >= 0 ; i --){
100             if(!(x & (1 << i))) continue;
101             if(!a[i]){
102                 a[i] = x; pos[i] = p;
103                 break;
104             }else{
105                 if(p > pos[i])swap(a[i],x),swap(p,pos[i]);
```

```
106                 x ^= a[i];
107             }
108         }
109     }
110     inline int getmax(int l){
111         int ans = 0;
112         for(int i = 30; i >= 0; i --){
113             if(pos[i] < l) continue;
114             if(ans < (ans ^ a[i])) ans ^= a[i];
115         }
116         return ans;
117     }
118 }pre[maxn];
119 int main(){
120     int T; Sca(T);
121     while(T--){
122         Sca2(N,M);
123         for(int i = 1; i <= N ; i ++){
124             int x = read();
125             pre[i] = pre[i - 1];
126             pre[i].add(x,i);
127         }
128         int ans = 0;
129         while(M--){
130             int op = read();
131             if(!op){
132                 int l = read() ^ ans,r = read() ^ ans;
133                 l = l % N + 1; r = r % N + 1;
134                 if(l > r) swap(l,r);
135                 ans = pre[r].getmax(l);
136                 Pri(ans);
137             }else{
138                 int x = read() ^ ans;
139                 N++;
140                 pre[N] = pre[N - 1];
141                 pre[N].add(x,N);
142             }
143         }
144     }
145     return 0;
146 }
```

4.17 拉格朗日插值

4.17.1 拉格朗日插值

$$f(k) = \sum_{i=0}^{n} y_i \prod_{i!=j} \frac{k - x[j]}{x[i] - x[j]}$$

</> 代码 4.34: /数论/拉格朗日插值

```
/*拉格朗日插值
众所周知，n + 1个x坐标不同的点可以确定唯一的最高为n次的多项式。在算法竞赛中，我们常常会碰到一类题目，题目中直接或间接的给出了n+1个点，让我们求由这些点构成的多项式在某一位置的取值
一个最显然的思路就是直接高斯消元求出多项式的系数，但是这样做复杂度巨大(n^3)且根据算法实现不同往往会存在精度问题
而拉格朗日插值法可以在n^2的复杂度内完美解决上述问题
假设该多项式为f(x)，第i个点的坐标为(xi,yi)，我们需要找到该多项式在k点的取值
根据拉格朗日公式可以直接计算
 */
 const int maxn = 2010;
const int INF = 0x3f3f3f3f;
const LL mod = 998244353;
int N,M,K;
LL x[maxn],y[maxn];
LL quick_power(LL a,LL b,LL Mod){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % Mod;
        a = a * a % Mod;
        b >>= 1;
    }
    return ans;
}
LL inv(LL a,LL Mod){
    return quick_power(a,Mod - 2,Mod);
}
LL v;
int main(){
    Sca(N); Scl(v);
    for(int i = 1; i <= N ; i ++) scanf("%lld%lld",&x[i],&y[i]);
    LL ans = 0;
    for(int i = 1; i <= N ; i ++){
        LL sum = 1;
        for(int j = 1; j <= N ; j ++){
            if(i != j) sum = sum * (x[i] + mod - x[j]) % mod;
        }
        sum = inv(sum,mod);
```

```
36          sum = sum * y[i] % mod;
37          for(int j = 1; j <= N ; j ++){
38              if(i != j) sum = sum * (v - x[j] + mod) % mod;
39          }
40          ans = (ans + sum) % mod;
41      }
42      Prl(ans);
43      return 0;
44  }
```

4.17.2 在 x 取值连续时的做值

$$pre_i = \prod_{j=0}^{i} k - j$$

$$suf_i = \prod_{j=i}^{n} k - j$$

$$f(k) = \sum_{i=0}^{n} y_i \frac{pre_{i-1} * suf_{i+1}}{fac[i] * fac[n-i]}$$

4.17.3 重心拉格朗日插值法

$$g = \prod_{i=1}^{n} k - x[i]$$

$$t_i = \frac{y_i}{\prod_{j \neq i} x_i - x_j}$$

$$f(k) = g \sum_{i=0}^{n} \frac{t_i}{(k - x[i])}$$

</> 代码 4.35: /数论/重心拉格朗日插值法

```
1  /*重心拉格朗日插值法
2  求原函数复杂度为O(n^2),单次求值复杂度O(n^2)。
3  动态加点？
4  每次增加一个点，一般求法会重新计算一次达到O(n2^)的复杂度，而重心拉格朗日插值法只需O(n)计算wi
       即可。
5   */
6   /*模板：tyvj1858
7  f(i)=1^k+2^k+3^k+......+i^k
8  g(x)=f(1)+f(2)+f(3)+....+f(x)
9  求(g(a)+g(a+d)+g(a+2d)+......+g(a+nd))mod p
10 解：可知f(i)是k + 1次多项式,g(x)是k + 2次多项式,h(i)是k + 3次多项式
11   */
12 const int maxn = 210;
13 const LL mod = 1234567891;
```

```
LL K,a,n,d,p;
inline LL mul(LL a,LL b){return (a % mod * b % mod + mod) % mod;}
inline LL add(LL a,LL b){return ((a + b) % mod + mod) % mod;}
LL quick_power(LL a,LL b){
    LL ans = 1;
    while(b){
        if(b & 1) ans = mul(ans,a);
        b >>= 1;
        a = mul(a,a);
    }
    return ans;
}
LL inv(LL a){
    return quick_power(a,mod - 2);
}
LL get(LL *t,LL *x,LL *y,LL Lim,LL n){
    LL g = 1;
    for(int i = 1; i <= Lim; i ++) if(n == x[i]) return y[i];
    for(int i = 1; i <= Lim; i ++) g = mul(g,add(n,-x[i]));
    LL ans = 0;
    for(int i = 1; i <= Lim ; i ++) ans = add(ans,mul(t[i],inv(add(n,-x[i]))));
    ans = mul(ans,g);
    return ans;
}
LL f[maxn],g[maxn],h[maxn],t[maxn];
LL x[maxn];
int main(){
    int T; Sca(T);
    while(T--){
        scanf("%lld%lld%lld%lld",&K,&a,&n,&d);
        for(int i = 1; i <= 130; i ++) f[i] = add(f[i - 1],quick_power(i,K));
        for(int i = 1; i <= 130; i ++) g[i] = add(g[i - 1],f[i]);
        for(int i = 1; i <= K + 3; i ++) x[i] = i;
        for(int i = 1; i <= K + 3; i ++){
            t[i] = 1;
            for(int j = 1 ; j <= K + 3; j ++){
                if(i == j) continue;
                t[i] = mul(t[i],x[i] - x[j]);
            }
            t[i] = mul(g[i],inv(t[i]));
        }
        for(int i = 1; i <= K + 4; i ++) h[i] = add(h[i - 1],get(t,x,g,K + 3,add(a,mul(i
    - 1,d))));
        for(int i = 1; i <= K + 4; i ++) x[i] = add(a,mul(i - 1,d));
```

```
57          for(int i = 1; i <= K + 4; i ++){
58              t[i] = 1;
59              for(int j = 1; j <= K + 4; j ++){
60                  if(i == j) continue;
61                  t[i] = mul(t[i],x[i] - x[j]);
62              }
63              t[i] = mul(h[i],inv(t[i]));
64          }
65          Prl(get(t,x,h,K + 4,add(a,mul(n,d))));
66      }
67      return 0;
68  }
```

4.17.4 自然数连续幂次和

</> 代码 4.36: /数论/自然数连续幂次和

```
1   //Σi ^ k
2   //1^k+2^k+3^k+...n^k
3   /*可以被证明，结果是一个n为自变量的k + 1次多项式
4   因此我们将n = (0,1,2,3....,k,k + 1)代入算出函数值
5   将这0 ~ k + 1所有的点用拉格朗日插值插入
6   求解的时候就将n作为下标代入就可以了。
7   可以用上面在x取值连续时的做法优化，时间复杂度k ^ 2
8    */
9    //luoguP4593 模板为函数getsum
10  const int maxn = 70;
11  const int mod = 1e9 + 7;
12  LL N,M,K;
13  LL fac[maxn],ifac[maxn]; //阶乘，阶乘的逆元
14  LL mul(LL a,LL b){
15      return a % mod * b % mod;
16  }
17  LL add(LL a,LL b){
18      return ((a + b) % mod + mod) % mod;
19  }
20  LL quick_power(LL a,LL b){
21      LL ans = 1;
22      while(b){
23          if(b & 1) ans = mul(ans,a);
24          a = mul(a,a);
25          b >>= 1;
26      }
27      return ans;
28  }
```

```
LL inv(LL x){
    return quick_power(x,mod - 2);
}
void init(){
    fac[0] = 1;
    for(int i = 1; i <= 60 ; i ++) fac[i] = mul(i,fac[i - 1]);
    ifac[60] = inv(fac[60]);
    for(int i = 59; i >= 0; i --) ifac[i] = mul(i + 1,ifac[i + 1]);
}
LL a[maxn],pre[maxn],suf[maxn],y[maxn];
LL getsum(LL n,LL k){ // 1 ^ k + 2 ^ k + ... + n ^ k
    LL Lim = k + 1,ans = 0; y[0] = 0;
    for(int i = 1; i <= Lim; i ++) y[i] = add(y[i - 1],quick_power(i,k));
    pre[0] = n; suf[Lim + 1] = 1;
    for(int i = 1; i <= Lim; i ++) pre[i] = mul(pre[i - 1],add(n,-i));
    for(int i = Lim; i >= 1; i --) suf[i] = mul(suf[i + 1],add(n,-i));
    for(int i = 1 ; i <= Lim; i ++){
        LL up = mul(y[i],mul(pre[i - 1],suf[i + 1]));
        LL down = mul(ifac[i],ifac[Lim - i]);
        if((Lim - i) & 1) down = mod - down;
        ans = add(ans,mul(down,up));
    }
    return ans;
}
int main(){
    int T; Sca(T); init();
    while(T--){
        scanf("%lld%lld",&N,&M); K = M + 1;
        for(int i = 1; i <= M; i ++) Scl(a[i]);
        a[++M] = ++N;
        sort(a + 1,a + 1 + M);
        LL ans = 0;
        for(int i = 1; i <= M ; i ++){
            for(int j = i ; j <= M ; j ++) ans = add(ans,add(getsum(a[j] - 1,K),-getsum(
    a[j - 1],K)));
            for(int j = i + 1; j <= M ; j ++) a[j] = add(a[j],-a[i]);
            a[i] = 0;
        }
        Prl(ans);
    }
    return 0;
}
```

4.18 FFT

4.18.1 卷积

</> 代码 4.37: /数论/fft 卷积

```
const double PI = acos(-1.0);
const int MN = 4e6 + 10;
struct Comp {
    double x, y;
    Comp(double xx = 0, double yy = 0) {
        x = xx, y = yy;
    }
    Comp operator+(Comp c) {
        return Comp(x + c.x, y + c.y);
    }
    Comp operator-(Comp c) {
        return Comp(x - c.x, y - c.y);
    }
    Comp operator*(Comp c) {
        double tx = x * c.x - y * c.y;
        double ty = x * c.y + y * c.x;
        return Comp(tx, ty);
    }
};
Comp ff[MN];
int rev[MN];
void FFT(Comp *f, int n, int type) {
    for (int i = 0; i < n; ++i) {
        if (i < rev[i]) {
            swap(f[i], f[rev[i]]);
        }
    }
    for (int h = 2; h <= n; h <<= 1) {
        Comp step(cos(2 * PI / h), sin(2 * PI * type / h));
        for (int j = 0; j < n; j += h) {
            Comp cur(1, 0);
            for (int k = j; k < j + h / 2; k++) {
                Comp f1 = f[k], f2 = f[k + h / 2];
                f[k] = f1 + cur * f2;
                f[k + h / 2] = f1 - cur * f2;
                cur = cur * step;
            }
        }
    }
    if (type == 1)
```

```
41          return;
42      for (int i = 0; i < n; i++)
43          f[i].x /= n, f[i].y /= n;
44  }
45  int main() {
46      int n = rr(), m = rr();
47      for (int i = 0; i <= n; i++)
48          ff[i].x = rr();
49      for (int i = 0; i <= m; i++)
50          ff[i].y = rr();
51      int lim = 1, lim_2;
52      while (lim <= n + m)
53          lim <<= 1;
54      lim_2 = lim >> 1;
55      for (int i = 0; i < lim; ++i) {
56          rev[i] = (rev[i >> 1] >> 1) | ((i & 1) * lim_2);
57      }
58      FFT(ff, lim, 1);
59      for (int i = 0; i <= lim; i++)
60          ff[i] = ff[i] * ff[i];
61      FFT(ff, lim, -1);
62      for (int i = 0; i <= m + n; i++)
63          printf("%d ", int(ff[i].y / 2 + 0.5));
64      return 0;
65  }
```

4.18.2 递归法

</> 代码 4.38: /数论/fft 递归法

```
1  /* fft 快速傅里叶变换
2  用于加速多项式的乘法,形如
3  给定一个n次多项式F(x), 和一个m次多项式G(x)。
4  求出F(x)和G(x)的卷积的题目,常规做法是O(n ^ 2)将系数遍历相乘。
5  fft先将两个多项式nlogn转化为点值表示法, 然后O(n)相乘,最后nlogn转化为系数表示法
6  注: 这里的卷积即两个多项式自然相乘
7   */
8   //递归法:常数巨大, 容易爆栈, 一般情况下用迭代法
9  const int maxn = 1e6 + 10;
10 const double PI = acos(-1.0);
11 int N,M,K;
12 struct complex{
13     double x,y;
14     complex(){}
15     complex(double x,double y):x(x),y(y){}
```

```
16      friend complex operator - (complex a,complex b){return complex(a.x - b.x,a.y - b.y)
        ;}
17      friend complex operator + (complex a,complex b){return complex(a.x + b.x,a.y + b.y)
        ;}
18      friend complex operator * (complex a,complex b){return complex(a.x * b.x - a.y * b.y
        ,a.x * b.y + a.y * b.x);}
19  }a[maxn << 2],b[maxn << 2]; //最坏情况下是4倍
20  void FFT(int limit,complex *a,int type){  //limit一定是2的幂次，这样保证每次可以对半分
21      if(limit == 1) return;
22      complex a1[limit >> 1],a2[limit >> 1]; //分出奇偶位置上的数
23      for(int i = 0 ; i < limit; i += 2){
24          a1[i >> 1] = a[i];
25          a2[i >> 1] = a[i + 1];
26      }
27      FFT(limit >> 1,a1,type);
28      FFT(limit >> 1,a2,type);
29      complex Wn = complex(cos(2 * PI / limit),type * sin(2 * PI / limit)),w = complex
        (1,0);
30      limit >>= 1;    //Wn为单位元,w为幂次
31      for(int i = 0 ; i < limit; i ++, w = w * Wn){
32          a[i] = a1[i] + w * a2[i];
33          a[i + limit] = a1[i] - w * a2[i];
34      }
35  }
36  int main(){
37      Sca2(N,M);
38      for(int i = 0; i <= N ; i ++){a[i].y = 0; scanf("%lf",&a[i].x);}
39      for(int j = 0; j <= M ; j ++){b[j].y = 0; scanf("%lf",&b[j].x);}
40      int limit = 1;
41      while(limit <= N + M) limit <<= 1;//使用的是0~limit - 1这几个数
42      FFT(limit,a,1); FFT(limit,b,1);   //系数转化为点值表示
43      for(int i = 0; i < limit ; i ++) a[i] = a[i] * b[i];  //点值相乘只要O(n)
44      FFT(limit,a,-1);                  //点值转化回系数表示
45      for(int i = 0 ; i <= N + M; i ++){
46          printf("%d ",(int)(a[i].x / limit + 0.5));  //输出系数
47      }
48      return 0;
49  }
```

4.18.3 迭代法

</> 代码 4.39: /数论/fft 迭代法

```
1  //迭代法,发现原位置下标和目标位置下表是二进制下翻转的关系
2  //因此可以预处理好位置优化，不用递归实现，常数较小较安全
```

```
3  //FFT小贴士:1.一定要看清题目需求决定答案要不要/2,例如只想求a,b组合出的情况，不除2的情况下系
       数同时表示a和b组合以及b和a组合
4  //2.如果是自身对自身fft一定要处理好相同位置相乘的情况
5  const int maxn = 1e6 + 10;
6  const double PI = acos(-1.0);
7  int N,M,K;
8  struct complex{
9      double x,y;
10     complex(){}
11     complex(double x,double y):x(x),y(y){}
12     friend complex operator - (complex a,complex b){return complex(a.x - b.x,a.y - b.y)
       ;}
13     friend complex operator + (complex a,complex b){return complex(a.x + b.x,a.y + b.y)
       ;}
14     friend complex operator * (complex a,complex b){return complex(a.x * b.x - a.y * b.y
       ,a.x * b.y + a.y * b.x);}
15 }a[maxn << 2],b[maxn << 2];
16 int r[maxn << 2];
17 void FFT(int limit,complex *A,int *r,int type){  //limit一定是2的幂次，这样保证每次可以对
       半分
18     for(int i = 0 ; i < limit; i ++) if(i < r[i]) swap(A[i],A[r[i]]);
19     for(int mid = 1; mid < limit; mid <<= 1){  //待合并区间长度的一半
20         complex Wn(cos(PI / mid),type * sin(PI / mid)); //单位根,mid只有一半所以分子不用
       * 2
21         int len = mid << 1;  //区间长度
22         for(int j = 0; j < limit; j += len){ //j表示更新到的位置
23             complex w(1,0);
24             for(int k = 0 ; k < mid; k ++, w = w * Wn){
25                 complex x = A[j + k],y = w * A[j + mid + k];
26                 A[j + k] = x + y;
27                 A[j + mid + k] = x - y;
28             }
29         }
30     }
31 }
32 int main(){
33     Sca2(N,M);
34     for(int i = 0; i <= N ; i ++){a[i].y = 0; a[i].x = read();}
35     for(int j = 0; j <= M ; j ++){b[j].y = 0; b[j].x = read();}
36     int limit = 1,l = 0; while(limit <= N + M) limit <<= 1,l++;
37     for(int i = 0 ; i < limit; i ++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
38     FFT(limit,a,r,1); FFT(limit,b,r,1);
39     for(int i = 0; i < limit ; i ++) a[i] = a[i] * b[i];
40     FFT(limit,a,r,-1);
```

```
41      for(int i = 0 ; i <= N + M; i ++){
42          printf("%d ",(int)(a[i].x / limit + 0.5));
43      }
44      return 0;
45  }
```

4.18.4 字符串匹配

</> 代码 4.40: /数论/fft 字符串匹配

```
1   /* FFT在字符串匹配中的应用
2   (1).普通的单模式串匹配(KMP模板题)
3   模式串A,长度为m, 文本串B,长度为n
4   1.定义A(x),B(y)的匹配函数C(x,y)为(A(x) - B(y))^2,若C(x,y)=0, 则称A的第x个字符和B的第y个字符
        匹配
5   2.定义完全匹配函数P(x) = C(i,x - m + i + 1) (0 <= i <= m - 1) 若P(x)=0, 则称B以第x位结束
        的连续m位, 与A完全匹配
6   3.将A反转为S,同时将S, B代入完全匹配函数P(x),得到
7   P(x) = [S(m - i - 1) - B(x - m + i + 1)]^2 (0 <= i <= m - 1)
8   展开化简, 得到
9   P(x) = S(m - i + 1)^2 + B(x - m + i + 1)^2 - 2 S(m - i - 1)B(x - m + i + 1) (0 <= i <= m -
        1)
10  第一项第二项是一个前缀和, 预处理一下就可以, 发现第三项的两个系数加起来正好是x,所以第三项可以
        写为
11  -2 S(i)B(j) (i + j = x) ,这就是一个基本卷积的形式了, 可以用FFT做
12   */
13   void FFT_Match(char *s1,char *s2,int m,int n){
14      for(int i=0;i<m;i++) A[i].x=s1[i]-'a'+1;
15      for(int i=0;i<n;i++) B[i].x=s2[i]-'a'+1;
16      reverse(A,A+m);double T=0;
17      for(int i=0;i<m;i++) T+=A[i].x*A[i].x;
18      f[0]=B[0].x*B[0].x;
19      for(int i=1;i<n;i++) f[i]=f[i-1]+B[i].x*B[i].x;
20      FFT(A,len,1);FFT(B,len,1);
21      for(int i=0;i<len;i++) g[i]=A[i]*B[i];
22      FFT(g,len,-1);
23      for(int x=m-1;x<n;x++){
24          double P=T+f[x]-f[x-m]-2*g[x].r;
25          if(fabs(P)<eps) printf("%d ",x-m+2);
26      }
27  }
28  /*(2)带有通配符的模式串匹配
29  这就不能用KMP了, 但是对于fft来说, 方法和上面相似
30  1.定义A(x),B(y)的匹配函数C(x,y) = (A(x) - B(y))^2 * A(x) * B(y); 将通配符*的值赋0,表示除了
        他俩相等, 还有出现通配符的情况都算匹配
```

```
2.定义完全匹配书P(x) = C(i,x - m + i + 1) (0 <= i <= m - 1)若P(x)=0, 则称B以第x位结束的连续m位, 与A完全匹配
3.将A反转为S,同时将S,B代入完全匹配函数P(x),得到
P(x) = [S(m - i - 1) - B(x - m + i + 1)]² * S(m - i - ) * B(x - m + i + 1)   (0 <= i <= m - 1)
展开化简,得到
P(x) = S(m - i - 1)³B(x - m + i + 1) + S(m - i - 1)B(x - m + i + 1)³ - 2S(m - i - 1)²B(x - m + i + 1)² (0 <= i <= m - 1)
因为(m - i + 1) + (x - m + i + 1) = x,所以
P(x) = S(i)³B(j) + S(i)B(j)³ - 2S(i)²B(j)²  (i + j = x)
三段fft可以解决
 */
 const int maxn = 3e5 + 10;
const double PI = acos(-1.0);
int N,M,K;
char str1[maxn],str2[maxn];
struct complex{
    double x,y;
    complex(double x = 0,double y = 0):x(x),y(y){}
    void init(){x = y = 0;}
    friend complex operator + (complex a,complex b){return complex(a.x + b.x,a.y + b.y);}
    friend complex operator - (complex a,complex b){return complex(a.x - b.x,a.y - b.y);}
    friend complex operator * (complex a,complex b){return complex(a.x * b.x - a.y * b.y,a.x * b.y + a.y * b.x);}
}a[maxn << 2],b[maxn << 2],ans[maxn << 2];
int ca[maxn],cb[maxn];
int r[maxn << 2];
vector<int>s;
void FFT(int limit,complex *A,int *r,int type){
    for(int i = 0 ; i < limit; i ++){
        if(i < r[i]) swap(A[i],A[r[i]]);
    }
    for(int mid = 1; mid < limit; mid <<= 1){
        int len = mid << 1;
        complex Wn(cos(PI/mid),type * sin(PI/mid));
        for(int j = 0 ; j < limit; j += len){
            complex w(1,0);
            for(int k = 0 ; k < mid; k ++,w = w * Wn){
                complex x = A[j + k],y = w * A[j + mid + k];
                A[j + k] = x + y;
                A[j + mid + k] = x - y;
            }
```

```
69          }
70      }
71  }
72  inline double cul(double x,int y){
73      if(y == 1) return x;
74      if(y == 2) return x * x;
75      return x * x * x;
76  }
77  int main(){
78      Sca2(N,M);
79      scanf("%s%s",str1,str2); //长串str2,短串str1
80      for(int i = 0 ; i < N / 2; i ++) swap(str1[i],str1[N - i - 1]);
81      int limit = 1,l = 0;
82      while(limit <= N + M - 2) limit <<= 1,l++;
83      for(int i = 0 ; i < limit; i ++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
84      for(int i = 0 ; i < N ; i ++) if(str1[i] != '*') ca[i] = str1[i] - 'a' + 1;
85      for(int i = 0 ; i < M ; i ++) if(str2[i] != '*') cb[i] = str2[i] - 'a' + 1;
86      for(int k = 1; k <= 3; k ++){
87          for(int i = 0 ; i < limit; i ++) a[i].init(),b[i].init();
88          for(int i = 0 ; i < N ; i ++) a[i].x = cul(ca[i],k);
89          for(int i = 0 ; i < M ; i ++) b[i].x = cul(cb[i],3 - k + 1);
90          FFT(limit,a,r,1); FFT(limit,b,r,1);
91          int t = 1; if(k == 2) t = -2;
92          for(int i = 0 ; i < limit; i ++) ans[i] = ans[i] + t * a[i] * b[i];
93      }
94      FFT(limit,ans,r,-1);
95      for(int i = N - 1; i < M; i ++){   //一般短串逆置的范围
96          if(!(int)(ans[i].x / limit + 0.5)) s.push_back(i - N + 2);
97      }
98      Pri(s.size());
99      for(int i = 0 ; i < s.size(); i ++) printf("%d ",s[i]);
100     return 0;
101 }
```

4.18.5 NTT

</> 代码 4.41: /数论/NTT

```
1  //NTT(快速数论变换)
2  /*
3  干的事情和FFT差不多,优点: 1.可以取模, FFT的浮点运算时不可以取模的
4  2.全部整数运算避免了fft的浮点数误差
5  缺点:1.多项式的系数必须是整数
6  2.如果取模的数字不是998244353,1004535809,469762049这几个原根是3的数字,
7  需要中国剩余定理求, 很麻烦
```

```
*/
//模数为998244353
const int maxn = 3 * 1e6 + 10, P = 998244353, G = 3, Gi = 332748118;
int N, M, limit = 1, L, r[maxn];
LL a[maxn], b[maxn];
inline LL fastpow(LL a, LL k) {
    LL base = 1;
    while(k) {
        if(k & 1) base = (base * a ) % P;
        a = (a * a) % P;
        k >>= 1;
    }
    return base % P;
}
inline void NTT(LL *A, int type) {
    for(int i = 0; i < limit; i++)
        if(i < r[i]) swap(A[i], A[r[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        LL Wn = fastpow( type == 1 ? G : Gi , (P - 1) / (mid << 1));
        for(int j = 0; j < limit; j += (mid << 1)) {
            LL w = 1;
            for(int k = 0; k < mid; k++, w = (w * Wn) % P) {
                int x = A[j + k], y = w * A[j + k + mid] % P;
                A[j + k] = (x + y) % P,
                A[j + k + mid] = (x - y + P) % P;
            }
        }
    }
}
int main() {
    N = read(); M = read();
    for(int i = 0; i <= N; i++) a[i] = (read() + P) % P;
    for(int i = 0; i <= M; i++) b[i] = (read() + P) % P;
    while(limit <= N + M) limit <<= 1, L++;
    for(int i = 0; i < limit; i++) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (L - 1));
    NTT(a, 1);NTT(b, 1);
    for(int i = 0; i < limit; i++) a[i] = (a[i] * b[i]) % P;
    NTT(a, -1);
    LL inv = fastpow(limit, P - 2);
    for(int i = 0; i <= N + M; i++)
        printf("%d ", (a[i] * inv) % P);
    return 0;
}
//任意模数NTT
```

```cpp
int mod;
namespace Math {
    inline int pw(int base, int p, const int mod) {
        static int res;
        for (res = 1; p; p >>= 1, base = static_cast<long long> (base) * base % mod) if (p & 1) res = static_cast<long long> (res) * base % mod;
        return res;
    }
    inline int inv(int x, const int mod) { return pw(x, mod - 2, mod); }
}

const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049, G = 3;
const long long mod_1_2 = static_cast<long long> (mod1) * mod2;
const int inv_1 = Math::inv(mod1, mod2), inv_2 = Math::inv(mod_1_2 % mod3, mod3);
struct Int {
    int A, B, C;
    explicit inline Int() { }
    explicit inline Int(int __num) : A(__num), B(__num), C(__num) { }
    explicit inline Int(int __A, int __B, int __C) : A(__A), B(__B), C(__C) { }
    static inline Int reduce(const Int &x) {
        return Int(x.A + (x.A >> 31 & mod1), x.B + (x.B >> 31 & mod2), x.C + (x.C >> 31 & mod3));
    }
    inline friend Int operator + (const Int &lhs, const Int &rhs) {
        return reduce(Int(lhs.A + rhs.A - mod1, lhs.B + rhs.B - mod2, lhs.C + rhs.C - mod3));
    }
    inline friend Int operator - (const Int &lhs, const Int &rhs) {
        return reduce(Int(lhs.A - rhs.A, lhs.B - rhs.B, lhs.C - rhs.C));
    }
    inline friend Int operator * (const Int &lhs, const Int &rhs) {
        return Int(static_cast<long long> (lhs.A) * rhs.A % mod1, static_cast<long long> (lhs.B) * rhs.B % mod2, static_cast<long long> (lhs.C) * rhs.C % mod3);
    }
    inline int get() {
        long long x = static_cast<long long> (B - A + mod2) % mod2 * inv_1 % mod2 * mod1 + A;
        return (static_cast<long long> (C - x % mod3 + mod3) % mod3 * inv_2 % mod3 * (mod_1_2 % mod) % mod + x) % mod;
    }
} ;

#define maxn 131072

```

```
90  namespace Poly {
91  #define N (maxn << 1)
92      int lim, s, rev[N];
93      Int Wn[N | 1];
94      inline void init(int n) {
95          s = -1, lim = 1; while (lim < n) lim <<= 1, ++s;
96          for (register int i = 1; i < lim; ++i) rev[i] = rev[i >> 1] >> 1 | (i & 1) << s;
97          const Int t(Math::pw(G, (mod1 - 1) / lim, mod1), Math::pw(G, (mod2 - 1) / lim,
        mod2), Math::pw(G, (mod3 - 1) / lim, mod3));
98          *Wn = Int(1); for (register Int *i = Wn; i != Wn + lim; ++i) *(i + 1) = *i * t;
99      }
100     inline void NTT(Int *A, const int op = 1) {
101         for (register int i = 1; i < lim; ++i) if (i < rev[i]) std::swap(A[i], A[rev[i
        ]]);
102         for (register int mid = 1; mid < lim; mid <<= 1) {
103             const int t = lim / mid >> 1;
104             for (register int i = 0; i < lim; i += mid << 1) {
105                 for (register int j = 0; j < mid; ++j) {
106                     const Int W = op ? Wn[t * j] : Wn[lim - t * j];
107                     const Int X = A[i + j], Y = A[i + j + mid] * W;
108                     A[i + j] = X + Y, A[i + j + mid] = X - Y;
109                 }
110             }
111         }
112         if (!op) {
113             const Int ilim(Math::inv(lim, mod1), Math::inv(lim, mod2), Math::inv(lim,
        mod3));
114             for (register Int *i = A; i != A + lim; ++i) *i = (*i) * ilim;
115         }
116     }
117 #undef N
118 }
119
120 int n, m;
121 Int A[maxn << 1], B[maxn << 1];
122 int main() {
123     scanf("%d%d%d", &n, &m, &mod); ++n, ++m;
124     for (int i = 0, x; i < n; ++i) scanf("%d", &x), A[i] = Int(x % mod);
125     for (int i = 0, x; i < m; ++i) scanf("%d", &x), B[i] = Int(x % mod);
126     Poly::init(n + m);
127     Poly::NTT(A), Poly::NTT(B);
128     for (int i = 0; i < Poly::lim; ++i) A[i] = A[i] * B[i];
129     Poly::NTT(A, 0);
130     for (int i = 0; i < n + m - 1; ++i) {
```

```
131         printf("%d", A[i].get());
132         putchar(i == n + m - 2 ? '\n' : ' ');
133     }
134     return 0;
135 }
```

4.19 FWT

</> 代码 4.42: /数论/FWT

```
1  /* FWT 快速沃尔什变换
2  给定A,B两个序列,可以通过FWT求出序列C
3  其中C(i) = A(j)B(k) (j k = i) 可以为or,and,xor
4   */
5  const int maxn = 2e5 + 10;
6  const LL mod = 998244353;
7  int N,M,K;
8  LL inv2 = mod + 1 >> 1;
9  LL a1[maxn],b1[maxn],a2[maxn],b2[maxn],a3[maxn],b3[maxn];
10 inline LL add(LL a,LL b){return ((a + b) % mod + mod) % mod;}
11 inline LL mul(LL a,LL b){return (a % mod * b % mod + mod) % mod;}
12 void FWT_or(int limit,LL *a,int op){
13     for(int i = 1; i < limit; i <<= 1){
14         for(int p = i << 1,j = 0; j < limit; j += p){
15             for(int k = 0; k < i ; k ++){
16                 if(op == 1) a[i + j + k] = add(a[j + k],a[i + j + k]);
17                 else a[i + j + k] = add(a[i + j + k],-a[j + k]);
18             }
19         }
20     }
21 }
22 void FWT_and(int limit,LL *a,int op){
23     for(int i = 1; i < limit; i <<= 1){
24         for(int p = i << 1,j = 0; j < limit; j += p){
25             for(int k = 0 ; k < i ; k ++){
26                 if(op == 1) a[j + k] = add(a[j + k],a[i + j + k]);
27                 else a[j + k] = add(a[j + k],-a[i + j + k]);
28             }
29         }
30     }
31 }
32 void FWT_xor(int limit,LL *a,int op){
33     for(int i = 1; i < limit; i <<= 1){
34         for(int p = i << 1,j = 0;j < limit; j += p){
```

```
            for(int k = 0 ; k < i ; k ++){
                LL x = a[j + k],y = a[i + j + k];
                a[j + k] = add(x,y); a[i + j + k] = add(x,-y);
                if(op == -1) a[j + k] = mul(a[j + k],inv2),a[i + j + k] = mul(a[i + j + k],inv2);
            }
        }
    }
}
int main(){
    Sca(N); N = (1 << N); N--;
    for(int i = 0; i <= N ; i ++) Scl(a1[i]),a2[i] = a3[i] = a1[i];
    for(int i = 0; i <= N ; i ++) Scl(b1[i]),b2[i] = b3[i] = b1[i];
    int limit = 1; while(limit <= N) limit <<= 1;
    FWT_or(limit,a1,1); FWT_or(limit,b1,1);
    for(int i = 0 ; i < limit; i ++) a1[i] = mul(a1[i],b1[i]);
    FWT_or(limit,a1,-1);

    FWT_and(limit,a2,1); FWT_and(limit,b2,1);
    for(int i = 0 ; i < limit; i ++) a2[i] = mul(a2[i],b2[i]);
    FWT_and(limit,a2,-1);

    FWT_xor(limit,a3,1); FWT_xor(limit,b3,1);
    for(int i = 0 ; i < limit; i ++) a3[i] = mul(a3[i],b3[i]);
    FWT_xor(limit,a3,-1);

    for(int i = 0; i <= N ; i ++) printf("%lld%c",a1[i],i == N?'\n':' ');
    for(int i = 0; i <= N ; i ++) printf("%lld%c",a2[i],i == N?'\n':' ');
    for(int i = 0; i <= N ; i ++) printf("%lld%c",a3[i],i == N?'\n':' ');
    return 0;
}
```

4.20 反演定理

4.20.1 二项式反演

</> 代码 4.43: /数论/二项式反演

```
//二项式反演
/*
    f(n) = ΣC(n,i) * g(i) (0 <= i <= n)
->  g(n) = Σ((-1) ^ (n - i)) * C(n,i) * f(i) (0 <= i <= n)
 */
//模板：n封信对应n个信封，求恰好全部装错了信封的方案数
```

```
//设g(i)为恰好装错了i个信封的个数，则装的全部种类f(n) = g(1) * C(n,1) + g(2) * C(n,2)
    ..... + g(n) * C(n,n)
//代入二项式反演公式，其中f(n) = n!
int N,M,K;
LL fac[maxn];
int comb[30][30];
void init(){
    fac[0] = 1;
    for(int i = 1; i <= 20; i ++) fac[i] = fac[i - 1] * i;
    for(int i = 0 ; i <= 20; i ++){
        comb[i][0] = comb[i][i] = 1;
        for(int j = 1; j < i ; j ++){
            comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
        }
    }
}
int main(){
    init();
    while(~Sca(N)){
        LL ans = 0;
        for(int i = 0 ; i <= N ; i ++){
            LL p = comb[N][i] * fac[i];
            if((N - i) & 1) p = -p;
            ans += p;
        }
        Prl(ans);
    }
    return 0;
}
```

4.20.2 莫比乌斯反演

</> 代码 4.44: /数论/莫比乌斯反演

```
//莫比乌斯反演
/*
f(n) = Σg(d) 其中(d|n)
->g(n) =  (d)f(n / d) 其中(d|n)
或者描述为
f(n) =  g(d) 其中(n|d)
->g(n) =  (d/n) * f(d) 其中(n|d)
*/
/*
 (d)为莫比乌斯函数，计算方式为
 (1) = 1
```

```
x = p1 * p2 * p3 ……*pk (x由k个不同的质数组成) 则 (x) = (-1)^k
其他情况,  (x) = 0
 (d)的常见性质: 对于任意正整数n有
1. (d) (d|n)  在n = 1的时候为1, 在n > 1的时候为0
2. (d)/d =  (n)/n , 其中(d|n), (n)为欧拉函数
 */
//线性筛求莫比乌斯函数 O(n)
const int maxn = 1e6 + 5;
int mu[maxn], vis[maxn], prime[maxn];
int tot;//用来记录prime的个数
void init(){
    mu[1] = 1;
    for(int i = 2; i < maxn; i ++){
        if(!vis[i]){
            prime[tot ++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < tot && i * prime[j] < maxn; j ++){
            vis[i * prime[j]] = 1;
            if(i % prime[j]) mu[i * prime[j]] = -mu[i];
            else{
                mu[i * prime[j]] = 0;
                break;
            }
        }
    }
}
int main(){
    init();
}
```

4.21 数的位数公式

</> 代码 4.45: /数论/数的位数公式

```
//求一个数x的位数的公式
//x的位数 = log10(x) + 1
/*
    例题: 洛谷P2759, 求x ^ x大于等于n位数的最小x
    将log(x ^ x) 化为x * log(x),二分查找即可
*/
LL N;
bool check(LL x){
    return x * log10(x) + 1 >= N;
```

```
}
LL solve(){
    LL l = 0,r = N;
    LL ans = 0;
    while(l <= r){
        LL m = (l + r) >> 1;
        if(check(m)){
            r = m - 1;
            ans = m;
        }else{
            l = m + 1;
        }
    }
    return ans;
}
int main(){
    Scl(N);
    Prl(solve());
    return 0;
}
```

4.22 常用公式

1. 基础内容

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

导数

$$f'(x) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$[f(x) \pm g(x)]' = f'(x) \pm g'(x)$$

$$[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x)$$

$$[\frac{f(x)}{g(x)}]' = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$$

$$\frac{d}{dx} f(g(x)) = \frac{df}{dg}(g(x))\frac{dg}{dx}(x)$$

导数公式

若 $f(x) = C(C\quad)$，则 $f'(x) = 0$

若 $f(x) = x^a(\alpha \in \mathbb{Q}^*)$，则 $f'(x) = ax^{a-1}$

若 $f(x) = sin(x)$，则 $f'(x) = cos(x)$

若 $f(x) = cos(x)$，则 $f'(x) = -sin(x)$

若 $f(x) = a^x$，则 $f'(x) = a^x \ln a$

若 $f(x) = e^x$，则 $f'(x) = e^x$

若 $f(x)=\log_a x$，则 $f'(x)=\frac{1}{x\ln a}$

若 $f(x)=\ln x$，则 $f'(x)=\frac{1}{x}$

积分

$$\int_a^b f(x)\mathrm{d}x=\sum_{i=1}^{n}f(\xi_i)\Delta x_i=\lim_{n\to\infty}\sum_{i=1}^{n}f[a+\frac{i}{n}(b-a)]\frac{b-a}{n}$$

$$\int_a^b f(x)\mathrm{d}x=F(x)|_a^b=F(b)-F(a)\quad F'(x)=f(x)$$

$$\int_a^b kf(x)\mathrm{d}x=k\int_a^b f(x)\mathrm{d}x$$

$$\int_a^b [f(x)\pm g(x)]\mathrm{d}x=\int_a^b f(x)\mathrm{d}x\pm\int_a^b g(x)\mathrm{d}x$$

$$\int_a^b f(x)\mathrm{d}x=\int_a^c f(x)\mathrm{d}x+\int_c^b f(x)\mathrm{d}x$$

$$\int_a^b f(x)\mathrm{d}x=-\int_b^a f(x)\mathrm{d}x$$

$$\int_a^a f(x)\mathrm{d}x=0$$

积分公式

$$\int k\,\mathrm{d}x=kx+C$$

$$\int x^a\,\mathrm{d}x=\frac{x^{a+1}}{a+1}+C(a\neq -1)$$

$$\int\frac{\mathrm{d}x}{x}=\ln|x|+C$$

$$\int e^x\,\mathrm{d}x=e^x+C$$

$$\int a^x\,\mathrm{d}x=\frac{a^x}{\ln a}+C$$

$$\int\frac{\mathrm{d}x}{1+x^2}=arctan(x)+C$$

$$\int\frac{\mathrm{d}x}{\sqrt{1-x^2}}=arcsin(x)+C$$

$$\int cos(x)\,\mathrm{d}x=sin(x)+C$$

$$\int sin(x)\,\mathrm{d}x=-cos(x)+C$$

$$\int\frac{\mathrm{d}x}{cos^2(x)}\,\mathrm{d}x=\int sec^2(x)\,\mathrm{d}x=tan(x)+C$$

$$\int\frac{\mathrm{d}x}{sin^2(x)}\,\mathrm{d}x=\int csc^2(x)\,\mathrm{d}x=-cot(x)+C$$

$$\int sec(x)tan(x)\,\mathrm{d}x=sec(x)+C$$

$$\int csc(x)cot(x)\,\mathrm{d}x=-csc(x)+C$$

2. 上升幂与下降幂

$$x^{\underline{n}}=(x-1)^{\underline{n-1}}x=\frac{(x)!}{(x-n)!}=\prod_{i=x-n+1}^{x}i(x^{\underline{0}}=1)$$

$$x^{\overline{n}}=(x+1)^{\overline{n-1}}x=\frac{(x+n-1)!}{(x-1)!}=\prod_{i=x}^{x+n-1}i(x^{\overline{0}}=1)$$

推导结论

$$x^{\underline{n}} = (-1)^n (-x)^{\overline{n}}$$
$$x^{\overline{n}} = (-1)^n (-x)^{\underline{n}}$$
$$x^{\underline{n}} = A_x^n$$
$$x^{\overline{n}} = A_{x+n-1}^n$$

3. 单位根

$$\omega_n^k = \cos(\frac{2\pi k}{n}) + \sin(\frac{2\pi k}{n})i$$
$$\omega_n^k = g^{\frac{k(P-1)}{n}} \mod P(P = k2^t + 1, P \in \{Prime\})$$

推导结论

$$\omega_n^k = (\omega_n^1)^k$$
$$\omega_n^j \omega_n^k = \omega_n^{j+k}$$
$$\omega_{2n}^{2k} = \omega_n^k$$
$$\omega_n^{(k+n/2)} = -\omega_n^k (n \quad)$$
$$\sum_{i=1}^{n-1} \omega_n^i = 0$$

4. Fibonacci 数列

$$fib_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib_{n-1} + fib_{n-2} & n > 1 \end{cases}$$

推导结论

$$\sum_{i=1}^{n} f_i = f_{n+2} - 1$$
$$\sum_{i=1}^{n} f_{2i-1} = f_{2n}$$
$$\sum_{i=1}^{n} f_{2i} = f_{2n+1} - 1$$
$$\sum_{i=1}^{n} (f_i)^2 = f_i f_{i+1}$$
$$f_{n+m} = f_{n-1} f_{m-1} + f_n f_m$$
$$(f_n)^2 = (-1)^{(n-1)} + f_{n-1} f_{n+1}$$
$$f_{2n-1} = (f_n)^2 - (f_{n-2})^2$$
$$f_n = \frac{f_{n+2} + f_{n-2}}{3}$$
$$\frac{f_i}{f_{i-1}} \approx \frac{\sqrt{5}-1}{2} \approx 0.618$$
$$f_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

5. GCD 和 LCM

$$\gcd(a,b) = \gcd(b, a-b)(a > b)$$
$$\gcd(a,b) = \gcd(b, a \mod b)$$
$$\gcd(a,b)\,\mathrm{lcm}(a,b) = ab$$

推导结论

$$k|\gcd(a,b) \iff k|a\ k|b$$

$$\gcd(k,ab)=1 \iff \gcd(k,a)=1\ \gcd(k,b)=1$$

$$(a+b)\mid ab \Longrightarrow \gcd(a,b)\neq 1$$

在 Fibonacc 数列中求相邻两项的 gcd 时，辗转相减次数等于辗转相除次数。

$$\gcd\left(fib_n,fib_m\right)=fib_{\gcd(n,m)}$$

6. 裴蜀定理

设 a,b 是不全为零的整数，则存在整数 x,y ，使得 $ax+by=\gcd(a,b)$

$$\gcd(a,b)|d \iff \exists x,y\in\mathbb{Z},ax+by=d$$

推导结论

设不定方程 $ax+by=\gcd(a,b)$ 的一组特解为 $\begin{cases}x=x_0\\y=y_0\end{cases}$,则 $ax+by=c(\gcd(a,b)|c)$ 的通解为 $\begin{cases}x=\frac{c}{\gcd(a,b)}x_0+k\frac{b}{\gcd(a,b)}\\y=\frac{c}{\gcd(a,b)}y_0-k\frac{a}{\gcd(a,b)}\end{cases}$ $\mathbb{Z}$) 。$\forall a,b,z\in\mathbb{N}^*,\gcd(a,b)=1,\exists x,y\in\mathbb{N},ax+by=ab-a-b+z$

7. 同余运算

$$\begin{cases}a\equiv b(\mathrm{mod}m)\\c\equiv d(\mathrm{mod}m)\end{cases}\Longrightarrow a+c\equiv b+d(\mathrm{mod}m)$$

$$\begin{cases}a\equiv b(\mathrm{mod}m)\\c\equiv d(\mathrm{mod}m)\end{cases}\Longrightarrow a-c\equiv b-d(\mathrm{mod}m)$$

$$a\equiv b(\mathrm{mod}m)\Longrightarrow ak\equiv bk(\mathrm{mod}m)$$

$$ka\equiv kb(\mathrm{mod}m),\gcd(k,m)=1\Longrightarrow a\equiv b(\mathrm{mod}m)$$

8. 费马小定理及其扩展

$$P\in\{Prime\},P\nmid a\Longrightarrow a^{P-1}=1(\mathrm{mod}P)$$

推导结论

对于任意多项式 $F(x)=\sum_{i=0}^{\infty}a_ix^i$ （a_i 对一个质数 P 取模），若满足 $a_0\equiv 1(\mathrm{mod}P)$，则 $\forall n\leqslant P,F^P(x)\equiv 1(\mathrm{mod}x^n)$ 。

9. 中国剩余定理（CRT）及其扩展

若 $m_1,m_2...m_k$ 两两互素，则同余方程组

$$\begin{cases}x\equiv a_1\,(\mathrm{mod}m_1)\\x\equiv a_2\,(\mathrm{mod}m_2)\\\vdots\\x\equiv a_k\,(\mathrm{mod}m_k)\end{cases}$$

有唯一解为：$x=\sum_{i=1}^{k}a_iM_iM_i^{-1}$, 其中 $M_i=\prod_{j\neq i}m_j$ 。

10. 佩尔（Pell）方程

形如 $x^2-Dy^2=1(D\in\mathbb{N}^*$且为非平方数) 的方程被称为第一类佩尔方程。设它的一组最小正整数解为 $\begin{cases}x=x_0\\y=y_0\end{cases}$ ，

则其第 n 个解满足：$x_n+\sqrt{D}y_n=(x_0+\sqrt{D}y_0)^{n+1}$, 递推式为 $\begin{cases}x_n=x_0x_{n-1}+Dy_0y_{n-1}\\y_n=x_0y_{n-1}+y_0x_{n-1}\end{cases}$ 。

形如 $x^2-Dy^2=-1(D\in\mathbb{N}^*$且为非平方数) 的方程被称为第二类佩尔方程。设它的一组最小正整数解为 $\begin{cases}x=x_0\\y=y_0\end{cases}$ ，

则其第 n 个解满足： $x_n+\sqrt{D}y_n=(x_0+\sqrt{D}y_0)^{2n+1}$ 。

11. 勾股方程

方程 $x^2+y^2=z^2$ 的正整数通解为

$$\begin{cases}x=k(u^2-v^2)\\y=2kuv & (u,v\in\{Prime\},k\in\mathbb{N}^*)\\z=k(u^2+v^2)\end{cases}$$

, 且均满足 $\gcd(x,y,z)=k$ 。

12. 牛顿二项式定理

$$(x+y)^n=\sum_{i=0}^{n}C_n^ix^{n-i}y^i$$

推导结论

$$\sum_{i=0}^{n}C_n^i=2^n$$
$$\sum_{i=0}^{n}iC_n^i=n2^{n-1}$$
$$\sum_{i=0}^{n}i^2C_n^i=n(n+1)2^{n-1}$$

13. 广义牛顿二项式定理

$$C_r^n=\begin{cases}0 & n<0,r\in\mathbb{R}\\1 & n=0,r\in\mathbb{R}\\\frac{r(r-1)\cdots(r-n+1)}{n!} & n>0,r\in\mathbb{R}\end{cases}$$
$$(1+x)^{-n}=\sum_{i=0}^{\infty}C_{-n}^ix^i=\sum_{i=0}^{\infty}(-1)^iC_{n+i-1}^ix^i$$
$$(x+y)^\alpha=\sum_{i=0}^{\infty}C_\alpha^ix^{\alpha-i}y^i(x,y,\alpha\in\mathbb{R}\text{ 且 }|\frac{x}{y}|<1)$$

14. 斯特林数

$s_n^m=s_{n-1}^{m-1}+(n-1)s_{n-1}^m(s_n^n=1,s_n^0=0^n)$【第一类斯特林数】

$S_n^m=S_{n-1}^{m-1}+mS_{n-1}^m(S_n^n=1,S_n^0=0^n)$【第二类斯特林数】

$S_n^m=\frac{\sum_{i=0}^{m}(-1)^{m-i}C_m^ii^n}{m!}=\sum_{i=0}^{m}\frac{(-1)^{m-i}}{(m-i)!}\frac{i^n}{i!}$

推导结论

$$n! = \sum_{i=0}^{n} s_n^i$$

$$x^{\overline{n}} = \sum_{i=0}^{n} s_n^i x^i$$

$$x^{\underline{n}} = \sum_{i=0}^{n} s_n^i x^i (-1)^{n-i}$$

$$x^n = \sum_{i=0}^{x,n} S_n^i x^{\underline{i}}$$

$$x^n = \sum_{i=0}^{x,n} S_n^i x^{\overline{i}} (-1)^{n-i}$$

$$\sum_{i=1}^{n} S_n^i s_i^m = \sum_{i=0}^{n} s_n^i S_i^m$$

$$\sum_{i=0}^{n} i^k = \sum_{j=0}^{n} j! S_k^j C_{n+1}^{j+1}$$

$$\sum_{i=1}^{n} C_n^i i^k = \sum_{j=0}^{k} S_k^j 2^{n-j} \frac{n!}{(n-j)!}$$

15. 贝尔数 (Bell)

$$B_n = \sum_{i=1}^{n} S_n^i (B_0 = 1)$$

$$B_n = \sum_{i=0}^{n-1} C_{n-1}^i B_i$$

16. Polya 定理

$$ans = \frac{\sum_{i=1}^{n} m^{k_i}}{n}$$

17. 容斥原理

$f(i) = \sum_{j=i}^{n} (-1)^{j-i} C_j^i g(j) = g(i) - \sum_{j=i+1} C_j^i f(j)$（$f(i)$ 为恰好 i 个满足 $balabala$ 的方案数，$g(i)$ 为钦定 i 个满足 $balabala$ 其他随意的方案数）

18. 生成函数

普通生成函数 (OGF) 收敛性式

$$\sum_{i=0}^{\infty} a^i x^i = \frac{1}{1-ax}$$
$$\sum_{i=0}^{\infty} (i+1) x^i = \frac{1}{(1-x)^2}$$
$$\sum_{i=0}^{\infty} C_n^i x^i = (1+x)^n$$
$$\sum_{i=0}^{\infty} C_{n+i-1}^i x^i = \frac{1}{(1-x)^n}$$
$$\sum_{i=0}^{\infty} fib_i x^i = \frac{x}{1-x-x^2}$$
$$\sum_{i=0}^{\infty} (\sum_{j=0}^{i} fib_j) x^i = \frac{x}{(1-x)(1-x-x^2)}$$
$$\sum_{i=0}^{\infty} cat_i x^i = \frac{1-\sqrt{1-4x}}{2x}$$

指数生成函数 (EGF) 收敛性式

$$\sum_{i=0}^{\infty} \frac{x^i}{i!} = e^x$$
$$\sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!} = \frac{e^x + e^{-x}}{2}$$
$$\sum_{i=0}^{\infty} \frac{x^{2i+1}}{(2i+1)!} = \frac{e^x - e^{-x}}{2}$$
$$\sum_{i=0}^{\infty} B_i \frac{x_i}{i!} = e^{e^x - 1}$$

19. 欧拉反演

$$\sum_{d|n} \varphi(d) = n \quad \varphi * 1 = \mathrm{id}$$

推导结论

$$\gcd(i,j) = \sum_{d|i, d|j} \varphi(d)$$
$$\sum_{i=1}^{n}\sum_{j=1}^{n} \gcd(i,j) = \sum_{d=1}^{n} d \left(2\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \varphi(i) - 1 \right)$$
$$\sum_{i=1}^{n}\sum_{j=1}^{m} \gcd(i,j) = \sum_{d=1}^{n} \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$$
$$\prod_{i=1}^{n}\prod_{j=1}^{n} \left(\frac{\mathrm{lcm}(i,j)}{\gcd(i,j)} \right) = \frac{(n!)^{2n}}{\left(\prod_{d=1}^{n} d^{\left(2S_\varphi(\lfloor \frac{n}{a} \rfloor)-1\right)} \right)^2}$$

20. 狄利克雷卷积 (Dirichlet) 与莫比乌斯反演 (Mobius)

$(f*g)(n) = \sum_{d|n} f(d) g(\frac{n}{d})$

$\sum_{d|n} \mu(d) = \epsilon(n)$ （即 $\mu * 1 = \epsilon$）

$f(n) = \sum_{d|n} g(d) \Longrightarrow g(n) = \sum_{d|n} \mu(\frac{n}{d}) f(d)$ （即 $f = g * 1 \Longrightarrow g = f * \mu$）

$f(n) = \sum_{n|d} g(d) \Longrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

$f(k)=\sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} g(dk) \Longrightarrow g(k)=\sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d)f(dk)$

推导结论

GCD 和 LCM

$$\gcd(i,j)=1]=\sum_{d|i,d|j}\mu(d)$$

$$\sum_{i=1}^{n}\sum_{i=1}^{m}[\gcd(i,j)=k]=\sum_{d=1}^{\lfloor \frac{n}{k} \rfloor}\mu(d)\lfloor\frac{n}{dk}\rfloor\lfloor\frac{m}{dk}\rfloor$$

$$\sum_{i=1}^{n}\sum_{i=1}^{m}[\gcd(i,j)\in\{Prime\}]=\sum_{d=1}^{n}\left(\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor\sum_{p|d\ \&\ p\in\{Prime\}}\mu(\frac{d}{p})\right)$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\operatorname{lcm}(i,j)=\sum_{d=1}^{n}d\left(\sum_{x=1}^{\lfloor \frac{n}{d} \rfloor}x^2\mu(x)\sum_{i=1}^{\lfloor \frac{n}{dx} \rfloor}i\sum_{i=1}^{\lfloor \frac{m}{dx} \rfloor}j\right)$$

除数函数

$\sigma_k=\sum_{d|n}d^k$（即 $\sigma_k=\mathrm{id}_k*1$）

$\sigma_0(xy)=\sum_{i|x}\sum_{j|y}[\gcd(i,j)=1]$（其中 $\sigma_0(x)$ 表示 x 的约数个数）

$\sum_{i=1}^{n}\sigma_0(i)=\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$

$\sum_{i=1}^{n}\sum_{j=1}^{m}\sigma_0(ij)=\sum_{k=1}^{n}\mu(k)\left(\sum_{i=1}^{\lfloor \frac{n}{k} \rfloor}\lfloor\frac{n}{ik}\rfloor\right)\left(\sum_{i=1}^{\lfloor \frac{m}{k} \rfloor}\lfloor\frac{m}{ik}\rfloor\right)$

$\sigma_1(xy)=\sum_{i|x}\sum_{j|y}\frac{iy}{j}[\gcd(i,j)=1]$（其中 $\sigma_0(x)$ 表示 x 的约数和）

$\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_1(ij)=\sum_{d=1}^{n}\mu(d)d\left(\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor}\sigma_1(i)\right)^2$

$\sum_{i=1}^{n}\sum_{j=1}^{m}\sigma_1(\gcd(i,j))=\sum_{d=1}^{n}\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor\left(\sum_{i|d}\mu(\frac{d}{i})\sigma_1(i)\right)$

莫比乌斯函数

$$\sum_{k=1}^{n}\mu^2(k)=\sum_{d=1}^{\sqrt{n}}\mu(d)\lfloor\frac{n}{d^2}\rfloor$$

$$\sum_{i=1}^{n}\mu^2(i)\sqrt{\frac{n}{i}}=n$$

21. 二项式反演

$$f(n)=\sum_{i=0}^{n}C_n^ig(i)\Longleftrightarrow g(n)=\sum_{i=0}^{n}(-1)^{n-i}C_n^if(i)$$

$$f(n)=\sum_{i=0}^{n}(-1)^iC_n^ig(i)\Longleftrightarrow g(n)=\sum_{i=0}^{n}(-1)^iC_n^if(i)$$

$$f(n)=\sum_{i=n}^{?}C_i^ng(i)\Longleftrightarrow g(n)=\sum_{i=n}^{?}(-1)^{i-n}C_i^nf(i)$$

$$f(n)=\sum_{i=n}^{?}(-1)^iC_i^ng(i)\Longleftrightarrow g(n)=\sum_{i=n}^{?}(-1)^iC_i^nf(i)$$

22. 斯特林反演

$$f(n)=\sum_{i=0}^{n}S_n^ig(i)\Longleftrightarrow g(n)=\sum_{i=0}^{n}(-1)^{n-i}s_n^ig(i)$$

$$f(n)=\sum_{i=0}^{n}s_n^ig(i)\Longleftrightarrow g(n)=\sum_{i=0}^{n}(-1)^{n-i}S_n^if(i)$$

$$f(n)=\sum_{i=n}^{?}S_i^ng(i)\Longleftrightarrow g(n)=\sum_{i=n}^{?}(-1)^{i-n}s_i^ng(i)$$

$$f(n)=\sum_{i=n}^{?}s_i^n g(i)\Longleftrightarrow g(n)=\sum_{i=n}^{?}(-1)^{i-n}S_i^n f(i)$$

23. 单位根反演

$$[n|k]=\frac{\sum_{i=0}^{n-1}w_n^{ik}}{n}$$

$$[a=b]=\frac{\sum_{i=0}^{n-1}w_n^{ai}w_n^{-ib}}{n}(a,b<n)$$

24. 子集反演

$f(S)=\sum_{T\subseteq S}g(T)\Longleftrightarrow g(S)=\sum_{T\subseteq S}(-1)^{|S|-|T|}f(T)$

$f(S)=\sum_{T\supseteq S}g(T)\Longleftrightarrow g(S)=\sum_{T\supseteq S}(-1)^{|T|-|S|}f(T)$

$f(S)=\sum_{T\subseteq S}g(T)\Longleftrightarrow g(S)=\sum_{T\subseteq S}\mu(|S|-|T|)f(T)$（$\mu(S)$ 在 S 有重复元素时为 0，否则为 $(-1)^{|S|}$）

$f(S)=\sum_{T\supseteq S}g(T)\Longleftrightarrow g(S)=\sum_{T\supseteq S}\mu(|T|-|S|)f(T)$（$\mu(S)$ 在 S 有重复元素时为 0，否则为 $(-1)^{|S|}$）

25. 最值反演（Min-Max 容斥）

$$\max(S)=\sum_{T\subseteq S}(-1)^{|T|+1}\min(T)$$

$$\min(S)=\sum_{T\subseteq S}(-1)^{|T|+1}\max(T)$$

$$E(\max(S))=\sum_{T\subseteq S}(-1)^{|T|+1}E(\min(T))$$

$$E(\min(S))=\sum_{T\subseteq S}(-1)^{|T|+1}E(\max(T))$$

$$\text{K-th}\max(S)=\sum_{T\subseteq S}(-1)^{|T|-k}C_{|T|-1}^{k-1}\min(T)$$

$$E(\text{K-th}\max(S))=\sum_{T\subseteq S}(-1)^{|T|-k}C_{|T|-1}^{k-1}E(\min(T))$$

26. 多项式

$$\begin{cases}F\left(\omega_n^k\right)=Fl\left(\omega_{n/2}^k\right)+\omega_n^kFR\left(\omega_{n/2}^k\right)\\F\left(\omega_n^{k+n/2}\right)=FL\left(\omega_{n/2}^k\right)-\omega_n^kFR\left(\omega_{n/2}^k\right)\end{cases}$$

$$\text{or}:\begin{cases}FWT:\{F_0=G_0,F_1=G_0+G_1\}\\IFWT:\{G_0=F_0,G_1=F_1-F_0\}\end{cases}$$

$$\text{and}:\begin{cases}FWT:\{F_0=G_0+G_1,F_1=G_1\}\\IFWT:\{G_0=F_0-F_1,G_1=F_1\}\end{cases}$$

$$\text{xor}:\begin{cases}FWT:\{F_0=G_0+G_1,F_1=G_0-G_1\}\\IFWT:\left\{G_0=\frac{F_0+F_1}{2},G_1=\frac{F_0-F_1}{2}\right\}\end{cases}$$

27. 拉格朗日插值

已知一个 n 次多项式 $F(x)$ 不同的 $n+1$ 处点值 $(x_i,y_i)_{i\in[0,n]}$，则

$$F(X)=\sum_{i=0}^{n}\left(y_i\prod_{j\neq i}\frac{X-x_j}{x_i-x_j}\right)$$

已知一个 n 次多项式 $F(x)$ 不同的 $n+1$ 处点值 $(i,y_i)_{i\in[0,n]}$，则

$$F(m+x)=\frac{(m+x)!}{(m+x-n-1)!}\sum_{i=x}^{n+x}\frac{1}{m-n+i}g(n+x-i)$$

，其中 $g(i)=\frac{y_i(-1)^{n-i}}{i!(n-i)!}$ 。

28. 多项式求逆

设 $F(x)=\sum_{n=0}^{\infty}a_nx^n, F^{-1}(x)=\sum_{n=0}^{\infty}b_nx^n$，则

$$b_0=\frac{1}{a_0}, b_n=\sum_{i=0}^{n-1}b_i\left(-\frac{a_{n-i}}{a_0}\right)$$

。

设 $F(x)G(x)\equiv 1(\mathrm{mod}x^n)$,

$$F(x)G(x)'\equiv 1(\mathrm{mod}x^{\frac{n}{2}}),\ G\equiv 2G'-FG'^2(\mathrm{mod}x^n)$$

第五章　搜索

5.1　二分查找

5.1.1　整数二分查找

</> 代码 5.1: /搜索/整数二分查找

```
const int M = 1e5+7;
int   p[M];
int main() {
    int    n,t;
    cin>>n>>t;
    for(int i=0; i<n; i++) cin>>p[i];
    //判断左右边界的二分（注意：有可能查找不到）
    //左右边界判断重点，某一边满足某一条件
    int l = 0, r = n-1, mid;
    while(l<r){             //左边界
        mid = (l+r)>>1;
        if(p[mid]<t) l = mid + 1;
        else r = mid;
    }
    l = 0, r = n-1;
    while(l<r){             //右边界
        mid = (l+r+1)>>1;   //注意一定要+1，否则死循环
        if(p[mid]<=t) l = mid;
        else r = mid - 1;
    }
    //二分整数查找
    l = 0, r = n-1;
    while(l<=r) {     //如果取的数组末尾后一位的话是l = r
        mid = (l+r)>>1;   //也可以mid = l+(r-l)/2;(尤其是迭代器的时候)
        if(p[mid]<t){l = mid + 1; continue;}
        else if(p[mid]>t) {r = mid - 1; continue;}
        else break;
    }
    return 0;
}
```

5.1.2 小数二分查找

</> 代码 5.2: /搜索/小数二分查找

```
const double esp = 1e-8;
int main() {
    double t;
    scanf("%lf",&t);
    double l = 0, r = t, mid;
    if(t<0)swap(l,r);   //判断一下区间正负
    //法一:
    while(r - l > esp) {
        mid = (l + r)/2;
        if(mid*mid*mid<t) l = mid;
        else if(mid*mid*mid>t) r = mid;
        else break;
    }
    mid = (l + r)/2;
    //法二
    while(r - l > esp) {
        mid = (l + r)/2;
        if(mid*mid*mid <= t) l = mid;
        else r = mid;
    }
    //其他
    int k = 100;
    while(k --)   {  //防止不必要的计算导致超时
        mid = (l + r)/2;
        if(mid*mid*mid <= t) l = mid;
        else r = mid;
    }
    return 0;
}
```

5.2 三分查找

</> 代码 5.3: /搜索/三分查找

```
//找四分点
double f(double a){/*根据题目意思计算*/}
double three(double l,double r) {//找凸点
    while(l<r-1) {
        double mid=(l+r)/2;
        double mmid=(mid+r)/2;
        if(f(mid)>f(mmid)) r=mmid;
```

```
        else l=mid;
    }
    if(f(l)>f(r)) return l;
    else return r;
}
//找三分点
double f(double a){/*根据题目意思计算*/}
double three(double l,double r) {
    while(l+EPS<r) {
        double mid=l+(r-l)/3;
        double midmid=r-(r-l)/3;
        if(f(mid)>f(midmid)) r=midmid;
        else l=mid;
    }
    return l;
}
```

5.3 BFS

5.3.1 多源 BFS

多源 BFS 就是把满足初始条件的点全部塞入队列

</> 代码 5.4: /搜索/多源 BFS

```
#define x first
#define y second
typedef pair<int, int> PII;
const int M = 1010;
int n, m, dist[M][M];
char g[M][M];
PII que[M * M];

void bfs()
{
    memset(dist, -1, sizeof dist);   //初始为-1, 用~来判断是否已经走过
    int hh = 0, tt = -1;
    for(int i = 0; i < n; i ++)
        for(int j = 0; j < m; j ++)
            if(g[i][j] == '1'){      //满足条件的点一开始直接全部压入队列
                que[++ tt] = {i, j};
                dist[i][j] = 0;
            }
    int dx[] = {1, 0, -1, 0}, dy[] = {0, 1, 0, -1};
    while(hh <= tt){
        PII now = que[hh ++];、
```

```
        for(int i = 0; i < 4; i ++){
            int a = now.x + dx[i], b = now.y + dy[i];
            if(a < 0 || a > n || b < 0 || b > n || ~dist[a][b]) continue;
            dist[a][b] = dist[now.x][now.y] + 1;
            que[++ tt] = {a, b};
        }
    }
}
int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n; i ++)scanf("%s", g[i]);
    bfs();
    for(int i = 0; i < n; i ++){
        for(int j = 0; j < m; j ++)
            printf("%d ", dist[i][j]);
        puts("");
    }
    return 0;
}
```

5.3.2 双向 BFS

从起始点和目标点同时向中间 BFS，两端接触算找到方案

</> 代码 5.5: /搜索/双向 BFS

```
int n;
string a[6], b[6];
int extend(queue<string> &now, unordered_map<string, int> &da, unordered_map<string, int
    > &db, string a[], string b[]){
    for(int cnt = now.size(), c = 0; c < cnt; c ++) {//只扩展一遍当前队列中已存在的元素,
    否则退化为普通bfs, 会TLE
        string k = now.front();
        now.pop();
        for(int i = 0; i < k.size(); i ++) {
            for(int j = 0; j < n; j ++) {
                if(k.substr(i, a[j].size()) == a[j]) {
                    string change = k.substr(0, i) + b[j] + k.substr(i + a[j].size());
                    if(da.count(change)) continue;
                    if(db.count(change)) return da[k] + db[change] + 1;
                    da[change] = da[k] + 1;
                    now.push(change);
                }
            }
        }
```

```
18      }
19      return INF;
20  }
21  int bfs(string start, string end) {
22  unordered_map<string, int> permp, sufmp; //头、尾
23      queue<string> per, suf; //头尾队列
24      per.push(start);
25      suf.push(end);
26      permp[start] = 0;
27      sufmp[end]   = 0;
28      while(per.size() && suf.size()) {// 如果有对列空了，说明不
29          int ans;        //双向广搜，每次扩展的队列是当前个数比较少的
30          if(per.size() <= suf.size()) ans = extend(per, permp, sufmp, a, b);
31          else ans = extend(suf, sufmp, permp, b, a);
32          if(ans != INF) return ans; //起点与终点第一次相遇便是最小值，直接返回
33      }
34      return INF;
35  }
36  int main() {
37      string A, B;
38      cin >> A >> B;
39      while(cin >> a[n] >> b[n]) n ++;
40      int ans = bfs(A, B);
41      if(ans > 10) puts("NO ANSWER!");
42      else printf("%d\n", ans);
43      return 0;
44  }
```

BFS 总结:

BFS 中的队列实际上就是一个简化的堆，可以保证队列内的数据，具有二段性，与单调性，所以第一次搜到的某点，就是最短路径。

双端队列 BFS:

可以解决边权为 0，1 的最短路问题，遇到边权为 0 的直接放在队头，为 1 放在队尾，这样队列依旧保持二段性与单调性。

5.4 A-star 算法

</> 代码 5.6: /搜索/A-star 算法

```
1  // 八数码问题，启发函数哈密顿距离 (dist = |x - x0| + |y - y0|)
2  typedef pair<int, string> PIS;
3  int f(string now) {// 启发函数
4      int dist = 0;
5      for(int i = 0; i < 9; i ++)
6          if(now[i] != 'x') {
```

```
            int k = now[i] - '1';
            dist += abs(i / 3 - k / 3) + abs(i % 3 - k % 3);
        }
    return dist;
}
void bfs(string st, string end) {
    priority_queue<PIS, vector<PIS>, greater<PIS>> heap; //小根堆
    unordered_map<string, pair<char, string>> road;      //存放路径
    unordered_map<string, int> dist;                     //存放距离
    heap.push({f(st), st});  dist[st] = 0;
    char op[] = {'u', 'r', 'd', 'l'};                    //对应操作符
    int dx[] = {-1, 0, 1, 0}, dy[] = {0, 1, 0, -1};
    while(heap.size()) {
        PIS t = heap.top();
        heap.pop();
        string now = t.second;
        if(now == end) break;                            //第一次找到终点便为最小操作方法
        int x, y;
        for(int i = 0; i < 9; i ++)
            if(now[i] == 'x') {
                x = i / 3;
                y = i % 3;
            }
        for(int i = 0; i < 4; i ++) {
            int a = x + dx[i], b = y + dy[i];
            if(a < 0 || a >= 3 || b < 0 || b >= 3) continue;
            swap(now[x * 3 + y], now[a * 3 + b]);
            if(!dist[now] || dist[now] > dist[t.second] + 1) {//更新距离
                dist[now] = dist[t.second] + 1;
                road[now] = {op[i], t.second};
                heap.push({dist[now] + f(now), now});
            }
            swap(now[x * 3 + y], now[a * 3 + b]);
        }
    }
    string ans;
    while(end != st) {
        ans += road[end].first;
        end = road[end].second;
    }
    reverse(ans.begin(), ans.end());
    cout << ans << endl;
}
int main() {
```

```
    string start, end;
    for(int i = 0; i < 9; i ++) {
        char k;
        cin >> k;
        start += k;
        if(i == 8) end += 'x';
        else end += char(i + '1');
    }
    int ni = 0;     //求逆序数，当逆序数为奇数，必然不能转换到原始状态
    for(int i = 0; i < 9; i ++)
        for(int j = i + 1; j < 9; j ++) {
            if(start[i] == 'x' || start[j] == 'x') continue;
            if(start[i] > start[j]) ni ++;
        }

    if(ni & 1) puts("unsolvable");
    else bfs(start, end);
    return 0;
}
//求第 K 短路，启发函数为Dijkstra
typedef pair<int, int> PII;
typedef pair<int, PII> PIII;
const int M = 1e5 + 7;
int n, m, S, T, K, idx;
int head[M], rhead[M], dist[M], cnt[M];
bool st[M];
struct Edge {
    int to, w, nxt;
}edge[M * 2];
void init() {
    memset(head, -1, sizeof head);
    memset(rhead, -1, sizeof rhead);
    memset(dist, 0x3f, sizeof dist);
}
void add(int head[], int u, int v, int w) {
    edge[idx] = {v, w, head[u]};
    head[u] = idx ++;
}
void Dijkstra() {   //求得是终点的最短路，作为启发函数
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    heap.push({0, T});
    dist[T] = 0;
    while(heap.size()) {
        PII now = heap.top();
```

```
95          heap.pop();
96          if(st[now.y]) continue;
97          st[now.y] = true;
98          for(int i = rhead[now.y]; ~i; i = edge[i].nxt){
99              int j = edge[i].to;
100             if(dist[j] > edge[i].w + now.x){
101                 dist[j] = edge[i].w + now.x;
102                 heap.push({dist[j], j});
103             }
104         }
105     }
106 }
107 int astra() {
108     priority_queue<PIII, vector<PIII>, greater<PIII>> heap;
109     heap.push({dist[S], {0, S}});
110     while(heap.size()) {
111         PIII now = heap.top();
112         heap.pop();
113         int distance = now.y.x, ver = now.y.y;
114         cnt[ver] ++;
115         if(cnt[T] == K)  return distance;      //第K次在堆顶遇到终点便为第K短路
116         for(int i = head[ver]; ~i; i = edge[i].nxt) {
117             int j = edge[i].to;
118             if(cnt[j] < K)
119                 heap.push({dist[j] + distance + edge[i].w, {distance + edge[i].w, j}});
120         }   //{当前距离 + 预计最短距离, {当前距离,  点}};
121     }
122     return -1;
123 }
124 int main() {
125     init();
126     scanf("%d%d", &n, &m);
127     for(int i = 1; i <= m; i ++) {
128         int x, y, w;
129         scanf("%d%d%d", &x, &y, &w);
130         add(head, x, y, w);     //正向建边 + 逆向建边, 方便Dijkstra
131         add(rhead, y, x, w);    //dist[k]内的值便代表 “终点” 到k点的最短距离, 用作启发函数
132     }
133     scanf("%d%d%d", &S, &T, &K);
134     if(S == T) K ++;            //至少含一条边, 所以起点终点相同时要去掉一种情况
135     Dijkstra();
136     printf("%d\n", astra());
137     return 0;
138 }
```

5.5 迭代加深

用 DFS 模仿 BFS 每次只搜索一层，尤其是对于分支特别多的搜索树，重复搜索相对于结果来说影响不大，但是 BFS 需要非常多的空间，DFS 可以节省空间记录下搜索路径
A* + 迭代加深 = IDA*

</> 代码 5.7: /搜索/迭代加深

```
const int N = 110;
int n;
int path[N];
bool dfs(int u, int k) {
    if (u == k) return path[u - 1] == n;
    bool st[N] = {0};
    for (int i = u - 1; i >= 0; i -- )
        for (int j = i; j >= 0; j -- ) {
            int s = path[i] + path[j];
            if (s > n || s <= path[u - 1] || st[s]) continue;
            st[s] = true;
            path[u] = s;
            if (dfs(u + 1, k)) return true;
        }
    return false;
}
int main() {
    path[0] = 1;
    while (cin >> n, n) {
        int k = 1;
        while (!dfs(1, k)) k ++ ; // 迭代加深，（如果没有搜到）每次深度 + 1;

        for (int i = 0; i < k; i ++ ) cout << path[i] << ' ';
        cout << endl;
    }
    return 0;
}
```

5.6 双向 DFS

一般用来处理“子集和”问题，从给定的 N 个数中选几个，使他们的和最接近 W，也可以认为是一个大背包问题，最暴力的解法是 $O(2^N)$ 的暴力枚举。利用双向搜索的思想，把物品分成两半，从前一半中选出若干，可能达到 0 W 之间的所有重量值存到一个数组 A 中，对数组排序，去重。然后进行第二次搜索，尝试从后一半礼物中选一些，对于每个可能达到的重量 t ，在第一部分得到的素组 A 中二分查找 $\leq W-t$ 的最大的一个，更新答案总复杂度接近 $O(N*x^{\frac{N}{2}})$

</> 代码 5.8: /搜索/双向 DFS

```
const int N = 1 << 25; // k最大是25， 因此最多可能有2^25种方案
```

```cpp
int n, m, k;
int g[50];      // 存储所有物品的重量
int weights[N]; // weights存储能凑出来的所有的重量
int cnt = 0;
int ans; // 用ans来记录一个全局最大值
// u表示当前枚举到哪个数了， s表示当前的和
void dfs(int u, int s) {
    // 如果我们当前已经枚举完第k个数（下标从0开始的）了， 就把当前的s，
    // 加到weights中去
    if (u == k) {
        weights[cnt++] = s;
        return;
    }
    // 枚举当前不选这个物品
    dfs(u + 1, s);
    // 选这个物品，做一个可行性剪枝
    if ((ll)s + g[u] <= m) { //计算和的时候转成long long防止溢出
        dfs(u + 1, s + g[u]);
    }
}
void dfs2(int u, int s) {
    if (u == n) { // 如果已经找完了n个节点， 那么需要二分一下
        int l = 0, r = cnt - 1;
        while (l < r) {
            int mid = (l + r + 1) >> 1;
            if (weights[mid] <= m - s)
                l = mid;
            else
                r = mid - 1;
        }
        ans = max(ans, weights[l] + s);
        return;
    }
    // 不选择当前这个物品
    dfs2(u + 1, s);
    // 选择当前这个物品
    if ((ll)s + g[u] <= m)
        dfs2(u + 1, s + g[u]);
}
int main() {
    cin >> m >> n;
    for (int i = 0; i < n; i++)
        cin >> g[i];
    // 优化搜索顺序（从大到小）
```

```
    sort(g, g + n);
    reverse(g, g + n);
    k = n / 2 + 2; // 把前k个物品的重量打一个表
    dfs(0, 0);
    // 做完之后， 把weights数组从小到大排序
    sort(weights, weights + cnt);
    // 判重
    int t = 1;
    for (int i = 1; i < cnt; i++)
        if (weights[i] != weights[i - 1])
            weights[t++] = weights[i];
    cnt = t;
    // 从k开始， 当前的和是0
    dfs2(k, 0);
    cout << ans << endl;
    return 0;
}
```

5.7 模拟退火

玄学算法，一般用在浮点数上，复杂度 $O(N^2)$
一般用于求某一连续多峰函数的最大值或者最小值

</> 代码 5.9: /搜索/模拟退火

```
typedef pair<double, double> PDD;
const int M = 110;
PDD p[M];
int n;
double ans = 1e8;
double rands(double l, double r){ //随机一个在 (l， r) 之间的小数
    return (double) rand() / RAND_MAX * (r - l) + l; // rand() / RAND_MAX (生成0, 1之间的小数) ;
}
double get_dist(PDD a, PDD b) { //求两点距离
    double dx = a.x - b.x, dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}
double calc(PDD k) {//求某点到各各点之间的距离
    double res = 0;
    for(int i = 0; i < n; i ++)
        res += get_dist(k, p[i]);
    ans = min(ans, res);
    return res;
}
```

```
20 void simulate_anneal() {
21     PDD point = {rands(0, 10000), rands(0, 10000)};
22     double last = calc(point);
23     for(double i = 1e4; i > 1e-4; i *= 0.99) {//范围 + 精度 + 退火系数（<=1，越接近1越精
       确，但复杂度也越大）
24 
25         PDD np(rands(point.x - i, point.x + i), rands(point.y - i, point.y + i)); //随机
       一个范围内的点
26         double now = calc(np);
27         double dt = now - last;
28         if (exp(-dt / i) > rands(0, 1)) {//物理公式，如果这个点比原来的点“差”，一定跳过这
       个点（范围缩小）
29                                         //如果这个点比原来的点“好”，则有一定概率跳过这个
       点（范围缩小）
30             point = np;
31             last = now;
32         }
33     }//通过退火过程，不断缩小范围，最后找到最值（但也有可能找不到，或者找错了）
34 }
35 
36 int main() {
37     srand(time(0));                //随机函数种子（运气）
38     scanf("%d", &n);
39     for(int i = 0; i < n; i ++) {
40         int x, y;
41         scanf("%d%d", &x, &y);
42         p[i] = {x, y};
43     }
44     for(int i = 0; i <= 10; i ++)  //退火过程多做几遍，使得答案正确可能性大一点
45         simulate_anneal();
46     printf("%.0lf", ans);          //四舍五入
47     return 0;
48 }
```

5.8 双向 DFS

</> 代码 5.10: /搜索/双向 DFS

```
1 const int N = 1 << 25; // k最大是25， 因此最多可能有2^25种方案
2 int n, m, k;
3 int g[50];      // 存储所有物品的重量
4 int weights[N]; // weights存储能凑出来的所有的重量
5 int cnt = 0;
6 int ans; // 用ans来记录一个全局最大值
```

```
// u表示当前枚举到哪个数了， s表示当前的和
void dfs(int u, int s) {
    // 如果我们当前已经枚举完第k个数（下标从0开始的）了， 就把当前的s，
    // 加到weights中去
    if (u == k) {
        weights[cnt++] = s;
        return;
    }
    // 枚举当前不选这个物品
    dfs(u + 1, s);
    // 选这个物品，做一个可行性剪枝
    if ((ll)s + g[u] <= m) { //计算和的时候转成long long防止溢出
        dfs(u + 1, s + g[u]);
    }
}
void dfs2(int u, int s) {
    if (u == n) { // 如果已经找完了n个节点， 那么需要二分一下
        int l = 0, r = cnt - 1;
        while (l < r) {
            int mid = (l + r + 1) >> 1;
            if (weights[mid] <= m - s)
                l = mid;
            else
                r = mid - 1;
        }
        ans = max(ans, weights[l] + s);
        return;
    }
    // 不选择当前这个物品
    dfs2(u + 1, s);
    // 选择当前这个物品
    if ((ll)s + g[u] <= m)
        dfs2(u + 1, s + g[u]);
}
int main() {
    cin >> m >> n;
    for (int i = 0; i < n; i++)
        cin >> g[i];
    // 优化搜索顺序（从大到小）
    sort(g, g + n);
    reverse(g, g + n);
    k = n / 2 + 2; // 把前k个物品的重量打一个表
    dfs(0, 0);
    // 做完之后， 把weights数组从小到大排序
```

```
51      sort(weights, weights + cnt);
52      // 判重
53      int t = 1;
54      for (int i = 1; i < cnt; i++)
55          if (weights[i] != weights[i - 1])
56              weights[t++] = weights[i];
57      cnt = t;
58      // 从k开始， 当前的和是0
59      dfs2(k, 0);
60      cout << ans << endl;
61      return 0;
62  }
```

第六章　图论

6.1　图论的基础

6.1.1　链式前向星

</> 代码 6.1:　/图论/init

```
struct Edge{
    int to,next,dis;
}edge[maxm]; //maxm为边的数量，无向图数目需乘二
int head[maxn],tot;
void init(){
    for(int i = 1; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
```

6.2　最短路

6.2.1　堆优化 Dijkstra

</> 代码 6.2:　/图论/Dijkstra

```
int dis[1005],vis[1005];
struct node{
    int pos,val;
    node(int pos,int val):pos(pos),val(val){}
    friend bool operator < (node a,node b){
        return a.val > b.val;
    }
};
int Dijkstra(int s,int t){
    for(int i = 1; i <= N ; i ++) dis[i] = INF;
```

```
    dis[s] = 0;
    priority_queue<node>Q;
    Q.push(node(s,0));
    while(!Q.empty()){
        node u = Q.top(); Q.pop();
        if(u.val > dis[u.pos]) continue;
        for(int i = head[u.pos]; ~i; i = edge[i].next){
            int v = edge[i].to;
            if(dis[v] > edge[i].dis + u.val){
                dis[v] = edge[i].dis + u.val;
                Q.push(node(v,dis[v]));
            }
        }
    }
    return dis[t];
}
```

6.2.2 SPFA

</> 代码 6.3: /图论/SPFA

```
//spfa判负环，spfa时间复杂度玄学，必要时手写队列以及打上快速读入，spfa容易被卡，非必要时用Dijkstra
bool vis[maxn];
int dis[maxn],cnt[maxn];
bool spfa(int s){
    for(int i = 1; i <= N ; i ++){
        vis[i] = 0; cnt[i] = 0; dis[i] = INF;
    }
    queue<int>Q;
    Q.push(s);
    vis[s] = 1; dis[s] = 0;
    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        vis[u] = 0;
        if(cnt[u] >= N) return true;
        for(int i = head[u]; ~i ; i = edge[i].next){
            int v = edge[i].to;
            if(dis[v] > dis[u] + edge[i].dis){
                dis[v] = dis[u] + edge[i].dis;
                if(!vis[v]){
                    Q.push(v);
                    vis[v] = 1;
                    cnt[v]++;
                    if(cnt[v] >= N) return true;
```

```
                }
            }
        }
    }
    return false;
}
```

6.2.3 Floyd

</> 代码 6.4: /图论/Floyd

```
//用于求任意两点间最短距离
//传递闭包: 所有两两之间的连通关系
for(int k = 1; k <= N ; k ++){
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= N; j ++){
            MAP[i][j] = min(MAP[i][j],MAP[i][k] + MAP[k][j]);
        }
    }
}
```

6.3 分层图

</> 代码 6.5: /图论/分层图

```
/*
    分层图思想, 其实就是在原图上建立很多的辅助结点
    然后用现有的算法 (例如最短路) 就可以简单的解决问题
    例如一个点买入东西在另一个点卖出且只能买入卖出一次的模型, 就可以建立三个图,
    每个图之间的边权都是0表示相互之间没有交易,
    图1和图2相同点之间边权为-val[i]表示买入
    图2和图3相同点之间边权为val[i]表示卖出
    最后图3上的结点的最长路就是经过一次买入卖出之后在这个节点上能赚到的最多钱
    分层图没有模板, 放在这里就是提供一个做题时候的思路
*/
```

6.4 差分约束

</> 代码 6.6: /图论/差分约束

```
/*差分约束系统: 如果一个系统由n个变量和m个约束条件组成, 其中每个约束条件形如 xj - xi<= bk (
    i , j  [1, n], k  [1, m]), 则称其为差分约束系统。
例如如下的约束条件:
X1 - X2 <= 0 X1 - X5 <= -1
```

```
X2 - X5 <= 1 X3 - X1 <= 5
X4 - X1 <= 4 X4 - X3 <= -1
X5 - X3 <= -3 X5 - X4 <= -3
全都是两个未知数的差小于等于某个常数（大于等于也可以，因为左右乘以-1就可以化成小于等于）。这样的不等式组就称作差分约束系统。
差分约束系统求解过程:
1.新建一个图，N个变量看作N个顶点，M个约束条件作为M条边。每个顶点Vi分别对于一个未知量，每个有向边对应两个未知量的不等式。
2.为了保证图的连通性，在图中新加一个节点Vs，图中每个节点Vi都能从Vs可达，建立边w(Vs, Vi) = 0。
3.对于每个差分约束Xj - Xi <= Bk(这里是小于等于号)，则建立边w(Xi, Xj) = Bk。
4.初始化Dist[] = INF, Dist[Vs] = 0.
5.求解以Vs为源点的单源最短路径，推荐用SPFA，因为一般可能存在负值。
如果图中存在负权回路，则该差分约束系统不存在可行解。
Vs到某点如果不存在最短路径，即最短路为INF，则对于该点表示的变量可以取任意值，都能满足差分约束的要求，如果存在最短路径，则得到该变量的最大值。
上述过程最终得到的解为满足差分约束系统各项的最大值。
注意点:
1. 如果要求最大值想办法把每个不等式变为标准 x - y <= k 的形式,然后建立一条从 y 到 x 权值为 k 的边，变得时候注意 x - y < k => x - y <= k-1。
2. 如果要求最小值的话，变为 x - y >= k 的标准形式，然后建立一条从 y到 x 权值为 k 的边，求出最长路径即可。
3. 如果权值为正，用Dijkstra，SPFA，BellmanFord都可以，如果为负不能用Dijkstra，并且需要判断是否有负环，有的话就不存在。*/
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#include<queue>
#define INF 0x7fffffff
using namespace std;
const int MAXN = 1100;
const int MAXM = 30030;

struct EdgeNode
{
    int to;
    int w;
    int next;
}Edges[MAXM];

int Head[MAXN],Dist[MAXN],vis[MAXN],outque[MAXN],id;

void AddEdges(int u,int v,int w)
```

```
{
    Edges[id].to = v;
    Edges[id].w = w;
    Edges[id].next = Head[u];
    Head[u] = id++;
}
void SPFA(int s,int N)
{
    int ans = 0;
    memset(vis,0,sizeof(vis));
    memset(outque,0,sizeof(outque));
    for(int i = 1; i <= N; ++i) Dist[i] = INF;
    Dist[s] = 0;
    vis[s] = 1;
    queue<int> Q;
    Q.push(s);
    while( !Q.empty() )
    {
        int u = Q.front();
        Q.pop();
        vis[u] = 0;
        outque[u]++;
        if(outque[u] > N+1) //如果出队次数大于N，则说明出现负环
        {
            ans = -1;
            break;
        }
        for(int i = Head[u]; i != -1; i = Edges[i].next){
            int temp = Dist[u] + Edges[i].w;
            if(temp < Dist[Edges[i].to]){
                Dist[Edges[i].to] = temp;
                if( !vis[Edges[i].to]){
                    vis[Edges[i].to] = 1;
                    Q.push(Edges[i].to);
                }
            }
        }
    }

    if(ans == -1)   //出现负权回路，不存在可行解
        printf("-1\n");
    else if(Dist[N] == INF) //可取任意值，都满足差分约束系统
        printf("-2\n");
    else
```

```
85          printf("%d\n",Dist[N]);  //求使得源点 s 到 终点 t 的最大的值
86  }
87
88  int main(){
89      int N,ML,MD,u,v,w;
90      while(~scanf("%d%d%d", &N, &ML, &MD)){
91          memset(Head,-1,sizeof(Head));
92          id = 0;
93          for(int i = 0; i < ML; ++i){
94              scanf("%d%d%d",&u,&v,&w);
95              AddEdges(u,v,w);//建边 u - v <= w
96          }
97          for(int i = 0; i < MD; ++i){
98              scanf("%d%d%d",&u,&v,&w);
99              AddEdges(v,u,-w);//建边 v - u <= w
100         }
101 //这里不加也可以
102 //        for(int i = 1; i < N; ++i)
103 //            AddEdges(i+1,i,0);
104         SPFA(1,N);  //求使得源点 s 到 终点 t 的最大的值
105     }
106     return 0;
107 }
```

6.5 欧拉路径

</> 代码 6.7: /图论/欧拉路径

```
1  //欧拉回路: 一笔画且回到起点
2  //欧拉路径 (欧拉通路) : 不要求回到起点
3  /*
4  无向图
5  G存在欧拉通路的充要条件是: G为连通图, 并且G仅有两个奇度结点 (度数为奇数的顶点) 或者无奇度结
       点。
6  推论
7   (1)  当G是仅有两个奇度结点的连通图时, G的欧拉通路必以此两个结点为端点;
8   (2) 当G是无奇度结点的连通图时, G必有欧拉回路
9   (3) G为欧拉图 (存在欧拉回路) 的充分必要条件是  G为无奇度结点的连通图
10 */
11 /*
12  (有向图)  定理
13 有向图D存在欧拉通路的充要条件是: D为有向图, D的基图连通, 并且所有顶点的出度与入度相等; 或者
       除两个顶点外, 其余顶点的出度与入度都相等, 而这两个顶点中一个顶点的出度与入度之差为1, 另
       一个顶点的出度与入度之差为-1.
```

```
推论
 (1) 当D除出、入度之差为1，-1的两个顶点之外，其余顶点的出度与入度相等时，D的有向欧拉通路必以
    出、入度之差为1的顶点作为始点，以出、入度之差为-1的顶点作为终点。
 (2) 当D的所有顶点的出、入度都相等时，D中存在有向欧拉回路。
 (3) 有向图D为有向欧拉图的充要条件是  D的基图为连通图，并且所有顶点的出、入度都相等。
*/
//模板：无向图找欧拉路径，字典序最小
//如果不要求字典序，可以采用邻接表
const int maxn = 510;
int N,M,K;
int MAP[maxn][maxn];
int ind[maxn];
vector<int>ans;
void dfs(int t){
    for(int i = 0 ; i < maxn; i ++){
        if(MAP[t][i]){
            MAP[t][i]--; MAP[i][t]--;
            dfs(i);
        }
    }
    ans.pb(t);
}
int main(){
    Sca(M); init();
    int root;
    for(int i = 1; i <= M ; i ++){
        int u,v; Sca2(u,v);
        MAP[u][v]++; MAP[v][u]++;
        ind[u]++; ind[v]++;
        root = v;
    }
    for(int i = 0 ; i < maxn; i ++){
        if(ind[i] & 1){
            root = i;
            break;
        }
    }
    dfs(root);
    for(int i = ans.size() - 1; i >= 0; i --){
        printf("%d\n",ans[i]);
    }
    return 0;
}
```

6.6 kruskal 重构树

</> 代码 6.8: /图论/kruskal 重构树

```
/*kruskal重构树
这个东西主要来处理最小生成树的最大边权问题
当然也可以处理最大生成树的最小边权问题  核心思想跟 krsskal差不多
我们重构树的过程是这样的
将所有边按边权从小到大排序
每次最小的一条边，如果条边相连的两个点在同一个集合中，那么就跳过，否则就将这两个点的祖先都连到一个虚点上去，让这个虚点的点权等于这条边的边权
这样的话这课被重构的树就有一些奇妙的性质
原本最小生成树上的点在重构树里都是叶节点
从任何一个点往根上引一条路径，这条路径经过的点的点权单调不降（最大生成树单调不升）
任意两点之间路径的最大边权就是他们的LCA的点权*/
//模板：给一张图，每次询问两点间路径最大值的最小值是多少
//做法1.最小生成树上LCA求树链最大值
//作法2(如下代码):kruskal重构树上直接求LCA点权
const int maxn = 3e5 + 10;
int N,M;
int tree[maxn];
int find(int t){
    if(tree[t] == t) return t;
    return tree[t] = find(tree[t]);
}
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1,tree[i] = i;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
struct E{
    int u,v,w;
}e[maxn];
bool cmp(E a,E b){
    return a.w < b.w;
}
int w[maxn];
```

```
41 const int SP = 20;
42 int pa[maxn][SP],dep[maxn];
43 void dfs(int u,int la){
44     pa[u][0] = la; dep[u] = dep[la] + 1;
45     for(int i = 1; i < SP; i ++){
46         pa[u][i] = pa[pa[u][i - 1]][i - 1];
47     }
48     for(int i = head[u]; ~i; i = edge[i].next){
49         int v = edge[i].to;
50         if(v == la) continue;
51         dfs(v,u);
52     }
53 }
54 int lca(int u,int v){
55     if(dep[u] < dep[v]) swap(u,v);
56     int t = dep[u] - dep[v];
57     for(int i = 0 ; i < SP; i ++) if(t & (1 << i)) u = pa[u][i];
58     for(int i = SP - 1; i >= 0; i --){
59         int uu = pa[u][i],vv = pa[v][i];
60         if(uu != vv){
61             u = uu;
62             v = vv;
63         }
64     }
65     return u == v?u:pa[u][0];
66 }
67 int main(){
68     scanf("%d%d",&N,&M); init();
69     for(int i = 1; i <= M ;  i++){
70         scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);
71     }
72     sort(e + 1,e + 1 + M,cmp);
73     for(int i = 1; i <= M ; i ++){
74         int u = find(e[i].u),v = find(e[i].v);
75         if(u == v) continue;
76         N++; tree[N] = N; w[N] = e[i].w; head[N] = -1;
77         tree[u] = N; tree[v] = N;
78         add(N,u); add(N,v);
79     }
80     for(int i = 1; i <= N ; i ++){
81         if(tree[i] == i) dfs(i,-1);
82     }
83     int Q; scanf("%d",&Q);
84     while(Q--){
```

```
        int u,v; scanf("%d%d",&u,&v);
        if(find(u) != find(v)) puts("impossible");
        else printf("%d\n",w[lca(u,v)]);
    }
    return 0;
}
```

6.7 Tarjan

6.7.1 割点

</> 代码 6.9: /图论/割点

```
//在无向图中，如果删除一个点这个图就不联通，那么这个点就叫割点
//模板：Tarjan求割点，N个点M条边找有多少割点并输出
int Low[maxn],dfn[maxn],Stack[maxn];
int Index,top;
bool Instack[maxn];
bool cut[maxn]; //记录是否为割点
int add_block[maxn];   //删除这个点之后增加的联通块
void Tarjan(int u,int la){
    int v;
    Low[u] = dfn[u] = ++Index;
    Stack[++top] = u;
    Instack[u] = true;
    int son = 0;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        if(!dfn[v]){
            son++;
            Tarjan(v,u);
            if(Low[u] > Low[v]) Low[u] = Low[v];
            if(u != la && Low[v] >= dfn[u]){
                cut[u] = 1;
                add_block[u]++;
            }
        }else if(Low[u] > dfn[v]) Low[u] = dfn[v];
    }
    if(u == la && son > 1) cut[u] = 1;
    if(u == la) add_block[u] = son - 1;
    Instack[u] = false;
    top--;
}
int solve(){
```

```
33      for(int i = 0 ; i <= N + 1; i ++)cut[i] = add_block[i] = dfn[i] = Instack[i] = 0;
34      Index = top = 0;
35      int ans = 0;
36      for(int i = 1; i <= N ; i ++) if(!dfn[i]) Tarjan(i,i);
37      for(int i = 1; i <= N ; i ++) ans += cut[i];
38      return ans;
39  }
40  int main(){
41      N = read(); M = read(); init();
42      for(int i = 1; i <= M ; i ++){
43          int u = read(), v = read();
44          add(u,v); add(v,u);
45      }
46      Pri(solve());
47      for(int i = 1; i <= N ; i ++) if(cut[i]) printf("%d ",i);
48      return 0;
49  }
```

6.7.2 缩点

</> 代码 6.10: /图论/缩点

```
1   //Tarjan缩点
2   /* 1.有向图缩点
3   将有向图中的所有强连通分量缩成一个点
4   缩完点之后原图就变成了一个DAG图（有向无环图）
5   模板例题
6       给定一个n个点m条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只
        需要求出这个权值和。
7   允许多次经过一条边或者一个点，但是，重复经过的点，权值只计算一次。
8   解法：对整个图缩完点之后跑拓扑排序DP一下即可
9   */
10  const int maxn = 1e4 + 10;
11  const int maxm = 1e5 + 10;
12  int val[maxn];
13  int Low[maxn],dfn[maxn],Stack[maxn],Belong[maxn],num[maxn];
14  int Index,top,scc;
15  bool Instack[maxn];
16  void Tarjan(int u){
17      int v;
18      Low[u] = dfn[u] = ++Index;
19      Stack[top++] = u;
20      Instack[u] = true;
21      for(int i = head[u]; ~i; i = edge[i].next){
22          v = edge[i].to;
```

```
        if(!dfn[v]){
            Tarjan(v);
            if(Low[u] > Low[v]) Low[u] = Low[v];
        }else if(Instack[v] && Low[u] > dfn[v]) Low[u] = dfn[v];
    }
    if(Low[u] == dfn[u]){
        scc++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
            num[scc]++;
        }while(v != u);
    }
}
int VAL[maxn],ind[maxn];
int dp[maxn];
vector<int>P[maxn];
void solve(){
    for(int i = 1; i <= N ; i ++) dfn[i] = Instack[i] = num[i] = 0;
    Index = scc = top = 0;
    for(int i = 1; i <= N ; i ++) if(!dfn[i]) Tarjan(i);   //将原图缩点为DAG图
    for(int i = 1; i <= N ; i ++){                         //建DAG图
        VAL[Belong[i]] += val[i];
        for(int j = head[i]; ~j; j = edge[j].next){
            int v = edge[j].to;
            if(Belong[v] != Belong[i]){
                P[Belong[i]].pb(Belong[v]);
                ind[Belong[v]]++;
            }
        }
    }
    queue<int>Q;
    for(int i = 1; i <= scc;i ++){
        if(!ind[i]) Q.push(i);
        dp[i] = VAL[i];
    }
    int ans = 0;
    while(!Q.empty()){            //拓扑排序
        int u = Q.front(); Q.pop();
        ans = max(ans,dp[u]);
        for(int i = 0 ; i < P[u].size(); i ++){
            int v = P[u][i];
            ind[v]--;
```

```
            dp[v] = max(dp[v],dp[u] + VAL[v]);
            if(!ind[v]) Q.push(v);
        }
    }
    Pri(ans);

}
int main(){
    N = read(); M = read(); init();
    for(int i = 1; i <= N ; i ++) val[i] = read();
    for(int i = 1; i <= M ; i ++){
        int u = read(),v = read();
        add(u,v);
    }
    solve();
    return 0;
}
//2.无向图缩点
//将所有含有无向图环的点缩点，最终形成一棵树
//模板:找一条路使得路上的点权最大，同一个点只能计入一次贡献，要求不能走回头路
//做法:如果成环那么整个环都可以走且可以走回来环的路上，那么缩点形成一棵树之后树形dp
const int maxn = 4e5 + 10;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
LL w[maxn];
int belong[maxn],cnt;
int dfn[maxn],low[maxn],num;
LL size[maxn],weight[maxn];//size表示缩点后的点内点的数量，weight表示缩点后的点的点权和
int Stack[maxn],top;
bool Instack[maxn];
void tarjan(int u,int la){
    dfn[u] = low[u] = ++num;
```

```
111     Stack[++top] = u;
112     Instack[u] = true;
113     int v;
114     for(int i = head[u]; ~i; i = edge[i].next){
115         v = edge[i].to;
116         if(v == la) continue;
117         if(!dfn[v]){
118             tarjan(v,u);
119             low[u] = min(low[u],low[v]);
120         }else{
121             low[u] = min(low[u],dfn[v]);
122         }
123     }
124     if(low[u] == dfn[u]){
125         v = Stack[top];
126         ++cnt;
127         while(v != u){
128             belong[v] = cnt;
129             weight[cnt] += w[v];
130             size[cnt]++;
131             Instack[v] = false;
132             v = Stack[--top];
133         }
134         --top;
135         belong[u] = cnt;
136         weight[cnt] += w[u];
137         size[cnt]++;
138         Instack[u] = false;
139     }
140 }
141 vector<int>P[maxn];
142 bool vis[maxn],ret[maxn];
143 LL dp[maxn],dp2[maxn];
144 void dfs(int u,int la){
145     dp2[u] = weight[u];
146     for(int i = 0 ; i < P[u].size(); i ++){
147         int v = P[u][i];
148         if(v != la && !vis[v]){
149             vis[v] = 1; dfs(v,u);
150             if(ret[v]) ret[u] = 1;
151             dp[u] += dp[v];
152             dp2[u] += dp[v];
153         }
154     }
```

```
155     if(size[u] > 1) ret[u] = 1;
156     if(ret[u]) dp[u] += weight[u];
157     LL t = dp2[u];
158     for(int i = 0 ; i < P[u].size(); i ++){
159         int v = P[u][i];
160         if(v == la) continue;
161         dp2[u] = max(dp2[u],dp2[v] + t - dp[v]);
162     }
163 }
164 int main(){
165     Sca2(N,M); init();
166     for(int i = 1; i <= N ; i ++) Scl(w[i]);
167     for(int i = 1; i <= M ; i ++){
168         int u,v; Sca2(u,v); add(u,v); add(v,u);
169     }
170     for(int i = 1; i <= N ; i ++) if(!dfn[i]) tarjan(i,i);
171     for(int u = 1; u <= N ; u ++){
172         for(int i = head[u]; ~i; i = edge[i].next){
173             int v = edge[i].to;
174             if(belong[u] == belong[v]) continue;
175             P[belong[u]].push_back(belong[v]);
176         }
177     }
178     int S = read(); vis[belong[S]] = 1;
179     dfs(belong[S],-1);
180     Prl(dp2[belong[S]]);
181     return 0;
182 }
```

6.7.3 点双

</> 代码 6.11: /图论/点双

```
1  //点的双联通分量
2  //一个割点可能被多个点双包含，一个割边不会被多个边双包含
3  const int maxn = 5e5 + 10;
4  const int INF = 0x3f3f3f3f;
5  const int mod = 998244353;
6  int N,M,K;
7  struct Edge{
8      int to,next;
9      bool vis;
10 }edge[maxn * 2];
11 int head[maxn],tot;
12 void init(){
```

```
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].vis = 0;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int Low[maxn],dfn[maxn],Stack[maxn];
bool Instack[maxn],cut[maxn];
int Index,top;
int dcn; //点双的个数
vector<int>dcc[maxn];//每个点双含有的点
void Tarjan(int u,int la){
    Low[u] = dfn[u] = ++Index;
    Stack[++top] = u;
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        if(!dfn[v]){
            Tarjan(v,u);
            if(Low[u] > Low[v]) Low[u] = Low[v];
            if(Low[v] >= dfn[u]){
                cut[u] = 1;
                dcn++;
                dcc[dcn].clear();
                dcc[dcn].pb(u);
                while(1){
                    int w = Stack[top--];
                    dcc[dcn].pb(w);
                    if(w == v) break;
                }
            }
        }else if(Low[u] > dfn[v]) Low[u] = dfn[v];
    }
}
LL q_p(LL a,LL b){
    LL ans = 1;
    while(b){
        if(b & 1) ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
```

```
    return ans;
}
int main(){
    Sca2(N,M); init();
    for(int i = 1; i <= M ; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    dcn = Index = top = 0;
    for(int i = 1; i <= N; i ++){
        Low[i] = dfn[i] = Stack[i] = Instack[i] = cut[i] = 0;
    }
    LL sum = 1;
    int cnt = 0;
    for(int i = 1; i <= N ; i ++){
        if(!dfn[i]) Tarjan(i,-1);
    }
    for(int i = 1; i <= dcn; i ++){
        int v = dcc[i].size();
        if(v == 2) cnt++;
        else if(v > 2) sum = sum * (q_p(2,v) - 1 + mod) % mod;
    }
    sum = sum * q_p(2,cnt) % mod;
    Prl(sum);

    return 0;
}
```

6.8 2-SAT

6.8.1 缩点法

</> 代码 6.12: /图论/2-SAT 缩点法

```
//2-SAT
/*
2—SAT:给出n个布尔变量以及一堆约束，让你寻找其中的可行解
约束的方式形如1.必须(不)选a 2.a与b必须选一个(不能都选) 3.a,b选择情况相同(相反)等等最多由两个变量建立起的约束。
对于这一类的问题，考虑用图论的方式解决，将一个点i拆点为i与i + N，i表示这个值为1，i + N表示为0
a -> b的边表示如果a则b
构图方法：以下a表示a这个点，a'表示a + N这个点
1.必选a: a'->a      2.必不选a:a->a'
2.a,b中选择一个:a'->b,b'->a     3.a,b不都选:a->b',b->a'
```

```
4.a,b选择情况相同 a->b,b->a,a'->b',b'->a'
5.a,b选择情况相反 a->b',b->a',a'->b,b'->a
诸如此类。
*/
/*     Tarjan缩点法
如果题目不要求字典序最小，只要求输出任意一组，我们可以考虑Tarjan缩点的做法
注意到同一个强连通分量里面的点必定互相满足，也就是说如果任意存在一个i使得i 和 i'在一个强连通分量内，
则说明题目无解，因为不可能一个点即为false又为true。
其次，缩点之后会形成一个DAG图，如果拓扑序排在前面的满足，那么拓扑序排在后面的也需要满足。
为了防止出现形如i为i'的前驱，选了i之后会满足i'成立这样的错误情况出现，对于每个点都选取拓扑序较后面的那一个
也就是如果i -> i'，则选i为false一定不会错，但这样的操作无法满足字典序最小，如果要求字典序，需要采用染色法
我们不必特意去跑拓扑，因为Tarjan标记出的强连通分量编号满足逆拓扑序，编号较小的点即为拓扑序在后面的点。
*/
const int maxn = 2e6 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn << 1];
int head[maxn],tot;
void init(){
    for(int i = 0; i <= (N << 1); i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int Low[maxn],dfn[maxn],Stack[maxn],Belong[maxn],num[maxn];
int Index,top,scc;
bool Instack[maxn];
void Tarjan(int u){
    int v;
    Low[u] = dfn[u] = ++Index;
    Stack[top++] = u;
    Instack[u] = true;
    for(int i = head[u]; ~i ; i = edge[i].next){
        v = edge[i].to;
```

```
        if(!dfn[v]){
            Tarjan(v);
            if(Low[u] > Low[v]) Low[u] = Low[v];
        }else if(Instack[v] && Low[u] > dfn[v]) Low[u] = dfn[v];
    }
    if(Low[u] == dfn[u]){
        scc++;
        do{
            v = Stack[--top];
            Instack[v] = false;
            Belong[v] = scc;
            num[scc]++;
        }while(v != u);
    }
}
int main(){
    Sca2(N,M); init();
    for(int i = 1; i <= M ; i ++){
        int x,a,y,b;
        scanf("%d%d%d%d",&x,&a,&y,&b);
        add(x + a * N,y + (b ^ 1) * N);
        add(y + b * N,x + (a ^ 1) * N);
    }
    for(int i = 1; i <= (N << 1); i ++) if(!dfn[i]) Tarjan(i);
    for(int i = 1; i <= N; i ++){
        if(Belong[i] == Belong[i + N]){
            puts("IMPOSSIBLE");
            return 0;
        }
    }
    puts("POSSIBLE");
    for(int i = 1; i <= N ; i ++){
        if(Belong[i] < Belong[i + N]) printf("1 ");
        else printf("0 ");
    }
    return 0;
}
```

6.8.2 染色法

</> 代码 6.13: /图论/2-SAT 染色法

```
//染色法
/* 当题目对字典序有要求的时候，就只能采用比较朴素的方法
在原来的图上，很显然相邻的点不能同时取到，那就是寻找一个字典序最小的染色方法
```

```
从小到大枚举点，如果能给这个点涂色就涂上，否则给i'涂色，如果这也不行，则没有可行方案
*/
//模板：1，2//3，4//5，6为各自组的点，为了使得i' = i ^ 1,将所有编号--，变为0,1//2,3//4,5
//给出的约束是u,v两点不能同时涂色
const int maxn = 20010;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn << 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= (N << 1) ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int color[maxn],top;
int Stack[maxn];
bool dfs(int t){
    if(color[t]) return true;
    if(color[t ^ 1]) return false;
    color[t] = 1;
    Stack[++top] = t;
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(!dfs(v)) return false;
    }
    return true;
}
bool solve(){
    for(int i = 0 ; i <= (N << 1); i ++) color[i] = 0;
    for(int i = 0; i < (N << 1) ; i += 2){
        if(!color[i] && !color[i + 1]){
            top = 0;
            if(dfs(i)) continue;
            for(int j = 1; j <= top; j ++) color[Stack[j]] = 0;
            if(!dfs(i + 1)) return false;
        }
    }
```

```
48      return true;
49  }
50  int main(){
51      while(~Sca2(N,M)){
52          init();
53          for(int i = 1; i <= M ; i ++){
54              int u,v; Sca2(u,v);
55              u--; v--;
56              add(u,v ^ 1); add(v,u ^ 1);
57          }
58          if(solve()){
59              for(int i = 0; i < (N << 1); i ++){
60                  if(color[i]) Pri(1 + i);
61              }
62          }else{
63              puts("NIE");
64          }
65      }
66      return 0;
67  }
```

6.9 斯坦纳树

</> 代码 6.14: /图论/斯坦纳树

```
1  /*  斯坦纳树
2  在一张图上寻找一颗边权和最小的生成树将指定的点集合连起来。
3  最小生成树事实上是斯坦纳树的一种特殊形式
4  一般给的点集不会过大(n <= 16左右)
5  求解方式是状压dp,dp[i][1 << k]表示在i点并且已经连起来了state状态的情况下的最小值
6  更新的状态用两种
7  1.相同点的两个子状态合并 dp[i][sta] = min(dp[i][sta1] + dp[i][sta2])
8  2.点与点之间的松弛 dp[i][sta] = min(dp[j][sta] + dis[i][j])
9  做法
10 (1)所以我们从小到大枚举状态
11 (2)先转移第一种: 在相同状态下先利用较小的状态更新过来
12 (2)更新完了之后利用SPFA松弛, 注意SPFA不用考虑进阶到下一状态, 只要保证当前状态最优就可以了
13  */
14  //luogoP4294 游览计划
15  /*
16  题意:给出一张网格图, 点权为正代表联通这个点的代价, 点权为0代表需要被联通的点集
17  求最小联通所有点集中的点的代价以及输出需要被联通的点,输出'x'表示
18   */
19 const int maxn = 12;
```

```
20 const int INF = 0x3f3f3f3f;
21 int N,M,K;
22 int MAP[maxn][maxn],id[maxn][maxn];
23 int dp[maxn][maxn][(1 << 11) + 10];
24 char ans[maxn][maxn];
25 struct node{
26     int x,y,state,val;
27     node(){}
28     node(int x,int y,int state,int val = 0):x(x),y(y),state(state),val(val){}
29     friend bool operator < (node a,node b){
30         return a.val > b.val;
31     }
32 };
33 node pre[maxn][maxn][(1 << 11) + 10];
34 const int a[4][2] = {0,1,1,0,0,-1,-1,0};
35 int cnt = 0;
36 bool check(int x,int y){
37     return 0 <= x && x < N && 0 <= y && y < M;
38 }
39 void dfs(int x,int y,int state){
40     if(x == N) return;
41     if(MAP[x][y]) ans[x][y] = 'o';
42     else ans[x][y] = 'x';
43     node u = pre[x][y][state];
44     dfs(u.x,u.y,u.state);
45     if(u.x == x && u.y == y) dfs(u.x,u.y,state - u.state); //说明是两个子集合并更新过来的
46 }
47 queue<PII>Q;
48 bool vis[maxn][maxn];
49 int limit;
50 void SPFA(int state){   //不要去更新下一层，只要把同一state的更新掉就可以了
51     while(!Q.empty()){
52         PII u = Q.front(); Q.pop();
53         vis[u.fi][u.se] = 0;
54         for(int i = 0 ; i < 4; i ++){
55             PII h = u;
56             h.fi += a[i][0]; h.se += a[i][1];
57             if(!check(h.fi,h.se)) continue;
58             if(dp[h.fi][h.se][state] > dp[u.fi][u.se][state] + MAP[h.fi][h.se]){
59                 dp[h.fi][h.se][state] = dp[u.fi][u.se][state] + MAP[h.fi][h.se];
60                 pre[h.fi][h.se][state] = node(u.fi,u.se,state);
61                 if(!vis[h.fi][h.se]){
62                     Q.push(h);
63                     vis[h.fi][h.se] = 1;
```

```
64                  }
65              }
66          }
67      }
68  }
69  void solve(){
70      for(int i = 0 ; i < N ; i ++){
71          for(int j = 0 ; j < M ; j ++){
72              if(!MAP[i][j]){
73                  Pri(dp[i][j][limit]);
74                  dfs(i,j,limit);
75                  return;
76              }
77          }
78      }
79  }
80  int main(){
81      Sca2(N,M); cnt = 0;
82      for(int i = 0; i < N ; i ++){
83          for(int j = 0; j < M; j ++){
84              Sca(MAP[i][j]); ans[i][j] = '_';
85              if(!MAP[i][j]) id[i][j] = cnt++;
86          }
87      }
88      limit = (1 << cnt) - 1; //1111111
89      //初始化
90      for(int i = 0; i < N ; i ++){
91          for(int j = 0; j < M ; j ++){
92              for(int k = 0 ; k <= limit; k ++) dp[i][j][k] = INF;
93              dp[i][j][0] = 0; pre[i][j][0] = node(N,M,-1);
94              if(!MAP[i][j]){
95                  dp[i][j][1 << id[i][j]] = 0;
96                  pre[i][j][1 << id[i][j]] = node(N,M,-1);
97              }
98          }
99      }
100     //状态转移
101     for(int state = 0 ; state <= limit; state ++){ //从小到大枚举状态
102         for(int i = 0 ; i < N ; i ++){
103             for(int j = 0 ; j < M ; j ++){
104                 for(int sta = state;sta; sta = (sta - 1) & state){    //枚举state的所有子
    集
105                     if(dp[i][j][sta] + dp[i][j][state - sta] - MAP[i][j] < dp[i][j][
    state]){
```

```
106                         dp[i][j][state] = dp[i][j][sta] + dp[i][j][state - sta] - MAP[i
    ][j]; //子集间的更新
107                         pre[i][j][state] = node(i,j,sta); //因为要输出方案，所以记录前驱
108                     }
109                 }
110                 if(dp[i][j][state] < INF){
111                     Q.push(mp(i,j));
112                     vis[i][j] = 1;
113                 }
114             }
115         }
116         SPFA(state); //当前状态全部从子状态更新上来之后SPFA松弛到最优
117     }
118     solve();
119     for(int i = 0 ; i < N ; i ++){
120         for(int j = 0 ; j < M; j ++){
121             printf("%c",ans[i][j]);
122         }
123         puts("");
124     }
125     return 0;
126 }
```

6.10 二分图匹配

</> 代码 6.15: /图论/二分图匹配

```
1  //二分图匹配 匈牙利算法
2  //模板：左边N个点匹配右边M个点，总共K条边的最大匹配数
3  const int maxn = 1010;
4  int linker[maxn];
5  int vis[maxn];
6  bool dfs(int u){
7      for(int i = head[u]; ~i ; i = edge[i].next){
8          int v = edge[i].to;
9          if(!vis[v]){
10             vis[v] = true;
11             if(linker[v] == -1 || dfs(linker[v])){
12                 linker[v] = u;
13                 return true;
14             }
15         }
16     }
17     return false;
```

```
}
int hungary(){
    int ans = 0;
    for(int i = 1; i <= M ; i ++) linker[i] = -1;   //注意是右边点的linker
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++) vis[j] = 0;
        if(dfs(i)) ans++;
    }
    return ans;
}
int main(){
    Sca2(N,M); K = read(); init();
    for(int i = 1; i <= K; i ++){
        int u,v; Sca2(u,v);
        if(u > N || v > M) continue;
        add(u,v);
    }
    Pri(hungary());
    return 0;
}
```

6.11 KM 算法

</> 代码 6.16: /图论/KM 算法

```
/*KM算法
二分图最大权匹配，时间复杂度O(n³)
1.最大权匹配不一定是最大匹配
2.若要求最小权匹配，就边权取反然后求最大权，将答案取反
3.对于顶标lx[i],ly[i]的理解
lx初始为i点连出去的最大的边权，ly初始为0
可行顶标：在任意时刻，对于任意边(u,v,w)都满足lx[u] + ly[v] >= w
同时，满足lx[u] + ly[v] >= MAP[u][v]约束下的最小顶标和( lx + ly)就是最大权匹配
满足lx[u] + ly[v] <= MAP[u][v]约束下的最大顶标和就是最小权匹配
*/
//模板：求最小权匹配
const int maxn = 210;
const LL INF = 1e18;
LL MAP[maxn][maxn];
LL vx[maxn],vy[maxn],lx[maxn],ly[maxn],slack[maxn];
int pre[maxn],Left[maxn],Right[maxn],NL,NR,Nm,N;
void match(int &u){
    for(;u;swap(u,Right[pre[u]])) Left[u] = pre[u];
}
```

```
20 void bfs(int u){
21     static int q[maxn],front,rear;
22     front = 0; vx[q[rear = 1] = u] = true;
23     while(1){
24         while(front < rear){
25             int u = q[++front];
26             for(int v = 1; v <= Nm; v ++){
27                 int tmp;
28                 if(vy[v] || (tmp = lx[u] + ly[v] - MAP[u][v]) > slack[v]) continue;
29                 pre[v] = u;
30                 if(!tmp){
31                     if(!Left[v]) return match(v);
32                     vy[v] = vx[q[++rear] = Left[v]] = true;
33                 }else slack[v] = tmp;
34             }
35         }
36         LL a = INF;
37         for(int i = 1; i <= Nm ; i ++)
38             if(!vy[i] && a > slack[i]) a = slack[u = i];
39         for(int i = 1; i <= Nm ; i ++){
40             if(vx[i]) lx[i] -= a;
41             if(vy[i]) ly[i] += a;
42             else slack[i] -= a;
43         }
44         if(!Left[u]) return match(u);
45         vy[u] = vx[q[++rear] = Left[u]] = true;
46     }
47 }
48 void solve(){
49     for(int i = 1; i <= Nm; i ++){
50         for(int j = 1; j <= Nm; j++){
51             slack[j] = INF;
52             vx[j] = vy[j] = false;
53         }
54         bfs(i);
55     }
56 }
57 LL KM(int nl,int nr){
58     NL = nl; NR = nr;
59     Nm = max(NL,NR);
60     for(int i = 1; i <= Nm; i ++){
61         lx[i] = -INF; ly[i] = 0;
62         pre[i] = Left[i] = Right[i] = 0;
63         for(int j = 1; j <= Nm; j ++){
```

```
64              lx[i] = max(lx[i],MAP[i][j]);
65          }
66      }
67      solve();
68      LL ans = 0;
69      for(int i = 1; i <= Nm; i ++) ans += lx[i] + ly[i];
70      return ans;
71  }
72  void output(){ //输出左边和右边匹配的点
73      for(int i = 1; i <= NL; i ++){
74          printf("%d ",(MAP[i][Right[i]]?Right[i]:0));
75      }
76      puts("");
77  }
78  int main(){
79      int T; scanf("%d",&T); int CASE = 1;
80      while(T--){
81          scanf("%d",&N);
82          for(int i = 1; i <= N ; i ++){
83              for(int j = 1; j <= N ; j ++){
84                  scanf("%lld",&MAP[i][j]);
85                  MAP[i][j] = -MAP[i][j];
86              }
87          }
88          printf("Case #%d: %lld\n",CASE++,-KM(N,N));
89      }
90      return 0;
91  }
```

6.12 最大流

6.12.1 Dinic

</> 代码 6.17: /图论/Dinic

```
1  //最大流 dinic算法 N个点M条边求S到T的最大流
2  //BFS将残余网络分层，DFS在dep[v] = dep[u] + 1的限制下找增广路
3  //注意：如果dinic写T了，要关注dfs里有没有用cur数组优化
4  const int maxn = 10010;
5  const int maxm = 100000;
6  const int INF = 0x3f3f3f3f;
7  int N,M,K;
8  struct Dinic{
9      struct Edge{
10         int from,to,next,cap,flow;
```

```
11          Edge(){}
12          Edge(int from,int to,int next,int cap,int flow):from(from),to(to),next(next),cap
    (cap),flow(flow){}
13      }edge[maxm * 2];     //记得 * 2
14      int n,s,t,head[maxn],tot;
15      int dep[maxn],cur[maxn];
16      void init(int n,int s,int t){
17          this->n = n; this->s = s; this->t = t; //记得三个this都要
18          tot = 0;
19          for(int i = 0 ; i <= n ; i ++) head[i] = -1;
20      }
21      inline void AddEdge(int s,int t,int w){
22          edge[tot] = Edge(s,t,head[s],w,0);
23          head[s] = tot++;
24          edge[tot] = Edge(t,s,head[t],0,0);
25          head[t] = tot++;
26      }
27      inline bool BFS(){
28          for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
29          dep[s] = 1;
30          queue<int>Q; Q.push(s);
31          while(!Q.empty()){
32              int u = Q.front(); Q.pop();
33              for(int i = head[u]; ~i ; i = edge[i].next){
34                  int v = edge[i].to;
35                  if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
36                  dep[v] = dep[u] + 1;
37                  Q.push(v);
38              }
39           }
40           return ~dep[t];
41      }
42      inline int DFS(const int& u,int a){
43          if(u == t || !a) return a;
44          int flow = 0;
45          for(int &i = cur[u]; ~i ; i = edge[i].next){
46              int v = edge[i].to;
47              if(dep[v] != dep[u] + 1) continue;
48              int f = DFS(v,min(a,edge[i].cap - edge[i].flow));
49              if(!f) continue;
50              edge[i ^ 1].flow -= f;
51              edge[i].flow += f;
52              a -= f;
53              flow += f;
```

```
        }
        return flow;
    }
    inline int maxflow(){
        return maxflow(s,t);
    }
    inline int maxflow(int s,int t){
        int flow = 0;
        while(BFS()){
            for(int i = 0; i <= n ; i ++) cur[i] = head[i];
            flow += DFS(s,INF);
        }
        return flow;
    }
}g;
int main(){
    int S,T;
    Sca2(N,M); Sca2(S,T);
    g.init(N,S,T);
    while(M--){
        int u,v,w; Sca3(u,v,w);
        g.AddEdge(u,v,w);
    }
    Pri(g.maxflow());
    return 0;
}
```

6.12.2 最小割

</> 代码 6.18: /图论/最小割

```
/*          最小割
最小割边：为了使原点（记为S）和汇点（记为T）不连通，最少要割几条边
最小割 = 最大流，求最大流即可

最小割点
删去最少的点使原图不联通
假设原来的点编号为i，总共有n个点，那么我们就把每个点拆成两个点，编号分别为i和i+n。
其中点i负责连接原图中连入这个点的边，点i+n负责连原图中连出这个点的边。
i和i + n之间有一条容量为1的边，其余边都为INF
删点的过程相当于删去i 到 i + n之间这条容量为1的边，使得其余点都不能通过这个点到其他的点
如果有不能删去的点，就在把i 到 i + n之间的容量变为INF
*/
//模板：求起点到终点的最小割点
const int maxn = 1010;
```

```
15 const int maxm = 3010;
16 const int INF = 0x3f3f3f3f;
17 int N,M,K;
18 struct Dinic{
19     struct Edge{
20         int to,next,cap,flow;
21         Edge(){}
22         Edge(int to,int next,int cap,int flow):to(to),next(next),cap(cap),flow(flow){}
23     }edge[maxm * 2];
24     int head[maxn],tot;
25     int pre[maxn],s,t,n,dis[maxn];
26     void init(int n){
27         this->n = n;
28         for(int i = 0 ; i <= n ; i ++) head[i] = -1;
29         tot = 0;
30     }
31     void add(int u,int v,int w){
32         edge[tot] = Edge(v,head[u],w,0);
33         head[u] = tot++;
34         edge[tot] = Edge(u,head[v],0,0);
35         head[v] = tot++;
36     }
37     bool BFS(){
38         for(int i = 0 ; i <= n ; i ++) dis[i] = -1;
39         dis[s] = 0;
40         queue<int>Q; Q.push(s);
41         while(!Q.empty()){
42             int u = Q.front(); Q.pop();
43             for(int i = head[u]; ~i ; i = edge[i].next){
44                 int v = edge[i].to;
45                 if(~dis[v] || edge[i].cap <= edge[i].flow) continue;
46                 dis[v] = dis[u] + 1;
47                 Q.push(v);
48             }
49         }
50         return ~dis[t];
51     }
52     int DFS(int u,int a){
53         if(u == t || !a) return a;
54         int flow = 0;
55         for(int& i = pre[u]; ~i ; i = edge[i].next){
56             int v = edge[i].to;
57             if(dis[v] == dis[u] + 1){
58                 int f = DFS(v,min(a,edge[i].cap - edge[i].flow));
```

```
                if(!f) continue;
                flow += f;
                a -= f;
                edge[i].flow += f;
                edge[i ^ 1].flow -= f;
            }
        }
        return flow;
    }
    int maxflow(int s,int t){
        this->s = s; this->t = t;
        int flow = 0;
        while(BFS()){
            for(int i = 0 ; i <= n; i ++) pre[i] = head[i];
            flow += DFS(s,INF);
        }
        return flow;
    }
}g;
int main(){
    Sca2(N,M); g.init(N + N);
    int S,T; Sca2(S,T);
    for(int i = 1; i <= N ; i ++) g.add(i,i + N,1);
    g.add(S,S + N,INF);
    g.add(T,T + N,INF);
    for(int i = 1; i <= M; i ++){
        int u,v; Sca2(u,v);
        g.add(u + N,v,INF);
        g.add(v + N,u,INF);
    }
    Pri(g.maxflow(S,T + N));
    return 0;
}
```

6.12.3 最大权闭合子图

</> 代码 6.19: /图论/最大权闭合子图

```
/*      最大权闭合子图
对于一个图而言，如果一个子图中所有点都满足这个点的后继点都在子图内，这就是一个闭合子图
满足点权和最大的闭合子图就是最大权闭合子图
例如有一些实验，完成一个实验需要有一些器材，实验会赚钱器材会花钱，
寻找一个最赚钱的买器材做实验的方案，答案就是最大权闭合子图的总和
做实验赚钱的点权为正，指向那些需要的器材，器材点权为负
最大权闭合子图的解法：
```

```
建立源点S连接到所有正点权上，容量为点权值
建立汇点T被所有负点权连接，容量大小为点权值取绝对值
原本的点之间的边照样建，容量为INF
然后答案就是所有正权值的和 - S到T的最大流

理解：假设一开始就把所有的实验经费都收入囊中，然后我们需要放弃一些来满足调节。
要么选择不做这个实验，要么选择购买这些器材，使得满足条件的最小花费就是这个图的最小割

方案：在跑完最大流之后的残余网络中，依然和源点S相连的所有点就是必须选择的器材和必须做的实验
*/
//例题洛谷 P 2762：跑一个图中的最大权闭合子图并输出方案
//1 - N为正权点，N + 1 - N + M为负权点
const int maxn = 1010;
const int maxm = 10010;
const int INF = 0x3f3f3f3f;
int N,M,K;
int vis[maxn];
struct Dinic{
    struct Edge{
        int to,next,cap,flow;
        Edge(){}
        Edge(int to,int next,int cap,int flow):to(to),next(next),cap(cap),flow(flow){}
    }edge[maxn * 2];
    int head[maxn],dis[maxn],tot,pre[maxn];
    int n,s,t;
    void init(int n,int s,int t){
        this->n = n; this->s = s; this->t = t;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w){
        edge[tot] = Edge(v,head[u],w,0);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0);
        head[v] = tot++;
    }
    bool BFS(){
        for(int i = 0 ; i <= n ; i ++) dis[i] = -1;
        queue<int>Q;
        dis[s] = 1; Q.push(s);
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = head[u]; ~i; i = edge[i].next){
                int v = edge[i].to;
```

```
52                  if(~dis[v] || edge[i].cap <= edge[i].flow) continue;
53                  dis[v] = dis[u] + 1; Q.push(v);
54              }
55          }
56          return ~dis[t];
57      }
58      int dfs(int u,int a){
59          if(u == t || !a) return a;
60          int flow = 0;
61          for(int& i = pre[u]; ~i; i = edge[i].next){
62              int v = edge[i].to;
63              if(dis[v] != dis[u] + 1) continue;
64              int f = dfs(v,min(edge[i].cap - edge[i].flow,a));
65              if(!f) continue;
66              edge[i].flow += f;
67              edge[i ^ 1].flow -= f;
68              flow += f;
69              a -= f;
70          }
71          return flow;
72      }
73      int maxflow(int s,int t){
74          int flow = 0;
75          while(BFS()){
76              for(int i = 0 ; i <= n ; i ++) pre[i] = head[i];
77              flow += dfs(s,INF);
78          }
79          return flow;
80      }
81      int maxflow(){return maxflow(s,t);}
82      void show(int t){
83          vis[t] = 1;
84          for(int i = head[t]; ~i ; i = edge[i].next){
85              int v = edge[i].to;
86              if(vis[v] || edge[i].cap <= edge[i].flow) continue;
87              show(v);
88          }
89      }
90  }g;
91  int main(){
92      cin >> N >> M; getchar();
93      int s = N + M + 1,t = N + M + 2;
94      g.init(N + M + 2,s,t);
95      int sum = 0;
```

```
96      for(int i = 1; i <= N ; i ++){
97          int x; Sca(x);
98          sum += x;
99          g.add(s,i,x);
100         while(1){
101             char c;
102             scanf("%d%c",&x,&c);
103             g.add(i,x + N,INF);
104             if(c == '\n' || c == '\r') break;
105         }
106     }
107     for(int i = 1; i <= M ; i ++){
108         int x; Sca(x);
109         g.add(i + N,t,x);
110     }
111     int p = g.maxflow();
112     g.show(s);                  //找一下那些点被选中了
113     for(int i = 1; i <= N ; i++){
114         if(vis[i]) printf("%d ",i);            //输出与S相连的正权点
115     }
116     puts("");
117     for(int i = 1 + N; i <= M + N ; i ++){      //输出与T相连的负权点
118         if(vis[i]) printf("%d ",i - N);
119     }
120     puts("");
121     Pri(sum - p);
122     return 0;
123 }
```

6.13 最小覆盖

6.13.1 DAG 最小点路径覆盖

</> 代码 6.20: /图论/最小点路径覆盖

```
1  /*
2      有向无环图的最小点路径覆盖
3      路径不相交，每个点属于一条路径（路径长度可为0），求最少路径
4      一开始最多有n条路径，对于每两个点的合并就会使路径减一
5      所以将点i拆为i和i + n两个点，i代表入点，i + n代表出点
6      S与入点连边，出点与T连边，容量均为1
7      对于每条边u,v,将u与v + N连边，容量均为1
8      S到T的最大流就是最多的可合并点，n - maxflow就是最小路径覆盖
9      如果要求方案的话，如果u - v + N这条边上有流量，代表这两个点被合并了
10 */
```

```cpp
//洛谷P2764 求N点M路的最小路径覆盖数量和方案
const int maxn = 410;
const int maxm = 20010;
int N,M,K;
struct Dinic{
    struct Edge{
        int to,next,cap,flow;
        Edge(){}
        Edge(int to,int next,int cap,int flow):to(to),next(next),cap(cap),flow(flow){}
    }edge[maxm * 2];
    int head[maxn],tot,n,s,t;
    int dis[maxn],pre[maxn];
    int nxt[maxn],vis[maxn];
    void init(int n,int s,int t){
        this->n = n; this->s = s; this->t = t;
        for(int i = 0 ; i <= n ; i ++) head[i] = -1;
        tot = 0;
    }
    void add(int u,int v,int w){
        edge[tot] = Edge(v,head[u],w,0);
        head[u] = tot++;
        edge[tot] = Edge(u,head[v],0,0);
        head[v] = tot++;
    }
    bool BFS(){
        for(int i = 0 ; i <= n ; i ++) dis[i] = -1;
        dis[s] = 0;
        queue<int>Q;
        Q.push(s);
        while(!Q.empty()){
            int u = Q.front(); Q.pop();
            for(int i = head[u]; ~i ; i = edge[i].next){
                int v = edge[i].to;
                if(~dis[v] || edge[i].cap <= edge[i].flow) continue;
                dis[v] = dis[u] + 1;
                Q.push(v);
            }
        }
        return ~dis[t];
    }
    int dfs(int u,int a){
        if(u == t || !a) return a;
        int flow = 0;
        for(int& i = pre[u]; ~i ; i = edge[i].next){
```

```
55              int v = edge[i].to;
56              if(dis[v] != dis[u] + 1) continue;
57              int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
58              if(!f) continue;
59              edge[i].flow += f;
60              edge[i ^ 1].flow -= f;
61              a -= f;
62              flow += f;
63          }
64          return flow;
65      }
66      int maxflow(int s,int t){
67          int flow = 0;
68          while(BFS()){
69              for(int i = 0 ; i <= n ; i ++) pre[i] = head[i];
70              flow += dfs(s,INF);
71          }
72          return flow;
73      }
74      int maxflow(){
75          return maxflow(s,t);
76      }
77      void show(){
78          for(int i = 1; i <= N ; i ++){
79              for(int j = head[i]; ~j; j = edge[j].next){
80                  int v = edge[j].to;
81                  if(!edge[j].flow) continue;
82                  if(1 + N <=v && v <= N + M){
83                      nxt[i] = v - N;
84                      vis[v - N] = 1;
85                      break;
86                  }
87              }
88          }
89          for(int i = 1; i <= N; i ++){
90              if(!vis[i]){
91                  int t = i;
92                  while(t){
93                      printf("%d ",t);
94                      t = nxt[t];
95                  }
96                  puts("");
97              }
98          }
```

```
    }
}g;
int main(){
    Sca2(N,M);
    int S = 2 * N + 1,T = 2 * N + 2;
    g.init(2 * N + 2,S,T);
    for(int i = 1; i <= N ; i ++){
        g.add(S,i,1);
        g.add(i + N,T,1);
    }
    for(int i = 1; i <= M ; i ++){
        int u,v; Sca2(u,v);
        g.add(u,v + N,1);
    }
    int ans = N - g.maxflow();
    g.show();
    Pri(ans);
    return 0;
}
```

6.14 费用流

6.14.1 SPFA 费用流

</> 代码 6.21: /图论/SPFA 费用流

```
//最小费用最大流
/*
    不断寻找费用最小的增广路，贪心的使得每一条找到的增广路都是当前残余网络下费用最小的路径
    最终寻找到流量最大前提下费用最小的路
*/
//模板 N点M边求S到T的最小费用最大流
const int maxn = 5010;
const int maxm = 50010;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Mcmf{
    struct Edge{
        int to,next,cap,flow,cost;
        Edge(){}
        Edge(int to,int next,int cap,int flow,int cost):to(to),next(next),cap(cap),flow(
    flow),cost(cost){}
    }edge[maxm * 2];
    int n,head[maxn],tot;
    int pre[maxn],dis[maxn],vis[maxn];
```

```
19      void init(int n){
20          this->n = n;
21          tot = 0;
22          for(int i = 0 ; i <= n ; i ++) head[i] = -1;
23      }
24      void add(int u,int v,int cap,int cost){
25          edge[tot] = Edge(v,head[u],cap,0,cost);
26          head[u] = tot++;
27          edge[tot] = Edge(u,head[v],0,0,-cost);
28          head[v] = tot++;
29      }
30      bool spfa(int s,int t){
31          for(int i = 0 ; i <= n ; i ++){
32              vis[i] = 0;
33              dis[i] = INF;
34              pre[i] = -1;
35          }
36          dis[s] = 0;
37          queue<int>Q; Q.push(s);
38          while(!Q.empty()){
39              int u = Q.front(); Q.pop();
40              vis[u] = 0;
41              for(int i = head[u]; ~i; i = edge[i].next){
42                  int v = edge[i].to;
43                  if(edge[i].cap - edge[i].flow > 0 && dis[v] > dis[u] + edge[i].cost){
44                      dis[v] = dis[u] + edge[i].cost;
45                      pre[v] = i;
46                      if(!vis[v]){
47                          vis[v] = 1;
48                          Q.push(v);
49                      }
50                  }
51              }
52          }
53          return ~pre[t];
54      }
55      int mcmf(int s,int t,int &cost){
56          int flow = 0;
57          cost = 0;
58          while(spfa(s,t)){
59              int Min = INF;
60              for(int i = pre[t]; ~i ; i = pre[edge[i ^ 1].to]){
61                  Min = min(Min,edge[i].cap - edge[i].flow);
62              }
```

```
63              flow += Min;
64              for(int i = pre[t]; ~i ; i = pre[edge[i ^ 1].to]){
65                  edge[i].flow += Min;
66                  edge[i ^ 1].flow -= Min;
67                  cost += Min * edge[i].cost;
68              }
69          }
70          return flow;
71      }
72  }g;
73  int main(){
74      int S,T;
75      scanf("%d%d%d%d",&N,&M,&S,&T);
76        g.init(N);
77      for(int i = 1; i <= M ; i ++){
78          int u,v,cap,cost;
79          scanf("%d%d%d%d",&u,&v,&cap,&cost);
80          g.add(u,v,cap,cost);
81      }
82      int maxflow,cost;
83      maxflow = g.mcmf(S,T,cost);
84      printf("%d %d",maxflow,cost);
85      return 0;
86  }
```

6.15 有上下界的网络流

6.15.1 无源汇上下界可行流

</> 代码 6.22: /图论/无源汇上下界可行流

```
1  //无源汇上下界可行流
2  /*
3  模型:一个网络,求出一个流,使得每条边的流量必须>=Li且<=Hi,每个点必须满足总流入量=总流出量(流量
       守恒)(这个流的特点是循环往复,无始无终).
4  求解方法: 1.以每条边的最低容量形成一个初始流 (不一定平衡)
5  2.求出初始流中每个点进入流量和输出流量的差
6  3.开始建图(附加流): 建立一个源点和汇点, 原图上所有边建立一条容量为Hi - Li的边, a[i]表示初始
       流中输入流量-输出流量,如果a[i] > 0,
7  则将i向汇点连容量为a[i]的边, 如果a[i] < 0则源点向i连容量为a[i]的边
8  4.可以证明所有a[i]之和相等, 因此在原图上跑最大流, 如果最大流与正数a[i]之和相等(即满流)则说明
       有解, 解就是
9  附加流上的流量加上初始流的流量
10  */
11 const int maxn = 210;
```

```
12 const int maxm = 11010;
13 const int INF = 0x3f3f3f3f;
14 const int mod = 1e9 + 7;
15 int N,M,K;
16 struct E{
17     int u,v,l,r;
18 }e[maxm];
19 int F[maxm];
20 int a[maxn];
21 struct dinic{
22     struct Edge{
23         int to,next,flow,cap,id;
24         Edge(){}
25         Edge(int to,int next,int flow, int cap,int id):
26             to(to),next(next),flow(flow),cap(cap),id(id){}
27     }edge[maxm * 2];
28     int head[maxn],tot;
29     int s,n,t,cur[maxn],dep[maxn];
30     void init(int n,int s,int t){
31         this->n = n; this->s = s; this->t = t;
32         for(int i = 0 ; i <= n ; i ++) head[i] = -1;
33         tot = 0;
34     }
35     void add(int u,int v,int w,int id){
36         edge[tot] = Edge(v,head[u],0,w,id);
37         head[u] = tot++;
38         edge[tot] = Edge(u,head[v],0,0,id);
39         head[v] = tot++;
40     }
41     inline bool bfs(){
42         for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
43         dep[s] = 1;
44         queue<int>Q; Q.push(s);
45         while(!Q.empty()){
46             int u = Q.front(); Q.pop();
47             for(int i = head[u]; ~i ; i = edge[i].next){
48                 int v = edge[i].to;
49                 if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
50                 dep[v] = dep[u] + 1;
51                 Q.push(v);
52             }
53         }
54         return ~dep[t];
55     }
```

```
56      inline int dfs(const int &u,int a){
57          if(u == t || !a) return a;
58          int flow = 0;
59          for(int &i = cur[u]; ~i ; i = edge[i].next){
60              int v = edge[i].to;
61              if(dep[v] != dep[u] + 1) continue;
62              int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
63              flow += f;
64              a -= f;
65              edge[i].flow += f;
66              edge[i ^ 1].flow -= f;
67          }
68          return flow;
69      }
70      int maxflow(){
71          return maxflow(s,t);
72      }
73      int maxflow(int s,int t){
74          int flow = 0;
75          while(bfs()){
76              for(int i = 0 ; i <= n ; i ++) cur[i] = head[i];
77              flow += dfs(s,INF);
78          }
79          return flow;
80      }
81      void solve(){
82          for(int i = 0 ; i <= n ; i ++){
83              for(int j = head[i]; ~j; j = edge[j].next){
84                  if(edge[j].flow > 0) F[edge[j].id] += edge[j].flow;
85              }
86          }
87      }
88  }g;
89  int main(){
90      Sca2(N,M);
91      int S = N + 1,T = N + 2;
92      g.init(N + 2,S,T);
93      for(int i = 1; i <= M ; i ++){
94          scanf("%d%d%d%d",&e[i].u,&e[i].v,&e[i].l,&e[i].r);
95          a[e[i].u] += e[i].l; a[e[i].v] -= e[i].l;    //a[i]表示初始流在这个点入流量和出流
        量的差
96          g.add(e[i].u,e[i].v,e[i].r - e[i].l,i);      //在附加流上建边
97      }
98      int sum = 0;
```

```
99      for(int i = 1; i <= N ; i ++){
100         if(a[i] > 0){
101             g.add(i,T,a[i],0);
102             sum += a[i];              //表示汇出的总流量
103         }
104         if(a[i] < 0) g.add(S,i,-a[i],0);
105     }
106     if(g.maxflow() != sum) puts("NO");   //不满流则无解
107     else{
108         g.solve();
109         puts("YES");
110         for(int i = 1; i <= M; i ++){
111             Pri(e[i].l + F[i]);
112         }
113     }
114     return 0;
115 }
```

6.15.2 有源汇上下界网络流

</> 代码 6.23: /图论/有源汇上下界网络流

```
1  /*
2  1.有源汇上下界可行流
3  首先有源汇与无源汇的区别在于：有源汇可行流中，源点和汇点不一定流量平衡，但是源点多余的流量=汇
       点的多余流量。
4  解决有源汇问题时我们通常将其转化为无源汇的问题，具体方法就是连一条(T,S,∞,0)的边。这样的话，
       汇点多余的输出就可以通过这条边进入源点。
5  然后我们建立超级源SS与超级汇TT，之后判断是否有可行流就用前面的方法
6  2.有源汇上下界最大流
7  因为S与T之间可能存在富余的边，导致没有得到最大流。
8  我们将超级源和超级汇以及与他们相连的边拆掉，然后在残余网络上在跑一次最大流。得到新的最大流与w
       的和就是答案。
9  3.有源汇上下界最小流
10 法1：与最大流类似。我们先得到了w之后我们尽量减少S和T之间的流量。于是我们拆了超级源和超级汇之
       后“反着”跑最大流，也就是拿T当源点，S当汇点跑最大流。可以这么理解：反边增加流量相当于正
       边减少流量。答案就是w-最大流。
11 法2：当然也可以不用改变图，直接反着跑最大流，答案就是INF-最大流。这里INF就是我们(T,S,∞,0)的
       上界。
12  */
13  //有源汇上下界最大流
14 const int maxn = 410;
15 const int maxm = 2e5 + 10;
16 const int INF = 0x3f3f3f3f;
17 int N,M,K,S,T;
```

```
18 int a[maxn],F[maxm];
19 struct E{
20     int u,v,l,r;
21 }e[maxm];
22 struct dinic{
23     struct Edge{
24         int to,next,flow,cap,id;
25         Edge(){}
26         Edge(int to,int next,int flow, int cap,int id):
27             to(to),next(next),flow(flow),cap(cap),id(id){}
28     }edge[maxm * 2];
29     int head[maxn],tot;
30     int s,n,t,cur[maxn],dep[maxn];
31     void init(int n){
32         this->n = n;
33         for(int i = 0 ; i <= n ; i ++) head[i] = -1;
34         tot = 0;
35     }
36     void add(int u,int v,int w,int id){
37         edge[tot] = Edge(v,head[u],0,w,id);
38         head[u] = tot++;
39         edge[tot] = Edge(u,head[v],0,0,id);
40         head[v] = tot++;
41     }
42     inline bool bfs(){
43         for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
44         dep[s] = 1;
45         queue<int>Q; Q.push(s);
46         while(!Q.empty()){
47             int u = Q.front(); Q.pop();
48             for(int i = head[u]; ~i ; i = edge[i].next){
49                 int v = edge[i].to;
50                 if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
51                 dep[v] = dep[u] + 1;
52                 Q.push(v);
53             }
54         }
55         return ~dep[t];
56     }
57     inline int dfs(const int &u,int a){
58         if(u == t || !a) return a;
59         int flow = 0;
60         for(int &i = cur[u]; ~i ; i = edge[i].next){
61             int v = edge[i].to;
```

```
62              if(dep[v] != dep[u] + 1) continue;
63              int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
64              flow += f;
65              a -= f;
66              edge[i].flow += f;
67              edge[i ^ 1].flow -= f;
68              if(!a) return flow;
69          }
70          return flow;
71      }
72      int maxflow(int s,int t){
73          this->s = s; this->t = t;
74          int flow = 0;
75          while(bfs()){
76              for(int i = 0 ; i <= n ; i ++) cur[i] = head[i];
77              flow += dfs(s,INF);
78          }
79          return flow;
80      }
81      void solve(){
82          for(int i = 0 ; i <= n ; i ++){
83              for(int j = head[i]; ~j; j = edge[j].next){
84                  if(edge[j].flow > 0) F[edge[j].id] += edge[j].flow;
85              }
86          }
87      }
88      void del(int S,int T,int SS,int TT){
89          for(int i = head[T]; ~i; i = edge[i].next){
90              if(edge[i].cap == INF){
91                  edge[i].cap = edge[i].flow = 0;
92                  edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
93              }
94          }
95          for(int i = head[SS]; ~i; i = edge[i].next){
96              edge[i].cap = edge[i].flow = 0;
97              edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
98          }
99          for(int i = head[TT]; ~i; i = edge[i].next){
100             edge[i].cap = edge[i].flow = 0;
101             edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
102         }
103     }
104 }g;
105 int main(){
```

```
106     scanf("%d%d%d%d",&N,&M,&S,&T);
107     int SS = N + 1,TT = N + 2;
108     g.init(N + 2);
109     for(int i = 1; i <= M ; i ++){
110         scanf("%d%d%d%d",&e[i].u,&e[i].v,&e[i].l,&e[i].r);
111         g.add(e[i].u,e[i].v,e[i].r - e[i].l,i);
112         a[e[i].u] += e[i].l; a[e[i].v] -= e[i].l;
113     }
114     g.add(T,S,INF,0);
115     int sum = 0;
116     for(int i = 1; i <= N; i ++){
117         if(a[i] > 0){
118             g.add(i,TT,a[i],M + 1);
119             sum += a[i];
120         }
121         else if(a[i] < 0) g.add(SS,i,-a[i],M + 1);
122     }
123     if(sum != g.maxflow(SS,TT)){
124         puts("please go home to sleep");
125         return 0;
126     }
127     //方法1.删除加入的边之后再残余网络上跑最大流
128     // g.solve();
129     // g.del(S,T,SS,TT);   //删除超级源点汇点以及加的T到S的边
130     // int ans = g.maxflow(S,T) + F[0];
131     //方法2.求解可行流并判断可行后，原封不动地进行一次最大流，这个最大流就是答案
132     int ans = g.maxflow(S,T);
133     Pri(ans);
134     return 0;
135 }
136 //有源汇上下界最小流
137 const int maxn = 50010;
138 const int maxm = 2e5 + 10;
139 const int INF = 0x3f3f3f3f;
140 const int mod = 1e9 + 7;
141 int N,M,K,S,T;
142 int a[maxn],F[maxm];
143 struct E{
144     int u,v,l,r;
145 }e[maxm];
146 struct dinic{
147     struct Edge{
148         int to,next,flow,cap,id;
149         Edge(){}
```

```
150         Edge(int to,int next,int flow, int cap,int id):
151             to(to),next(next),flow(flow),cap(cap),id(id){}
152     }edge[maxm * 2];
153     int head[maxn],tot;
154     int s,n,t,cur[maxn],dep[maxn];
155     void init(int n){
156         this->n = n;
157         for(int i = 0 ; i <= n ; i ++) head[i] = -1;
158         tot = 0;
159     }
160     void add(int u,int v,int w,int id){
161         edge[tot] = Edge(v,head[u],0,w,id);
162         head[u] = tot++;
163         edge[tot] = Edge(u,head[v],0,0,id);
164         head[v] = tot++;
165     }
166     inline bool bfs(){
167         for(int i = 0 ; i <= n ; i ++) dep[i] = -1;
168         dep[s] = 1;
169         queue<int>Q; Q.push(s);
170         while(!Q.empty()){
171             int u = Q.front(); Q.pop();
172             for(int i = head[u]; ~i ; i = edge[i].next){
173                 int v = edge[i].to;
174                 if(~dep[v] || edge[i].flow >= edge[i].cap) continue;
175                 dep[v] = dep[u] + 1;
176                 Q.push(v);
177             }
178         }
179         return ~dep[t];
180     }
181     inline int dfs(const int &u,int a){
182         if(u == t || !a) return a;
183         int flow = 0;
184         for(int &i = cur[u]; ~i ; i = edge[i].next){
185             int v = edge[i].to;
186             if(dep[v] != dep[u] + 1) continue;
187             int f = dfs(v,min(a,edge[i].cap - edge[i].flow));
188             flow += f;
189             a -= f;
190             edge[i].flow += f;
191             edge[i ^ 1].flow -= f;
192             if(!a) return flow;
193         }
```

```
194         return flow;
195     }
196     int maxflow(int s,int t){
197         this->s = s; this->t = t;
198         int flow = 0;
199         while(bfs()){
200             for(int i = 0 ; i <= n ; i ++) cur[i] = head[i];
201             flow += dfs(s,INF);
202         }
203         return flow;
204     }
205     void solve(){
206         for(int i = 0 ; i <= n ; i ++){
207             for(int j = head[i]; ~j; j = edge[j].next){
208                 if(edge[j].flow > 0) F[edge[j].id] += edge[j].flow;
209             }
210         }
211     }
212     void del(int S,int T,int SS,int TT){
213         for(int i = head[T]; ~i; i = edge[i].next){
214             if(edge[i].cap == INF){
215                 edge[i].cap = edge[i].flow = 0;
216                 edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
217             }
218         }
219         for(int i = head[SS]; ~i; i = edge[i].next){
220             edge[i].cap = edge[i].flow = 0;
221             edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
222         }
223         for(int i = head[TT]; ~i; i = edge[i].next){
224             edge[i].cap = edge[i].flow = 0;
225             edge[i ^ 1].cap = edge[i ^ 1].flow = 0;
226         }
227     }
228 }g;
229 int main(){
230     scanf("%d%d%d%d",&N,&M,&S,&T);
231     int SS = N + 1,TT = N + 2;
232     g.init(N + 2);
233     for(int i = 1; i <= M ; i ++){
234         e[i].u = read(),e[i].v = read();e[i].l = read();e[i].r = read();
235         g.add(e[i].u,e[i].v,e[i].r - e[i].l,i);
236         a[e[i].u] += e[i].l; a[e[i].v] -= e[i].l;
237     }
```

```
    g.add(T,S,INF,0);
    int sum = 0;
    for(int i = 1; i <= N; i ++){
        if(a[i] > 0){
            g.add(i,TT,a[i],M + 1);
            sum += a[i];
        }
        else if(a[i] < 0) g.add(SS,i,-a[i],M + 1);
    }
    if(sum != g.maxflow(SS,TT)){
        puts("please go home to sleep"); //不可行情况
        return 0;
    }
    g.solve();
    /*g.del(S,T,SS,TT);   //删除超级源点汇点以及加的T到S的边
    int ans = F[0] - g.maxflow(T,S);*/
    int ans = INF - g.maxflow(T,S);  //可将本行替换为上面的注释内容
    Pri(ans);
    return 0;
}
```

6.16 树的重心

</> 代码 6.24: /图论/树的重心

```
//树的重心：删除该点之后，使得形成的多棵树中节点数最大值最小。
//模板：求树的所有重心从小到达以此输出
const int maxn = 5e4 + 10;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int size[maxn],ans,weight[maxn];
void dfs(int t,int la){
```

```
    size[t] = 1;
    int heavy = 0;
    for(int i = head[t]; ~i; i = edge[i].next){
        int y = edge[i].to;
        if(y == la) continue;
        dfs(y,t);
        size[t] += size[y];
        heavy = max(heavy,size[y]);
    }
    heavy = max(heavy,N - size[t]);
    weight[t] = heavy;
    ans = min(ans,heavy);
}
int main(){
    Sca(N); init(); ans = INF;
    for(int i = 1; i <= N - 1; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    int root = 1;
    dfs(root,-1);
    for(int i = 1; i <= N ; i ++){
        if(ans == weight[i]){
            printf("%d ",i);
        }
    }
    return 0;
}
```

6.17 树的直径

6.17.1 双 dfs 法

</> 代码 6.25: /图论/树的直径双 dfs

```
//树的直径：树上最长长度的链被称作树的直径，求树直径长度的方法有双dfs法和树dp法
//一个有用的结论：从树上任意一个点出发到达的最远的点一定是这棵树的直径的一个端点（距离较长的那一个）。
//双dfs:任选一个点寻找出距离他最远的点a，然后以a为起点寻找出距离他最远的点b,ab距离为树的直径
const int maxn = 1e5 + 10;
int N,M,K;
struct Edge{
    int to,next,dis;
}edge[maxn * 2];
int head[maxn],tot;
```

```cpp
int dis[maxn];
void init(){
    for(int i = 0 ; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
void dfs(int t,int la){
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dis[v] = dis[t] + edge[i].dis;
        dfs(v,t);
    }
}
int main(){
    int T;
    scanf("%d",&T);
    int CASE = 1;
    while(T--){
        Sca(N); init();
        for(int i = 1; i <= N - 1; i ++){
            int u,v,w; Sca3(u,v,w);
            add(u,v,w); add(v,u,w);
        }
        int root = 1;
        dis[root] = 0; dfs(root,-1);
        for(int i = 1; i <= N ; i ++) if(dis[root] < dis[i]) root = i;
        dis[root] = 0; dfs(root,-1);
        for(int i = 1; i <= N ; i ++) if(dis[root] < dis[i]) root = i;
        printf("Case %d: ",CASE++);
        Pri(dis[root]);
    }
    return 0;
}
```

6.17.2 树形 dp 法

</> 代码 6.26: /图论/树的直径树 dp

```cpp
//树形dp法：记录当前节点最远端的节点和次远端的节点，两遍dfs更新即可
```

```cpp
//注：如果是图上直径，可采用两边BFS的方法
const int maxn = 1e5 + 10;
int N,M,K;
struct Edge{
    int to,next,dis;
}edge[maxn * 2];
int head[maxn],tot;
int dp[maxn],dp2[maxn];
void init(){
    for(int i = 0 ; i <= N ; i ++){
        head[i] = -1;
        dp[i] = dp2[i] = 0;
    }
    tot = 0;
}
void add(int u,int v,int w){
    edge[tot].to = v;
    edge[tot].next = head[u];
    edge[tot].dis = w;
    head[u] = tot++;
}
void change(int t,int w){
    if(dp[t] < w){
        dp2[t] = dp[t];
        dp[t] = w;
    }else if(dp2[t] < w){
        dp2[t] = w;
    }
}
void dfs(int t,int la){
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to,w = edge[i].dis;
        if(v == la) continue;
        dfs(v,t);
        change(t,dp[v] + w);
    }
}
void dfs2(int t,int la){
    for(int i = head[t];~i ; i = edge[i].next){
        int v = edge[i].to,w = edge[i].dis;
        if(v == la) continue;
        if(dp[t] == dp[v] + w) change(v,dp2[t] + w);
        else change(v,dp[t] + w);
        dfs2(v,t);
```

```
    }
}
int main(){
    int T;
    scanf("%d",&T);
    int CASE = 1;
    while(T--){
        Sca(N); init();
        for(int i = 1; i <= N - 1; i ++){
            int u,v,w; Sca3(u,v,w); u++; v++;
            add(u,v,w); add(v,u,w);
        }
        int root = 1;
        dfs(root,-1);
        dfs2(root,-1);
        int ans = 0;
        for(int i = 1; i <= N ; i++) ans = max(ans,dp[i] + dp2[i]);
        printf("Case %d: %d\n",CASE++,ans);
    }
    return 0;
}
```

6.18 树上 k 半径覆盖

</> 代码 6.27: /图论/树上 k 半径覆盖

```
/*树上k半径覆盖问题
一个点覆盖距离他最长为k的所有点，求一棵树上最少几个点可以全覆盖
贪心做法，从叶子结点向上遍历，每次遇到不得不放点的位置就在他往上k个位置放点。
*/
//例题 洛谷P2279 一个点覆盖最长距离为2的所有点，求做小覆盖点数
const int maxn =2010;
int N,M,K;
int fa[maxn],deep[maxn],a[maxn],o[maxn]; //a数组用来排序，o数组记录和最近的已经放的点还有多少距离
bool cmp(int a,int b){
    return deep[a] > deep[b];
}
int main(){
    Sca(N);  o[1] = INF;
    fa[1] = N + 1;
    fa[N + 1] = N + 2;
    o[N + 1] = o[N + 2] = INF;
    for(int i = 1; i <= N; i ++) a[i] = i;
```

```
    for(int i = 2; i <= N ; i ++){
        Sca(fa[i]);
        deep[i] = deep[fa[i]] + 1;
        o[i] = INF;
    }
    sort(a + 1,a + 1 + N,cmp);
    int ans = 0;
    for(int i = 1; i <= N ; i ++){
        int u = a[i];
        int v = fa[u],w = fa[fa[u]];
        o[u] = min(o[u],min(o[v] + 1,o[w] + 2));
        if(o[u] > 2){
            o[w] = 0;
            o[fa[w]] = min(o[fa[u]],1);
            o[fa[fa[w]]] = min(o[fa[fa[w]]],2);
            ans++;
        }
    }
    Pri(ans);
    return 0;
}
```

6.19 dfs 序

</> 代码 6.28: /图论/括号序列

```
//括号序列
/*首先dfs整棵树一遍，进入一个节点的时候加上一个左括号，然后是节点编号，
当这个节点的所有子树遍历完后再添上一个右括号，这就是括号序列
假如一棵树的括号序列是(1(2(3))(4(5)(6)(7(8))))
我们要求3到8的距离，截取两点间的括号序列为 3))(4(5)(6)(7(8
把编号和匹配的括号删掉 ))(((
剩下了5个左右括号，而这就是3到8的距离。这就是括号序列的性质。*/
/* 模板
操作1.将一个结点黑白变换，操作2.询问所有黑点之间的最长距离
解法：相当于求出括号序列，线段树维护两个黑点之间的最大距离sum*/
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
```

```
19      for(int i = 0 ; i <= N ; i ++) head[i] = -1;
20      tot = 0;
21  }
22  void add(int u,int v){
23      edge[tot].to = v;
24      edge[tot].next = head[u];
25      head[u] = tot++;
26  }
27  //括号序列
28  int id[maxn << 2],cnt,pos[maxn];
29  void dfs(int u,int la){
30      id[++cnt] = -1; //左括号
31      id[++cnt] = u; pos[u] = cnt;
32      for(int i = head[u]; ~i ; i = edge[i].next){
33          int v = edge[i].to;
34          if(v == la) continue;
35          dfs(v,u);
36      }
37      id[++cnt] = -2; // 右括号
38  }
39  //设左子树左右括号数量为l1,r1,右子树为l2,r2
40  //sum = l1 + abs(r1 - l2) + r2 = max(l1 + r1 + (r2 - l2),l1 - r1 + (r2 + l2))
41  //所以维护后缀的max(l + r),max(l - r),前缀的max(r - l),max(r + l)
42  struct Tree{
43      int l,r;
44      int a,b,sum;
45      int l1,l2; //max(l + r),max(r - l)
46      int r1,r2; //max(l - r),max(l + r)
47  }tree[maxn << 4];
48  //l1表示当前区间前缀的一段中左括号和右括号的和最大是多少，其他同理
49  bool col[maxn];
50  //因为是黑点的最远距离，除了黑点之外的坐标都不维护pre,erp,sum,直接当作-INF
51  void update(int t){
52      tree[t].a = tree[t].b = 0;
53      tree[t].sum = tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = -INF;
54      int v = tree[t].l;
55      if(id[v] == -1) tree[t].b = 1;
56      else if(id[v] == -2) tree[t].a = 1;
57      else if(!col[id[v]]){
58          tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = 0;
59      }
60  }
61  void Pushup(int t){
62      tree[t].b = tree[rc].b;
```

```
63      tree[t].a = tree[lc].a;
64      if(tree[lc].b > tree[rc].a) tree[t].b += tree[lc].b - tree[rc].a;
65      else tree[t].a += tree[rc].a - tree[lc].b;
66      tree[t].l1 = max(tree[lc].l1,max(tree[rc].l1 + tree[lc].a - tree[lc].b,tree[rc].l2 +
        tree[lc].a + tree[lc].b));
67      tree[t].l2 = max(tree[lc].l2,tree[rc].l2 + tree[lc].b - tree[lc].a);
68      tree[t].r1 = max(tree[rc].r1,tree[lc].r1 + tree[rc].a - tree[rc].b);
69      tree[t].r2 = max(tree[rc].r2,max(tree[lc].r2 + tree[rc].b - tree[rc].a,tree[lc].r1 +
        tree[rc].b + tree[rc].a));
70      tree[t].sum = max(tree[lc].r1 + tree[rc].l1,tree[lc].r2 + tree[rc].l2);
71      tree[t].sum = max(max(tree[lc].sum,tree[rc].sum),tree[t].sum);
72  }
73  void Build(int t,int l,int r){
74      tree[t].l = l; tree[t].r = r;
75      if(l == r){
76          update(t);
77          return;
78      }
79      int m = l + r >> 1;
80      Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
81      Pushup(t);
82  }
83  void update(int t,int p){
84      if(tree[t].l == tree[t].r){
85          update(t);
86          return;
87      }
88      int m = tree[t].l + tree[t].r >> 1;
89      if(p <= m) update(t << 1,p);
90      else update(t << 1 | 1,p);
91      Pushup(t);
92  }
93  int main(){
94      Sca(N); init();
95      for(int i = 1; i <= N - 1; i ++){
96          int u = read(),v = read();
97          add(u,v); add(v,u);
98      }
99      dfs(1,-1);
100     Build(1,1,cnt);
101     Sca(M); int num = N;
102     while(M--){
103         char op[3]; scanf("%s",op);
104         if(op[0] == 'G'){
```

```
105             if(num == 1) puts("0");
106             else if(!num) puts("-1");
107             else Pri(tree[1].sum);
108         }else{
109             int v = read();
110             if(col[v]) num--;
111             else num++;
112             col[v] ^= 1; update(1,pos[v]);
113         }
114     }
115     return 0;
116 }
```

6.20 括号序列

</> 代码 6.29: /图论/括号序列

```
1  //括号序列
2  /*首先dfs整棵树一遍，进入一个节点的时候加上一个左括号，然后是节点编号，
3  当这个节点的所有子树遍历完后再添上一个右括号，这就是括号序列
4  假如一棵树的括号序列是(1(2(3))(4(5)(6)(7(8))))
5  我们要求3到8的距离，截取两点间的括号序列为 3))(4(5)(6)(7(8
6  把编号和匹配的括号删掉 ))(((
7  剩下了5个左右括号，而这就是3到8的距离。这就是括号序列的性质。*/
8  /* 模板
9  操作1.将一个结点黑白变换，操作2.询问所有黑点之间的最长距离
10 解法：相当于求出括号序列，线段树维护两个黑点之间的最大距离sum*/
11 const int maxn = 1e5 + 10;
12 const int INF = 0x3f3f3f3f;
13 int N,M,K;
14 struct Edge{
15     int to,next;
16 }edge[maxn * 2];
17 int head[maxn],tot;
18 void init(){
19     for(int i = 0 ; i <= N ; i ++) head[i] = -1;
20     tot = 0;
21 }
22 void add(int u,int v){
23     edge[tot].to = v;
24     edge[tot].next = head[u];
25     head[u] = tot++;
26 }
27 //括号序列
```

```
int id[maxn << 2],cnt,pos[maxn];
void dfs(int u,int la){
    id[++cnt] = -1; //左括号
    id[++cnt] = u; pos[u] = cnt;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs(v,u);
    }
    id[++cnt] = -2; // 右括号
}
//设左子树左右括号数量为l1,r1,右子树为l2,r2
//sum = l1 + abs(r1 - l2) + r2 = max(l1 + r1 + (r2 - l2),l1 - r1 + (r2 + l2))
//所以维护后缀的max(l + r),max(l - r),前缀的max(r - l),max(r + l)
struct Tree{
    int l,r;
    int a,b,sum;
    int l1,l2; //max(l + r),max(r - l)
    int r1,r2; //max(l - r),max(l + r)
}tree[maxn << 4];
//l1表示当前区间前缀的一段中左括号和右括号的和最大是多少，其他同理
bool col[maxn];
//因为是黑点的最远距离，除了黑点之外的坐标都不维护pre,erp,sum,直接当作-INF
void update(int t){
    tree[t].a = tree[t].b = 0;
    tree[t].sum = tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = -INF;
    int v = tree[t].l;
    if(id[v] == -1) tree[t].b = 1;
    else if(id[v] == -2) tree[t].a = 1;
    else if(!col[id[v]]){
        tree[t].l1 = tree[t].l2 = tree[t].r1 = tree[t].r2 = 0;
    }
}
void Pushup(int t){
    tree[t].b = tree[rc].b;
    tree[t].a = tree[lc].a;
    if(tree[lc].b > tree[rc].a) tree[t].b += tree[lc].b - tree[rc].a;
    else tree[t].a += tree[rc].a - tree[lc].b;
    tree[t].l1 = max(tree[lc].l1,max(tree[rc].l1 + tree[lc].a - tree[lc].b,tree[rc].l2 +
     tree[lc].a + tree[lc].b));
    tree[t].l2 = max(tree[lc].l2,tree[rc].l2 + tree[lc].b - tree[lc].a);
    tree[t].r1 = max(tree[rc].r1,tree[lc].r1 + tree[rc].a - tree[rc].b);
    tree[t].r2 = max(tree[rc].r2,max(tree[lc].r2 + tree[rc].b - tree[rc].a,tree[lc].r1 +
     tree[rc].b + tree[rc].a));
```

```
70      tree[t].sum = max(tree[lc].r1 + tree[rc].l1,tree[lc].r2 + tree[rc].l2);
71      tree[t].sum = max(max(tree[lc].sum,tree[rc].sum),tree[t].sum);
72  }
73  void Build(int t,int l,int r){
74      tree[t].l = l; tree[t].r = r;
75      if(l == r){
76          update(t);
77          return;
78      }
79      int m = l + r >> 1;
80      Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
81      Pushup(t);
82  }
83  void update(int t,int p){
84      if(tree[t].l == tree[t].r){
85          update(t);
86          return;
87      }
88      int m = tree[t].l + tree[t].r >> 1;
89      if(p <= m) update(t << 1,p);
90      else update(t << 1 | 1,p);
91      Pushup(t);
92  }
93  int main(){
94      Sca(N); init();
95      for(int i = 1; i <= N - 1; i ++){
96          int u = read(),v = read();
97          add(u,v); add(v,u);
98      }
99      dfs(1,-1);
100     Build(1,1,cnt);
101     Sca(M); int num = N;
102     while(M--){
103         char op[3]; scanf("%s",op);
104         if(op[0] == 'G'){
105             if(num == 1) puts("0");
106             else if(!num) puts("-1");
107             else Pri(tree[1].sum);
108         }else{
109             int v = read();
110             if(col[v]) num--;
111             else num++;
112             col[v] ^= 1; update(1,pos[v]);
113         }
```

```
    }
    return 0;
}
```

6.21 LCA

</> 代码 6.30: /图论/LCA

```
//倍增lca
//注：如果动态建树的话，dfs函数非必要，建树的同时维护稀疏表即可
const int SP = 20; //(1 << SP)为这棵树最多的深度
int pa[maxn][SP],dep[maxn];
void init(){
    Mem(head, -1);
    tot = 0;
}
void dfs(int u, int fa){
    pa[u][0] = fa; dep[u] = dep[fa] + 1;
    for(int i = 1; i < SP; i ++) pa[u][i] = pa[pa[u][i - 1]][i - 1];
    for (int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if (v == fa) continue;
        dfs(v, u);
    }
}
int lca(int u, int v){
    if (dep[u] < dep[v]) swap(u, v);
    int t = dep[u] - dep[v];
    for(int i = 0 ; i < SP; i ++) if (t & (1 << i)) u = pa[u][i];
    for(int i = SP - 1; i >= 0 ; i --){
        int uu = pa[u][i], vv = pa[v][i];
        if (uu != vv){
            u = uu;
            v = vv;
        }
    }
    return u == v ? u : pa[u][0];
}
```

6.22 点分治

</> 代码 6.31: /图论/点分治

```
/*    树上点分治
```

```
    处理树链端点之间的关系，例如树上是否存在距离k的点对，或者树链点权相加小于等于K的个数之类
    核心思想是
    (1)找到树的重心
    (2)求这个重心的子树之间互相是否形成满足题意的条件（子树与子树之间，操作的是经过根节点的
    那条链）
    (3)遍历所有子树，将子树当成独立的树之后每棵树重复(1)
*/
//例题：洛谷P2634 寻找一棵树上有多少点之间的距离为3的的倍数（T），以及不是3的倍数(F) 求T / (
    F + T)的最简分数
int root,max_part,SUM;
int size[maxn];
void dfs_root(int u,int la){
    size[u] = 1;int heavy = 0;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la || vis[v]) continue;
        dfs_root(v,u);
        heavy = max(heavy,size[v]);
        size[u] += size[v];
    }
    if(max_part > max(heavy,SUM - heavy)){
        max_part = max(heavy,SUM - heavy);
        root = u;
    }
}
int judge[5],dis[5];
void dfs_dis(int t,int la,int d){
    dis[d]++;
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v] || v == la) continue;
        dfs_dis(v,t,(d + edge[i].dis) % 3);
    }
}
void work(int t){
    judge[0]++;
    for(int i = head[t]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        dfs_dis(v,t,edge[i].dis);
        for(int k = 0 ; k < 3; k ++){
            for(int j = 0 ; j < 3; j ++){
                if((k + j) % 3 == 0){
                    T += dis[k] * judge[j];
```

```
                }else{
                    F += dis[k] * judge[j];
                }
            }
        }
        for(int k = 0 ; k < 3; k ++){
            judge[k] += dis[k];
            dis[k] = 0;
        }
    }
    for(int i = 0 ; i < 3; i ++) judge[i] = 0;
}
void divide(int t){
    max_part = INF,root = t;
    dfs_root(t,-1);
    vis[root] = 1;
    work(root);
    for(int i = head[root]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(vis[v]) continue;
        SUM = size[v];
        divide(v);
    }
}
int gcd(int a,int b){
    return !b?a:gcd(b,a % b);
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N - 1; i ++){
        int u,v,w; Sca3(u,v,w);
        w %= 3;
        add(u,v,w); add(v,u,w);
    }
    SUM = N;divide(1);
    T <<= 1; F <<= 1;
    T += N;//cout << T << "  " << F + T << endl;
    LL GCD = gcd(T,T + F);
    printf("%lld/%lld",T / GCD,(T + F) / GCD);
    return 0;
}
```

6.23 动态点分治

</> 代码 6.32: /图论/动态点分治

```
1  //动态点分治(点分树)
2  /*点分树: 当我们可以形如点分治一样的统计答案, 即每次确定一个重心, 然后计算他们子树之间的贡献
       和得出答案的时候
3  我们可以将每个区域的重心作为其所有子树的重心的父亲, 构成一颗新的树, 显然这棵树的深度不会超过
       logn
4  每次对于单点 (边) 更新的时候, 只要对其所有的父亲更新, 就只需要更新log个点, 这样的数据结构就是
       点分树
5  */
6  //模板:BZOJ1095
7  /*
8  对于本题来说, 最终的答案是在每个点作为链上一个点的时候, 找每个点出发的最长链和次长链的和的最
       大值
9  所以用一个堆A维护每个点在点分树中子树下所有的点到这个点父亲的距离
10 再用一个堆B维护每个点所有儿子点的堆A的最大值, 即每条链的最长的长度
11 最后用一个堆C维护每个点的最长值 + 次长值的大小
12 tips:
13 1.树上两两之间的点的距离可以rmq + ST表预处理之后O(1)查询, 注意转化成序列不是dfs序, 具体看代
       码
14 2.做一个供删除的优先队列, 可以用两个优先队列A,B, 一个正常用, 删除操作就是把元素加入B, 当AB顶
       部相同的时候一起弹出
15 */
16 const int maxn = 2e5 + 10;
17 int N,M,K;
18 //带删除的优先队列
19 struct heap{
20     priority_queue<int>A,B;
21     void push(int x){A.push(x);}
22     void del(int x){B.push(x);}
23     void init(){
24         while(!B.empty() && A.top() == B.top()){
25             A.pop(); B.pop();
26         }
27     }
28     int top(){
29         init();
30         if(A.empty()) return 0;
31         return A.top();
32     }
33     int size(){
34         return A.size() - B.size();
35     }
36     int Maxdis(){
37         if(size() < 2) return 0;
```

```
38          int x = top(); A.pop();
39          int y = top(); A.push(x);
40          return x + y;
41      }
42  };
43  struct Edge{
44      int to,next;
45  }edge[maxn << 1];
46  int head[maxn],tot;
47  void init(){
48      for(int i = 0 ; i <= N ; i ++) head[i] = -1;
49      tot = 0;
50  }
51  void add(int u,int v){
52      edge[tot].to = v;
53      edge[tot].next = head[u];
54      head[u] = tot++;
55  }
56  //st表预处理树上距离
57  int dep[maxn];
58  int id[maxn],pos[maxn],cnt;
59  void dfsinit(int u,int la){
60      id[++cnt] = dep[u];
61      pos[u] = cnt;
62      for(int i = head[u]; ~i ; i = edge[i].next){
63          int v = edge[i].to;
64          if(v == la) continue;
65          dep[v] = dep[u] + 1;
66          dfsinit(v,u);
67          id[++cnt] = dep[u]; //注意这个点是u而不是v，这不是一个dfs序
68      }
69  }
70  const int SP = 20;
71  int MIN[maxn][SP],mm[maxn];
72  void initRMQ(int n,int b[]){
73      for(int i = 1; i <= n ; i ++){
74          for(int j = 0 ; j < SP; j ++){
75              MIN[i][j] = INF;
76          }
77      }
78      mm[0] = -1;
79      for(int i = 1; i <= n; i ++){
80          mm[i] = ((i & (i - 1)) == 0)?mm[i - 1] + 1:mm[i - 1];
81          MIN[i][0] = b[i];
```

```
    }
    for(int j = 1; j <= mm[n]; j ++){
        for(int i = 1; i + (1 << j) - 1 <= n; i ++){
            MIN[i][j] = min(MIN[i][j - 1],MIN[i + (1 << (j - 1))][j - 1]);
        }
    }
}
int rmq(int x,int y){
    if(x > y) swap(x,y);
    int k = mm[y - x + 1];
    return min(MIN[x][k],MIN[y - (1 << k) + 1][k]);
}
int getdis(int x,int y){ //查询x到y的树上距离
    return dep[x] + dep[y] - 2 * rmq(pos[x],pos[y]);
}
struct dtnode{
    int fa;
    heap Q,P;
    int Maxdis(){return Q.top();}
    int Maxline(){return P.Maxdis();}
}dt[maxn];
heap fans;
int root,max_part,SUM;
int size[maxn],vis[maxn];
int dis[maxn];
void dfs_root(int u,int la){
    size[u] = 1; int heavy = 0;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la || vis[v]) continue;
        dfs_root(v,u);
        heavy = max(heavy,size[v]);
        size[u] += size[v];
    }
    if(max_part > max(heavy,SUM - heavy)){
        max_part = max(heavy,SUM - heavy);
        root = u;
    }
}
void dfs(int u,int la){
    dis[u] = getdis(u,dt[root].fa);
    dt[root].Q.push(dis[u]);
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
```

```
126         if(v == la || vis[v]) continue;
127         dfs(v,u);
128     }
129 }
130 int divide(int t,int la){
131     max_part = INF;
132     root = t;
133     dfs_root(t,-1);
134     int now = root;
135     dt[root].fa = la; dt[root].P.push(0);
136     if(~la) dfs(root,-1);
137     vis[root] = 1;
138     for(int i = head[now]; ~i ; i = edge[i].next){
139         int v = edge[i].to;
140         if(vis[v]) continue;
141         SUM = size[v];
142         v = divide(v,now);
143         dt[now].P.push(dt[v].Maxdis());
144     }
145     if(dt[now].P.size() >= 2)fans.push(dt[now].Maxline());
146     return now;
147 }
148 int use[maxn];
149 int main(){
150     Sca(N); init();
151     for(int i = 1; i < N ; i ++){
152         int u = read(),v = read();
153         add(u,v); add(v,u);
154     }
155     dfsinit(1,-1); initRMQ(cnt,id);
156     SUM = N; divide(1,-1);
157     int num = N;
158     Sca(M);
159     while(M--){
160         char op[3]; scanf("%s",op);
161         if(op[0] == 'G'){
162             if(num == 1) puts("0");
163             else if(!num) puts("-1");
164             else{
165                 Pri(fans.top());
166             }
167         }else{
168             int v = read(); int tmp = v;
169             if(use[v]) num++;
```

```
            else num--;
            if(dt[v].P.size() >= 2) fans.del(dt[v].Maxline());
            if(use[v]) dt[v].P.push(0);
            else dt[v].P.del(0);
            if(dt[v].P.size() >= 2) fans.push(dt[v].Maxline());
            while(~dt[tmp].fa){
                int u = dt[tmp].fa;
                if(dt[u].P.size() >= 2) fans.del(dt[u].Maxline());
                if(dt[tmp].Q.size()) dt[u].P.del(dt[tmp].Maxdis());
                int d = getdis(v,u);
                if(use[v]) dt[tmp].Q.push(d);
                else dt[tmp].Q.del(d);
                if(dt[tmp].Q.size()) dt[u].P.push(dt[tmp].Maxdis());
                if(dt[u].P.size() >= 2) fans.push(dt[u].Maxline());
                tmp = u;
            }
            use[v] ^= 1;
        }
    }
    return 0;
}
```

6.24 树上差分

</> 代码 6.33: /图论/树上差分

```
/*    树上差分
    用来解决一系列将树链上的点权或者边权加上一个数，最后询问每个点的点权（边的边权）的操作
    如果是点权，对于u到v这条树链上的点权全部加上w，
    就进行val[u] += w; val[v] += w; val[lca(u,v)] -= w; val[fa[lca(u,v)]] -= w;
    最后求子树和的操作，每个数的子树和就是我们要的点权
    如果是边权，就将边都转为两端深度较深的那个点，u到v的边权加上w，就是
    val[u] += w; val[v] += w; val[lca(u,v)] -= 2 * w;
    最后同样求一个子树和的操作即可。
*/
//模板 点权的差分

int val[maxn];
void dfs2(int u,int la){
    for(int i = head[u]; ~i; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs2(v,u);
        val[u] += val[v];
```

```
    }
}
int main(){
    Sca(N); init();
    for(int i = 1; i <= N ; i ++) Sca(a[i]);
    for(int i = 1; i <= N - 1; i ++){
        int u,v; Sca2(u,v);
        add(u,v); add(v,u);
    }
    dfs(1,0);   //是lca里的dfs
    for(int i = 1; i < N ; i ++){
        int u = a[i],v = a[i + 1];
        val[u]++;val[v]++;
        int l = lca(u,v);
        val[l]--; val[fa[l][0]]--;
    }
    dfs2(1,0);
    for(int i = 1; i <= N ; i ++) Pri(val[i]);
    return 0;
}
```

6.25 树链剖分

</> 代码 6.34: /图论/树链剖分

```
//树链剖分，处理将整棵树化成一个序列方便区间操作，用线段树或者其他数据结构统一处理
//注意：如果树剖写T了，要关注序列有没有按照重链来排，有可能是重链那部分写错了
const int maxn = 3e4 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 1;i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
```

```
20
21 //求结点信息
22 int nw[maxn];
23 int dep[maxn],top[maxn],fa[maxn],key[maxn],pos[maxn],size[maxn],son[maxn];
24 int To_num[maxn];
25 void dfs1(int t,int la){
26     size[t] = 1; son[t] = t;
27     int heavy = 0;
28     for(int i = head[t]; ~i ; i = edge[i].next){
29         int v = edge[i].to;
30         if(v == la) continue;
31         dep[v] = dep[t] + 1;
32         fa[v] = t;
33         dfs1(v,t);
34         if(size[v] > heavy){
35             heavy = size[v];
36             son[t] = v;
37         }
38         size[t] += size[v];
39     }
40 }
41 int cnt;
42 void dfs2(int t,int la){
43     top[t] = la;
44     pos[t] = ++cnt;
45     To_num[cnt] = t;
46     nw[cnt] = key[t];
47     if(son[t] == t) return;
48     dfs2(son[t],la);
49     for(int i = head[t]; ~i ; i = edge[i].next){
50         int v = edge[i].to;
51         if((fa[t] == v) || (v == son[t])) continue;
52         dfs2(v,v);
53     }
54 }
55 //线段树
56 struct Tree{
57     int l,r;
58     int sum,MAX;
59 }tree[maxn << 2];
60 void Pushup(int t){
61     tree[t].sum = tree[t << 1].sum + tree[t << 1 | 1].sum;
62     tree[t].MAX = max(tree[t << 1].MAX,tree[t << 1 | 1].MAX);
63 }
```

```
64 void Build(int t,int l,int r){
65     tree[t].l = l; tree[t].r = r;
66     if(l == r){
67         tree[t].sum = tree[t].MAX = nw[l];
68         return;
69     }
70     int m = (l + r) >> 1;
71     Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
72     Pushup(t);
73 }
74 void update(int t,int p,int x){
75     if(tree[t].l == tree[t].r){
76         tree[t].MAX = tree[t].sum = x;
77         return;
78     }
79     int m = (tree[t].l + tree[t].r) >> 1;
80     if(p <= m) update(t << 1,p,x);
81     else update(t << 1 | 1,p,x);
82     Pushup(t);
83 }
84 int query(int t,int l,int r,int p){
85     if(l <= tree[t].l && tree[t].r <= r){
86         if(p) return tree[t].sum;
87         else return tree[t].MAX;
88     }
89     int m = (tree[t].l + tree[t].r ) >> 1;
90     if(r <= m) return query(t << 1,l,r,p);
91     else if(l > m) return query(t << 1 | 1,l,r,p);
92     else{
93         if(p) return query(t << 1,l,m,p) + query(t << 1 | 1,m + 1,r,p);
94         else return max(query(t << 1,l,m,p),query(t << 1 | 1,m + 1,r,p));
95     }
96 }
97
98 //树链剖分求链上信息
99 int query(int u,int v,int p){
100     int ans = 0;
101     if(!p) ans = -INF;
102     while(top[u] != top[v]){
103         if(dep[top[u]] < dep[top[v]]) swap(u,v);
104         if(p) ans += query(1,pos[top[u]],pos[u],1);
105         else ans = max(ans,query(1,pos[top[u]],pos[u],0));
106         u = fa[top[u]];
107     }
```

```
108     if(dep[u] > dep[v]) swap(u,v);
109     if(p) ans += query(1,pos[u],pos[v],p);
110     else ans = max(ans,query(1,pos[u],pos[v],p));
111     return ans;
112 }
113 int main(){
114     Sca(N); init();
115     for(int i = 1; i <= N - 1; i ++){
116         int u,v; Sca2(u,v);
117         add(u,v); add(v,u);
118     }
119     for(int i = 1; i <= N ; i ++) Sca(key[i]);
120     int root = 1;
121     dfs1(root,-1);
122     cnt = 0;
123     dfs2(root,root);
124     Build(1,1,N);
125     Sca(M);
126     while(M--){
127         char op[10]; int u,v;
128         scanf("%s%d%d",op,&u,&v);
129         if(op[0] == 'C'){
130             update(1,pos[u],v);     //把结点u权值改为t
131         }else if(op[1] == 'M'){
132             Pri(query(u,v,0));      //查询链上结点最大值
133         }else if(op[1] == 'S'){
134             Pri(query(u,v,1));       //查询树链上的权值和
135         }
136     }
137     return 0;
138 }
```

6.26 图论小贴士

!!!!!!!!! 1.SPFA 不得已不要用，虽然在有负权边（不只是负环）的时候无法使用 Dijkstra, 但是如果是一个 DAG 图，可以考虑拓扑排序求最短路
2. 如果一道长得很像最短路题，他的最优状态数组形如 dp[i][j] 表示 i 个点 j 个状态，可以考虑把他拆为 i * j 个点然后重新建图跑
3. 先看眼是有向图还是无向图，无向图数组开两倍。
4. 如果题目中没有声明无自环和重边，需要注意
5. 有些遍历的题要考虑环，否则可能死循环，可以使用缩点
6. 如果题目中边权小于等于零，要考虑负环、零环的情况

第七章　数据结构

7.1　链表

</> 代码 7.1:　/

```
1  //手写链表，可用于优化某些需要反复删除的数组。
2
3  int pre[maxn],nxt[maxn];
4  void del(int x){          //删除x结点
5      nxt[pre[x]] = nxt[x];
6      pre[nxt[x]] = pre[x];
7  }
8  void init(){             //初始化
9      nxt[N] = 0;
10     for(int i = 1; i <= N ; i ++){
11         nxt[i - 1] = i; pre[i] = i - 1;
12     }
13 }
```

7.2　并查集

7.2.1　可撤销并查集

</> 代码 7.2:　/

```
1  //可撤销并查集，可以将并查集的操作倒回到n步之前
2  //用一个栈存储进行的操作，倒回的时候依次退栈即可。
3  //不可以使用路径压缩，只能用按秩合并（size小的连到size大的上去）优化
4  int Stack[maxn],size[maxn];
5  int fa[maxn],top;
6  int now;
7  void init(){
8      for(int i = 1; i <= N ; i ++){
9          fa[i] = -1;size[i] = 0;
10     }
11     top = 0;
12 }
```

```
int find(int x){
    while(fa[x] != -1) x = fa[x];
    return x;
}
bool Union(int x,int y){
    x = find(x); y = find(y);
    if(x == y) return false;
    if(size[x] > size[y]) swap(x,y);
    Stack[top++] = x;
    fa[x] = y;
    size[y] += size[x] + 1;
    now--;
    return true;
}
void rewind(int t){
    while(top > t){
        int x = Stack[--top];
        size[fa[x]] -= size[x] + 1;
        fa[x] = -1;
        now++;
    }
}
```

7.2.2 可持久化并查集

</> 代码 7.3: /

```
/*用可持久化数组维护并查集的fa数组，模板为BZOJ3674
强制在线，每个值 ^ lastans
n个集合 m个操作
操作:
1 a b 合并a,b所在集合
2 k 回到第k次操作之后的状态(查询算作操作)
3 a b 询问a,b是否属于同一集合，是则输出1否则输出0
*/
const int maxn = 2e5 + 10;
int N,M,K;
int T[maxn],tot;
struct Tree{
    int lt,rt;
    int fa,size;
}tree[maxn * 52];
int fa[maxn],Size[maxn];
void newnode(int &t){
    t = ++tot;
```

```
19      tree[t].lt = tree[t].rt = tree[t].fa = 0;
20  }
21  void Build(int &t,int l,int r){
22      newnode(t);
23      if(l == r){
24          tree[t].fa = l;
25          tree[t].size = 0;
26          return;
27      }
28      int m = l + r >> 1;
29      Build(tree[t].lt,l,m); Build(tree[t].rt,m + 1,r);
30  }
31  int query(int t,int l,int r,int x){
32      if(l == r){
33          Size[x] = tree[t].size;
34          return tree[t].fa;
35      }
36      int m = l + r >> 1;
37      if(x <= m) return query(tree[t].lt,l,m,x);
38      else return query(tree[t].rt,m + 1,r,x);
39  }
40  void update(int &t,int pre,int l,int r,int x){
41      newnode(t);
42      tree[t] = tree[pre];
43      if(l == r){
44          tree[t].fa = fa[x];
45          tree[t].size = Size[x];
46          return;
47      }
48      int m = l + r >> 1;
49      if(x <= m) update(tree[t].lt,tree[pre].lt,l,m,x);
50      else update(tree[t].rt,tree[pre].rt,m + 1,r,x);
51  }
52  int find(int id,int x){
53      int f = query(T[id],1,N,x);
54      if(f == x) return x;
55      return find(id,f);
56  }
57  int main(){
58      //freopen("C.in","r",stdin);
59      Sca2(N,M);
60      Build(T[0],1,N);
61      int ans = 0;
62      for(int i = 1; i <= M; i ++){
```

```
63          int op = read();
64          if(op == 1){
65              int a = read() ^ ans,b = read() ^ ans;
66              a = find(i - 1,a); b = find(i - 1,b);
67              if(Size[a] > Size[b]) swap(a,b);
68              fa[a] = b;
69              update(T[i],T[i - 1],1,N,a);
70              if(Size[a] == Size[b]){
71                  Size[b]++; fa[b] = b;
72                  int x;
73                  update(x,T[i],1,N,b);
74                  T[i] = x;
75              }
76          }else if(op == 2){
77              int k = read() ^ ans;
78              T[i] = T[k];
79          }else if(op == 3){
80              T[i] = T[i - 1];
81              int a = read() ^ ans,b = read() ^ ans;
82              a = find(i,a); b = find(i,b);
83              ans = (a == b);
84              if(a == b) puts("1");
85              else puts("0");
86          }
87
88      }
89      return 0;
90  }
```

7.2.3 种类并查集

</> 代码 7.4: /

```
1  //带权并查集 （又称种类并查集）
2  //除了tree用来记录是否在同一个集合，利用group数组来记录当前节点和父亲结点的关系
3  int find(int x){
4      if (x == tree[x]) return x;
5      int fx = find(tree[x]);
6      group[x] = (group[tree[x]] + group[x] + 2) % 2; //孙子和爷爷的关系 = 孙子和父亲的关系
       + 父亲和爷爷的关系
7      tree[x] = fx;
8      return fx;
9  }
10 void Union(int a, int b){
11     int fa = find(a),fb = find(b);
```

```
12      if (fa != fb){
13          group[fa] = (-group[a] + 1 + group[b] + 2) % 2; // 父节点和 x 的关系，x和 y 的关系，y 和父节点的关系
14          tree[fa] = fb;
15      }
16  }
```

7.3 启发式合并

</> 代码 7.5: /

```
1   /* dsu on tree 启发式合并
2   nlogn时间处理一类静态无修的子树信息查询问题
3   通常暴力是每个结点下的子树dfs一遍，时间复杂度n²
4   像树剖一样处理出每个子树下的重子树，从上往下dfs的时候重子树的信息就可以选择保留
5   具体步骤:
6   1.递归所有u的轻点，目的在于计算出他们的答案，不保留贡献
7   2.递归u的重点，除了计算答案之外还保留贡献
8   3.再次递归u的轻点，目的在于计算贡献
9   */
10  //模板:每个节点有个颜色，一个子树中颜色节点最多的颜色占领整个子树，问所有子树被占领的颜色编号和（由于节点数有并列，所以是颜色编号和）
11  const int maxn = 1e5 + 10;
12  int N,M,K;
13  LL color[maxn];
14  struct Edge{
15      int to,next;
16  }edge[maxn * 2];
17  int head[maxn],tot;
18  void init(){
19      for(int i = 1; i <= N ; i ++) head[i] = -1;
20      tot = 0;
21  }
22  void add(int u,int v){
23      edge[tot].to = v;
24      edge[tot].next = head[u];
25      head[u] = tot++;
26  }
27  int cnt[maxn];
28  int fa[maxn],son[maxn],size[maxn];
29  void dfs1(int t,int la){
30      fa[t] = la; son[t] = 0;
31      size[t] = 1;
32      int heavy = 0;
```

```
33      for(int i = head[t]; ~i ; i = edge[i].next){
34          int v = edge[i].to;
35          if(v == la) continue;
36          dfs1(v,t);
37          if(size[v] > heavy){
38              son[t] = v;
39              heavy = size[v];
40          }
41          size[t] += size[v];
42      }
43  }
44  LL Max,ans[maxn],sum;
45  void dfs2(int t,int isson,int keep){
46      if(keep){
47          for(int i = head[t]; ~i ; i = edge[i].next){
48              int v = edge[i].to;
49              if(v == son[t] || v == fa[t]) continue;
50              dfs2(v,0,1);
51          }
52      }
53      if(son[t]) dfs2(son[t],1,keep);
54      for(int i = head[t]; ~i ; i = edge[i].next){
55          int v = edge[i].to;
56          if(v == son[t] || v == fa[t]) continue;
57          dfs2(v,0,0);
58      }
59      cnt[color[t]]++;
60      if(cnt[color[t]] > Max) sum = color[t],Max = cnt[color[t]];
61      else if(cnt[color[t]] == Max) sum += color[t];
62      if(keep) ans[t] = sum;
63      if(keep && !isson){
64           for(int i = 1; i <= N; i ++) cnt[i] = 0;
65           Max = sum = 0;
66      }
67  }
68  int main(){
69      Sca(N); init();
70      for(int i = 1; i <= N ; i ++) Scl(color[i]);
71      for(int i = 1; i < N ; i ++){
72          int u,v; Sca2(u,v);
73          add(u,v); add(v,u);
74      }
75      dfs1(1,0); dfs2(1,1,1);
76      for(int i = 1; i <= N ; i ++){
```

```
77          printf("%lld ",ans[i]);
78      }
79      return 0;
80  }
```

7.4 ST 表

</> 代码 7.6: /

```
1   /*  ST表
2       静态处理区间最大最小值
3       不可求和，不可动态
4   */
5   //模板：洛谷P2880 求区间最大值和最小值的差
6   const int maxn = 50010;
7   const int SP = 20;
8   int MAX[maxn][SP];
9   int MIN[maxn][SP];
10  int mm[maxn];
11  void initRMQ(int n,int b[]){          //预处理b数组，长度为n
12      for(int i = 1; i <= n ; i ++){
13          for(int j = 0; j < SP; j ++){
14              MIN[i][j] = INF;
15          }
16      }
17      mm[0] = -1;
18      for(int i = 1; i <= n ; i ++){
19          mm[i] = ((i & (i - 1)) == 0)?mm[i - 1] + 1:mm[i - 1];
20          MIN[i][0] = MAX[i][0] = b[i];
21      }
22      for(int j = 1; j <= mm[n]; j ++){
23          for(int i = 1; i + (1 << j) - 1 <= N ; i ++){
24              MAX[i][j] = max(MAX[i][j - 1],MAX[i + (1 << (j - 1))][j - 1]);
25              MIN[i][j] = min(MIN[i][j - 1],MIN[i + (1 << (j - 1))][j - 1]);
26          }
27      }
28  }
29  int rmq(int x,int y){
30      int k = mm[y - x + 1];
31      return max(MAX[x][k],MAX[y - (1 << k) + 1][k]) - min(MIN[x][k],MIN[y - (1 << k) +
        1][k]);
32  }
33  //倘若我们对模板稍加修改，便可以得到一个求最值下标的ST表
34  //模板：求区间最大值的下标
```

```
const int SP = 20;
int Max[maxn][SP];
int mm[maxn];
void initRMQ(int n,LL b[]){
    mm[0] = -1;
    for(int i = 1; i <= n; i ++){
        mm[i] = ((i & (i - 1)) == 0)?mm[i - 1] + 1:mm[i - 1];
        Max[i][0] = i;
    }
    for(int j = 1; j <= mm[n]; j ++){
        for(int i = 1; i + (1 << j) - 1 <= n ; i ++){
            int x = Max[i][j - 1],y = Max[i + (1 << (j - 1))][j - 1];
            Max[i][j] = b[x] > b[y]?x:y;
        }
    }
}
int rmq(int x,int y,LL b[]){
    int k = mm[y - x + 1];
    return b[Max[x][k]] > b[Max[y - (1 << k) + 1][k]]?Max[x][k]:Max[y - (1 << k) + 1][k
    ];
}
```

7.5 树状数组

</> 代码 7.7: /

```
//树状数组 注意0是无效下标
int tree[maxn];
void add(int x,int v){   //单点修改
    for(;x < N; x += x & -x) tree[x] += v;
}
LL sum(int x){           //查询前缀和
    LL ans = 0;
    for(;x > 0;x -= x & -x) ans += tree[x];
    return ans;
}
int kth(int k){           //查询从小到大数第K小,数组的下标表示数的大小, 第二维表示数的个数
    int ans = 0;
    LL cnt = 0;
    for(int i = SP - 1; i >= 0 ; i --){
        ans += (1 << i);
        if(ans >= N || cnt + tree[ans] >= k){
            ans -= 1 << i;
        }else cnt += tree[ans];
```

```
    }
    return ans + 1;
}
//模板:POJ2985 初始N个为1的集合，操作1.合并两个集合，操作2.查询第K大的集合的大小
const int maxn = 2e5 + 10;
const int SP = 20;
int N,M,K;
int fa[maxn],tree[maxn],sz[maxn];
void init(){
    for(int i = 0 ; i <= N ; i ++) fa[i] = i,sz[i] = 1;
}
int find(int x){
    if(fa[x] == x) return x;
    return fa[x] = find(fa[x]);
}
void add(int x,int v){
    for(;x < N; x += x & -x) tree[x] += v;
}
LL sum(int x){
    LL ans = 0;
    for(;x > 0;x -= x & -x) ans += tree[x];
    return ans;
}
int kth(int k){
    int ans = 0;
    LL cnt = 0;
    for(int i = SP - 1; i >= 0 ; i --){
        ans += (1 << i);
        if(ans >= N || cnt + tree[ans] >= k){
            ans -= 1 << i;
        }else cnt += tree[ans];
    }
    return ans + 1;
}
int main(){
    Sca2(N,M); init();
    add(1,N); int num = N;
    for(int i = 1; i <= M ; i ++){
        int op = read();
        if(op == 0){
            int u,v; Sca2(u,v);
            u = find(u),v = find(v);
            if(u == v) continue;
            add(sz[u],-1); add(sz[v],-1);
```

```
63              fa[u] = v;
64              sz[v] += sz[u];
65              add(sz[v],1); num--;
66          }else{
67              int k; Sca(k);
68              k = num - k + 1;
69              Pri(kth(k));
70          }
71      }
72      return 0;
73  }
74  //二维树状数组
75  //模板hdoj2642:单点修改,区间查询
76  const int maxn = 1010;
77  int N,M,K;
78  LL tree[maxn][maxn];
79  bool MAP[maxn][maxn];
80  void add(int x,int y,int v){
81      for(int tx = x;tx < maxn; tx += tx & -tx){
82          for(int ty = y;ty < maxn; ty += ty & -ty) tree[tx][ty] += v;
83      }
84  }
85  LL getsum(int x,int y){
86      LL ans = 0;
87      for(int tx = x;tx > 0; tx -= tx & -tx){
88          for(int ty = y; ty > 0; ty -= ty & -ty) ans += tree[tx][ty];
89      }
90      return ans;
91  }
92  LL getsum(int x1,int y1,int x2,int y2){
93      return getsum(x2,y2) + getsum(x1 - 1,y1 - 1) - getsum(x2,y1 - 1) - getsum(x1 - 1,y2)
        ;
94  }
95  int main(){
96      while(~Sca(M)){
97          Mem(tree,0);
98          while(M--){
99              char op[2]; scanf("%s",op);
100             if(op[0] == 'B'){
101                 int x = read(),y = read();
102                 x++;y++;
103                 if(!MAP[x][y]){
104                     MAP[x][y] = 1;
105                     add(x,y,1);
```

```
106                 }
107             }else if(op[0] == 'D'){
108                 int x = read(),y = read();
109                 x++;y++;
110                 if(MAP[x][y]){
111                     add(x,y,-1);
112                     MAP[x][y] = 0;
113                 }
114             }else{
115                 int x1 = read(),x2 = read(),y1 = read(),y2 = read();
116                 x1++;x2++;y1++;y2++;  //消除x = 0,y = 0的影响,0是无效点
117                 if(x1 > x2) swap(x1,x2);
118                 if(y1 > y2) swap(y1,y2);
119                 LL t = getsum(x1,y1,x2,y2);
120                 Prl(t);
121             }
122         }
123     }
124     return 0;
125 }
```

7.6 二维平面

7.6.1 二维前缀和

</> 代码 7.8: /数据结构/二维前缀和

```
1  //二维前缀和
2  //模板 hdu6541 给出1e6个全1矩阵，然后查询1e6个矩阵是否为全1矩阵
3  //做法：先用前缀和求出每个点是否是1，然后转化为01矩阵之后再求一个前缀和
4  const int maxn = 1e7 + 10;
5  int N,M,K;
6  LL MAP[maxn];
7  int id(int i,int j){
8      if(i <= 0 || j <= 0 || i > N || j > M) return 0;
9      return (i - 1) * M + j;
10 }
11 void add(int i,int j,int x){
12     int v = id(i,j);
13     if(!v) return;
14     MAP[v] += x;
15 }
16 int main(){
17     while(~Sca2(N,M)){
18         int Q = read();
```

```
19          for(int i = 0 ; i <= N * M; i ++) MAP[i] = 0;
20          while(Q--){
21              int x1 = read(),y1 = read(),x2 = read(),y2 = read();
22              if(x1 > x2) swap(x1,x2);
23              if(y1 > y2) swap(y1,y2);
24              add(x1,y1,1);
25              add(x1,y2 + 1,-1);
26              add(x2 + 1,y1,-1);
27              add(x2 + 1,y2 + 1,1);
28          }
29          for(int i = 1; i <= N; i ++){
30              for(int j = 1; j <= M ; j ++){
31                  MAP[id(i,j)] += MAP[id(i - 1,j)] + MAP[id(i,j - 1)] - MAP[id(i - 1,j -
    1)];
32              }
33          }
34          for(int i = 1; i <= N ; i ++){
35              for(int j = 1; j <= M ; j ++){
36                  MAP[id(i,j)] = (MAP[id(i,j)] > 0);
37              }
38          }
39          for(int i = 1; i <= N; i ++){
40              for(int j = 1; j <= M ; j ++){
41                  MAP[id(i,j)] += MAP[id(i - 1,j)] + MAP[id(i,j - 1)] - MAP[id(i - 1,j -
    1)];
42              }
43          }
44          Q = read();
45          while(Q--){
46              int x1 = read(),y1 = read(),x2 = read(),y2 = read();
47              if(x1 > x2) swap(x1,x2);
48              if(y1 > y2) swap(y1,y2);
49              if(1ll * (x2 - x1 + 1) * (y2 - y1 + 1) == MAP[id(x2,y2)] + MAP[id(x1 - 1,y1
    - 1)] - MAP[id(x2,y1 - 1)] - MAP[id(x1 - 1,y2)])
50                  puts("YES");
51              else
52                  puts("NO");
53          }
54      }
55      return 0;
56  }
```

7.6.2 二维 ST 表

</> 代码 7.9: /数据结构/二维 ST 表

```
//二维ST表 n²log(n)²预处理,O(1)查询
//注意空间复杂度不要MLE,SP和maxn能小尽量小
const int maxn = 255;
const int SP = 8;
int N,M,K;
int Max[maxn][maxn][SP][SP],Min[maxn][maxn][SP][SP];
int a[maxn][maxn],fac[20];
inline int max4(int a,int b,int c,int d){
    return max(max(a,b),max(c,d));
}
inline int min4(int a,int b,int c,int d){
    return min(min(a,b),min(c,d));
}
void init(){
    for(int i = 0 ; i < SP; i ++) fac[i] = (1 << i);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            Min[i][j][0][0] = a[i][j];
            Max[i][j][0][0] = a[i][j];
        }
    }
    int k = (int)(log((double)N) / log(2.0));
    for(int x = 0; x <= k; x++){
        for(int y = 0;y <= k; y++){
            if(x == 0 && y == 0) continue;
            for(int i = 1; i + fac[x] - 1 <= N; i ++){
                for(int j = 1; j + fac[y] - 1 <= M; j ++){
                    if(x == 0){
                        Min[i][j][x][y] = min(Min[i][j][x][y - 1],Min[i][j + (1 << (y -
    1))][x][y - 1]);
                        Max[i][j][x][y] = max(Max[i][j][x][y - 1],Max[i][j + (1 << (y -
    1))][x][y - 1]);
                    }else{
                        Min[i][j][x][y] = min(Min[i][j][x - 1][y],Min[i + (1 << (x - 1))
    ][j][x - 1][y]);
                        Max[i][j][x][y] = max(Max[i][j][x - 1][y],Max[i + (1 << (x - 1))
    ][j][x - 1][y]);
                    }
                }
            }
        }
    }
}
```

```
inline int getmax(int x1,int y1,int x2,int y2){
    int k1 = x2 - x1 + 1,k2 = y2 - y1 + 1;
    k1 = (int)(log((double)k1)/log(2.0));
    k2 = (int)(log((double)k2)/log(2.0));
    return max4(Max[x1][y1][k1][k2],Max[x1][y2 - fac[k2] + 1][k1][k2],Max[x2 - fac[k1] +
     1][y1][k1][k2],
    Max[x2 - fac[k1] + 1][y2 - fac[k2] + 1][k1][k2]);
}
inline int getmin(int x1,int y1,int x2,int y2){
    int k1 = x2 - x1 + 1,k2 = y2 - y1 + 1;
    k1 = (int)(log((double)k1)/log(2.0));
    k2 = (int)(log((double)k2)/log(2.0));
    return min4(Min[x1][y1][k1][k2],Min[x1][y2 - fac[k2] + 1][k1][k2],Min[x2 - fac[k1] +
     1][y1][k1][k2],
    Min[x2 - fac[k1] + 1][y2 - fac[k2] + 1][k1][k2]);
}
int ans[maxn][maxn];
int main(){
    int B;
    Sca3(N,B,K); M = N;
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= N ; j ++){
            Sca(a[i][j]);
        }
    }
    init();
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            int ii = i + B - 1,jj = j + B - 1;
            ii = min(ii,N); jj = min(jj,M);
            ans[i][j] = getmax(i,j,ii,jj) - getmin(i,j,ii,jj);
        }
    }
    while(K--){
        int x = read(),y = read();
        Pri(ans[x][y]);
    }
    return 0;
}
```

7.7 莫队算法

7.7.1 无修莫队

</> 代码 7.10: /数据结构/无修莫队

```
//无修莫队 时间复杂度n√n
//小z的袜子 查询区间内任选两个颜色相同的概率，用分数表示
const int maxn = 50010;
const int maxm = 50010;
int N,M,K,unit;
struct Query{
    int L,R,id;
    int l,r;
}node[maxm];
struct Ans{
    LL up,down;
    void reduce(){
        if(!up || !down){
            up = 0;
            down = 1;
            return;
        }
        LL g = __gcd(up,down);
        up /= g; down /= g;
    }
}ans[maxm];
int a[maxn];
bool cmp(Query a,Query b){
    if(a.l != b.l) return a.l < b.l;
    return a.r < b.r;
}
int num[maxn];
LL up,down,sum;
void add(int t){
    down += sum; sum++;
    up += num[t]; num[t]++;
}
void del(int t){
    sum--; down -= sum;
    num[t]--; up -= num[t];
}
void solve(){
    int L = 1,R = 0;
    sum = up = down = 0;
    for(int i = 1; i <= M ; i ++){
        while(R < node[i].R) add(a[++R]);
        while(R > node[i].R) del(a[R--]);
        while(L < node[i].L) del(a[L++]);
```

```
44          while(L > node[i].L) add(a[--L]);
45          ans[node[i].id].up = up;
46          ans[node[i].id].down = down;
47      }
48  }
49  int main(){
50      Sca2(N,M); unit = (int)sqrt(N);
51      for(int i = 1; i <= N ; i ++) Sca(a[i]);
52      for(int i = 1; i <= M ; i ++){
53          Sca2(node[i].L,node[i].R); node[i].id = i;
54          node[i].l = node[i].L / unit;
55          node[i].r = node[i].R / unit;
56      }
57      sort(node + 1,node + 1 + M,cmp);
58      solve();
59      for(int i = 1; i <= M ; i ++){
60          ans[i].reduce();
61          printf("%lld/%lld\n",ans[i].up,ans[i].down);
62      }
63      return 0;
64  }
```

7.7.2 回滚莫队

</> 代码 7.11: /数据结构/回滚莫队

```
1  /*例题: 无修改求区间L-R出现次数 * 权值最大的点
2  问题: 普通的莫队无法删除, 因为删除了最大值之后无法直接找到次大值
3  解决方法: 按照左端点所在的块排序, 相同的块按照右端点排序
4  对于左右端点相同块的询问直接暴力 √n
5  对于左端点处于同一块的询问一起处理, 因为右端点递增, 所以右端点只有删除惭怍
6  对于左端点, 每次将左端点初始化到所在块的最右端, 先向右扩充之后记录当前答案tmp, 然后向左扩充到
       当前询问的左端点, 找到当前询问的答案之后回溯到最右端, 答案也返回tmp
7  虽然依然有删除操作, 但是因为我们之前记录过tmp,避免了真正删除完寻找答案的过程
8  时间复杂度n√n
9  */
10 //BZOJ 4241
11 const int maxn = 1e5 + 10;
12 int N,M,K,Q,unit;
13 LL a[maxn],Hash[maxn];
14 int belong[maxn]; //每个点所在块的编号
15 struct Query{
16     int l,r,id;
17 }query[maxn];
18 bool cmp(Query a,Query b){
```

```
19      if(belong[a.l] == belong[b.l]) return a.r < b.r;
20      return belong[a.l] < belong[b.l];
21  }
22  LL vis[maxn],Max;
23  LL ans[maxn];
24  void add(int p){
25      vis[a[p]]++;
26      if(vis[a[p]] * Hash[a[p]]> Max){
27          Max = vis[a[p]] * Hash[a[p]];
28      }
29  }
30  void del(int p){
31      vis[a[p]]--;
32  }
33  LL use[maxn];
34  LL cul(int l,int r){
35      LL sum = 0;
36      for(int i = l; i <= r; i ++){
37          use[a[i]]++;
38          sum = max(sum,use[a[i]] * Hash[a[i]]);
39      }
40      for(int i = l; i <= r; i ++) use[a[i]]--;
41      return sum;
42  }
43  void solve(){
44      int up = N / unit;
45      int j = 1;
46      for(int i = 0; i <= up; i ++){
47          Max = 0;
48          LL L = (i + 1) * unit,R = L - 1,la,normal = L;
49          for(;j <= Q && belong[query[j].l] == i; j ++){  //对于所有左端点在一个块里的单独处
        理
50              if(belong[query[j].r] == i){  //如果处于同一个块 就暴力
51                  ans[query[j].id] = cul(query[j].l,query[j].r);
52                  continue;
53              }
54              while(R < query[j].r) add(++R);   //先扩充右边的
55              la = Max;                         //记录下当前答案
56              while(L > query[j].l) add(--L);  //再扩充左边的
57              ans[query[j].id] = Max;           //真正的答案
58              while(L < normal) del(L++);       //回溯，删除之前扩充的左边的
59              Max = la;                         //回溯答案
60          }
61          while(L <= R) del(L++);
```

```
    }
}
int main(){
    Sca2(N,Q); unit = (int)sqrt(N);
    for(int i = 1; i <= N ; i ++) Hash[i] = a[i] = read();
    sort(Hash + 1,Hash + 1 + N);
    int cnt = unique(Hash + 1,Hash + 1 + N) - Hash - 1;
    for(int i = 1; i <= N ; i ++) a[i] = lower_bound(Hash + 1,Hash + 1 + cnt,a[i]) -
    Hash;
    for(int i = 1; i <= N ; i ++) belong[i] = i / unit;
    for(int i = 1; i <= Q; i ++){
        query[i].l = read(); query[i].r = read();
        query[i].id = i;
    }
    sort(query + 1,query + 1 + Q,cmp);
    solve();
    for(int i = 1; i <= Q; i ++) Prl(ans[i]);
    return 0;
}
```

7.7.3 带修莫队

</> 代码 7.12: /数据结构/带修莫队

```
//带修莫队 时间复杂度三次根号内(n ^ 4 * t)
//模板: 带修询问区间颜色种类数
const int maxn = 150010;
int N,M,K,cnt,t,unit;
int a[maxn];
struct Update{
    int pre,aft;  //修改前和修改后
    int pos;    //修改的位置
    Update(int pos = 0,int pre = 0,int aft = 0):pos(pos),pre(pre),aft(aft){}
}update[maxn];
struct Query{
    int l,r,t;
    int L,R;
    int upnum; //修改过的次数
    Query(){}
    Query(int t,int l,int r,int upnum):t(t),l(l),r(r),upnum(upnum){}
}query[maxn];
bool cmp(Query a,Query b){
    if(a.L != b.L) return a.L < b.L;
    if(a.R != b.R) return a.R < b.R;
    return a.t < b.t;
```

```
}
int num[1000010],sum,ans[maxn];
void add(int t){
    num[t]++;
    sum += (num[t] == 1);
}
void del(int t){
    num[t]--;
    sum -= (num[t] == 0);
}
void solve(){
    int l = 1, r = 0,t = 0;
    sum = 0;
    for(int i = 1; i <= M ; i ++){
        while(query[i].l < l) add(a[--l]);
        while(query[i].l > l) del(a[l++]);
        while(query[i].r > r) add(a[++r]);
        while(query[i].r < r) del(a[r--]);
        while(query[i].upnum < t){
            int p = update[t].pos;
            if(l <= p && p <= r) del(a[p]);
            a[p] = update[t--].pre;
            if(l <= p && p <= r) add(a[p]);
        }
        while(query[i].upnum > t){
            int p = update[++t].pos;
            if(l <= p && p <= r) del(a[p]);
            a[p] = update[t].aft;
            if(l <= p && p <= r) add(a[p]);
        }
        ans[query[i].t] = sum;
    }
}
int main(){
    Sca2(N,M); cnt = t = 0;
    for(int i = 1; i <= N ; i ++) Sca(a[i]);
    for(int i = 1; i <= M ; i ++){
        char op[2]; scanf("%s",op);
        if(op[0] == 'R'){ //修改
            int p = read(),c = read();//p点改为c
            update[++t] = Update(p,a[p],c); a[p] = c;
        }else{
            cnt++; int l = read(),r = read();
            query[cnt] = Query(cnt,l,r,t);
```

```
        }
    }
    unit = ceil(exp((log(N) + log(t))/3));
    for(int i = t; i >= 1; i --) a[update[i].pos] = update[i].pre;  //初始化a数组
    for(int i = 1; i <= cnt ; i ++){
        query[i].L = query[i].l/unit;
        query[i].R = query[i].R/unit;
    }
    sort(query + 1,query + 1 + cnt,cmp);
    solve();
    for(int i = 1; i <= cnt ; i ++) Pri(ans[i]);
    return 0;
}
```

7.7.4 树上莫队

</> 代码 7.13: /数据结构/树上莫队

```
//树上莫队
/* 查询树链的关系
求出树的欧拉序,将树链关系转化为序列关系
对于不同的子树上的两点而言，序列上前一个的ed到后一个的st包含的所有只出现一次的结点就是树链上的结点。
但是lca不包含在内，因此要特判lca
*/
//查询树链上不同颜色的种数
const int maxn = 4e5 + 10;
const int maxm = 4e5 + 10;
const int SP = 20;
int fa[maxn][SP],dep[maxn];
int N,M,K,unit;
struct Edge{
    int to,next;
}edge[maxn * 2];
int head[maxn],tot;
void init(){
    for(int i = 0; i <= N ; i ++) head[i] = -1;
    tot = 0;
}
void add(int u,int v){
    edge[tot].to = v;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int idx[maxn],cnt;
```

```
PII pos[maxn];
int ans[maxn],b[maxn],a[maxn];
void dfs(int u,int la){
    fa[u][0] = la;
    dep[u] = dep[la] + 1;
    for(int i = 1; i < SP; i ++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    idx[++cnt] = u; a[cnt] = b[u];
    pos[u].fi = cnt;
    for(int i = head[u]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if(v == la) continue;
        dfs(v,u);
    }
    idx[++cnt] = u; a[cnt] = b[u];
    pos[u].se = cnt;
}
int lca(int u,int v){
    if(dep[u] < dep[v]) swap(u,v);
    int t = dep[u] - dep[v];
    for(int i = 0 ; i < SP; i ++) if(t & (1 << i)) u = fa[u][i];
    for(int i = SP - 1; i >= 0 ; i --){
        int uu = fa[u][i],vv = fa[v][i];
        if(uu != vv){
            u = uu;
            v = vv;
        }
    }
    return u == v? u : fa[u][0];
}
struct Query{
    int l,r,t;
    int L,R;
    int lc;  //特判lca
}node[maxm];
bool cmp(Query a,Query b){
    if(a.l != b.l) return a.l < b.l;
    return a.r < b.r;
}
int num[maxm],sum,vis[maxn];
void add(int p){
    num[p]++;
    sum += (num[p] == 1);
}
void del(int p){
```

```
71      num[p]--;
72      sum -= (num[p] == 0);
73  }
74  void work(int t){
75      if(vis[t]) del(b[t]);
76      else add(b[t]);
77      vis[t] ^= 1;
78  }
79  void solve(){
80      int L = 1,R = 0;
81      sum = 0;
82      for(int i = 1; i <= M ; i ++){
83          while(R < node[i].R) work(idx[++R]);
84          while(R > node[i].R) work(idx[R--]);
85          while(L < node[i].L) work(idx[L++]);
86          while(L > node[i].L) work(idx[--L]);
87          if(node[i].lc) work(node[i].lc);
88          ans[node[i].t] = sum ;
89          if(node[i].lc) work(node[i].lc);
90      }
91  }
92  int Hash[maxn];
93  int main(){
94      Sca2(N,M); init(); cnt = 0;
95      for(int i = 1; i <= N ; i ++) Hash[i] = b[i] = read();
96      sort(Hash + 1,Hash + 1 + N);
97      int c = unique(Hash + 1,Hash + 1 + N) - Hash;
98      for(int i = 1; i <= N ; i ++) b[i] = lower_bound(Hash + 1,Hash + 1 + c,b[i]) - Hash;
99      for(int i = 1; i <= N - 1 ; i ++){
100         int u,v; Sca2(u,v);
101         add(u,v); add(v,u);
102     }
103     dfs(1,-1); unit = (int)sqrt(cnt);
104     for(int i = 1; i <= M ; i ++){
105         int u = read(),v = read();
106         if(pos[u].fi > pos[v].fi) swap(u,v);
107         node[i].lc = lca(u,v);
108         node[i].L = pos[u].se; node[i].R = pos[v].fi;
109         if(node[i].lc == u){    //如果u是v的祖先,就不用特判lca
110             node[i].L = pos[u].fi;
111             node[i].lc = 0;
112         }
113         node[i].l = node[i].L / unit; node[i].r = node[i].R / unit;
114         node[i].t = i;
```

```
    }
    sort(node + 1,node + 1 + M,cmp);
    solve();
    for(int i = 1; i <= M ; i ++) Pri(ans[i]);
    return 0;
}
```

7.8 珂朵莉树

</> 代码 7.14: /数据结构/珂朵莉树

```

/* 珂朵莉树
1.名称来源：从CF896C发明的算法，题干主角是动漫人物珂朵莉
2.解决一类问题：(1)含有区间赋值操作 (2)数据随机 (重要)
3.具体思路：
(1)用set存储每个区间的(l,r)端点和信息
(2)split:当出现新的在l,r之间的端点pos的时候将区间分为(l,pos- 1)(pos,r)
(3)assign:为了降低节点数量，区间赋值的时候把整个区间的结点都去掉，换一个Node(l,r,v)进去
 */
 //模板CF896C,区间加,区间赋值,区间K小,区间每个数K次方和
const int maxn = 1e5 + 10;
const int mod = 1e9 + 7;
int N,M,K;
LL n,m,seed,vmax,ret;
LL rnd(){
    ret = seed;
    seed = (seed * 7 + 13) % mod;
    return ret;
}
LL quick_power(LL a,LL b,LL Mod){
    LL ans = 1;
    a %= Mod;
    while(b){
        if(b & 1) ans = (ans * a) % Mod;
        b >>= 1;
        a = a * a % Mod;
    }
    return ans;
}
#define IT set<node>::iterator
struct node{
    int l,r;
    mutable LL v; //表示可变的变量，突破const函数的限制
```

```
34      node(int l,int r = -1,LL v = 0):l(l),r(r),v(v) {}
35      friend bool operator < (node a,node b){
36          return a.l < b.l;
37      }
38  };
39  set<node>s;
40  //分割出左端点包含l的区间
41  IT split(int pos){
42      IT it = s.lower_bound(node(pos));
43      if(it != s.end() && it->l == pos) return it;
44      --it;
45      int L = it->l,R = it->r;
46      LL V = it->v;
47      s.erase(it);
48      s.insert(node(L,pos - 1,V));
49      return s.insert(node(pos,R,V)).first;
50  }
51  //推平l-r区间
52  void assign(int l,int r,LL val){
53      IT itr = split(r + 1),itl = split(l); //一定要先split(r + 1)
54      s.erase(itl,itr);
55      s.insert(node(l,r,val));
56  }
57  void add(int l,int r,LL val){
58      IT itr = split(r + 1),itl = split(l);
59      for(;itl != itr; ++itl) itl->v += val;
60  }
61  LL ranks(int l,int r,int k){ //第k小
62      vector<PLI> vp;
63      IT itr = split(r + 1),itl = split(l);
64      vp.clear();
65      for(;itl != itr; ++itl) vp.pb(mp(itl->v,itl->r - itl->l + 1));
66      sort(vp.begin(),vp.end());
67      for(int i = 0 ; i < vp.size(); i ++){
68          k -= vp[i].se;
69          if(k <= 0) return vp[i].fi;
70      }
71  }
72
73  LL sum(int l,int r,LL k,LL Mod){
74      IT itr = split(r + 1),itl = split(l);
75      LL ans = 0;
76      for(;itl != itr; itl++) ans = (ans + (LL)(itl->r - itl->l + 1) * quick_power(itl->v,
        k,Mod)) % Mod;
```

```
77      return ans;
78  }
79  LL a[maxn];
80  int main(){
81      scanf("%lld%lld%lld%lld",&n,&m,&seed,&vmax);
82      for(int i = 1; i <= n ; i ++){
83          a[i] = (rnd() % vmax) + 1;
84          s.insert(node(i,i,a[i]));
85      }
86      s.insert(node(n + 1,n + 1,0)); //防止查询到空
87      for(int i = 1; i <= m ; i ++){
88          int op = (rnd() % 4) + 1;
89          int l = (rnd() % n) + 1,r = (rnd() % n) + 1;
90          if(l > r) swap(l,r);
91          int x,y;
92          if(op == 3) x = (rnd() % (r - l + 1)) + 1;
93          else x = (rnd() % vmax) + 1;
94          if(op == 4) y = (rnd() % vmax) + 1;
95          //以上随机 op,x,y,l,r
96          if(op == 1) add(l,r,x); //区间加
97          else if(op == 2) assign(l,r,x); //推平
98          else if(op == 3) Prl(ranks(l,r,x)); //k小
99          else if(op == 4) Prl(sum(l,r,x,y)); //区间x的次方和模y
100     }
101     return 0;
102 }
```

7.9 动态开点线段树

</> 代码 7.15: /数据结构/动态开点线段树

```
1  /* 动态线段树
2  如果线段树要操作的范围在1 - INF (一个空间存不下的大小)
3  如果离线可以考虑离散化, 如果强制在线就要考虑一下动态线段树了
4  动态线段树不是一口气把全部结点都开完, 而是动态的只开要开的结点
5  */
6  //模板, hdu1199 n个操作为把a - b范围涂成黑色或者白色, 查询为最长的连续白色区间, 同样长输出最
       左边
7  const int maxn = 2010;
8  const int INF = 0x3f3f3f3f;
9  const int mod = 1e9 + 7;
10
11 int N,M;
12 struct Tree{
```

```
13      LL sum;    //最大连续
14      LL lsum;   //左连续
15      LL rsum;   //右连续
16      int lt,rt,lazy;
17      void init(){
18          lsum = rsum = sum = lt = rt = 0;
19          lazy = -1;
20      }
21  }tree[maxn * 60];
22  int tot;
23  void check(int &t){
24      if(t) return;
25      t = ++tot;
26      tree[t].init();
27  }
28  void add(int &t,LL L,LL R,int v){
29      if(v){
30          tree[t].sum = tree[t].lsum = tree[t].rsum = R - L + 1;
31      }else{
32          tree[t].sum = tree[t].lsum = tree[t].rsum = 0;
33      }
34      tree[t].lazy = v;
35  }
36  void Pushdown(int& t,LL L,LL R){
37      if(tree[t].lazy == -1) return;
38      int &lt = tree[t].lt; int &rt = tree[t].rt;
39      LL M = (L + R) >> 1;
40      check(lt); check(rt);
41      add(lt,L,M,tree[t].lazy);
42      add(rt,M + 1,R,tree[t].lazy);
43      tree[t].lazy = -1;
44  }
45  void Pushup(int t,LL L,LL R){
46      int &ls = tree[t].lt; int &rs = tree[t].rt;
47      LL M = (L + R) >> 1;
48      check(ls); check(rs);
49      tree[t].sum = max(tree[ls].sum,tree[rs].sum);
50      tree[t].sum = max(tree[t].sum,tree[ls].rsum + tree[rs].lsum);
51      tree[t].lsum = tree[ls].lsum;
52      if(tree[ls].lsum == M - L + 1){
53          tree[t].lsum = tree[ls].lsum + tree[rs].lsum;
54      }
55      tree[t].rsum = tree[rs].rsum;
56      if(tree[rs].rsum == R - M){
```

```
57          tree[t].rsum = tree[rs].rsum + tree[ls].rsum;
58      }
59  }
60  //如果是单点修改的话，一个小优化是一路直接修改下去而不是从底部往上Pushup
61  void update(int &t,int q1,int q2,LL L,LL R,int v){
62      check(t);
63      if(q1 <= L && R <= q2){
64          add(t,L,R,v);
65          return;
66      }
67      Pushdown(t,L,R);
68      LL m = (L + R) >> 1;
69      if(q1 <= m) update(tree[t].lt,q1,q2,L,m,v);
70      if(q2 > m) update(tree[t].rt,q1,q2,m + 1,R,v);
71      Pushup(t,L,R);
72  }
73  LL Left,Right;
74  //查询有时可用的一个优化是如果这个点不存在，不需要开辟空间继续查询，之间return 0即可
75  void query(int t,LL L,LL R){
76      if(L == R){
77          Left = L;
78          Right = R;
79          return;
80      }
81      check(tree[t].lt); check(tree[t].rt);
82      int ls = tree[t].lt; int rs = tree[t].rt;
83      LL M = (L + R) >> 1;
84      if(tree[ls].sum == tree[t].sum) query(ls,L,M);
85      else if(tree[rs].sum == tree[t].sum) query(rs,M + 1,R);
86      else{
87          Left = M - tree[ls].rsum + 1;
88          Right = M + tree[rs].lsum;
89          return;
90      }
91
92  }
93  int main()
94  {
95      while(~Sca(N)){
96          LL L = 1; LL R = 2147483647;
97          tot = 0;
98          int root = 0;
99          update(root,L,R,L,R,0);
100         For(i,1,N){
```

```
            LL l,r;
            char op[3];
            scanf("%lld%lld%s",&l,&r,&op);
            if(op[0] == 'w'){
                update(root,l,r,L,R,1);
            }else{
                update(root,l,r,L,R,0);
            }
        }
        if(!tree[root].sum){
            puts("Oh, my god");
            continue;
        }
        query(root,L,R);
        printf("%lld %lld\n",Left,Right);
    }
    return 0;
}
```

7.10 李超树

</> 代码 7.16: /数据结构/李超树

```
/* 李超线段树
用来维护二维平面上很多条线段(直线)在x = x0上的最值问题
定义:
1.永久化标记: 即线段树标记不删除, 每个结点维护的也不一定是最优的信息, 需要查询的时候一路统计标记达到最优
2.优势线段: 李超树每个节点含有一个优势线段, 意为完全覆盖当前区间且在当前区间mid处相比于其他该位置线段最大(小)的线段, 李超树的id记录的即为当前的优势线段
3.核心思想: 假设维护最大值, 每次插入线段时, 如果斜率较大的线段为优势线段, 则斜率较小的线段只有在左子树才有机会比优势线段大
如果斜率较小的线段为优势线段, 则较大的线段只有在右子树才能翻盘
修改(log(n))²(但是常数不大),查询log(n),
*/
/* 模板
操作1.增加一条斜率为P,起始点为S的直线
操作2.询问k点上所有直线的最大值
*/
const int maxn = 5e4 + 10;
int N,M,K;
struct Tree{
    int l,r;
    double S,P;  //P为斜率,S为初始值
```

```
}tree[maxn << 2];
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    tree[t].S = tree[t].P = 0;
    if(l == r) return;
    int m = l + r >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
}
void update(int t,double S,double P){
    int mid = tree[t].l + tree[t].r >> 1;
    if(P > tree[t].P) swap(tree[t].P,P), swap(tree[t].S,S);
    if(S + P * mid < tree[t].S + tree[t].P * mid){ //斜率大的更大就用小的更新左边
        if(tree[t].l == tree[t].r) return;
        update(t << 1,S,P);
    }else{                                          //否则用大的更新右边
        swap(tree[t].P,P); swap(tree[t].S,S);
        if(tree[t].l == tree[t].r) return;
        update(t << 1 | 1,S,P);
    }
}
double ans;
void query(int t,int k){
    int mid = tree[t].l + tree[t].r >> 1;
    ans = max(ans,k * tree[t].P + tree[t].S);
    if(tree[t].l == tree[t].r) return ;
    if(k <= mid) query(t << 1,k);
    else query(t << 1 | 1,k);
}
char op[10];
int main(){
    Sca(N);
    Build(1,1,50000);
    for(int i = 1; i <= N ; i ++){
        scanf("%s",op);
        if(op[0] == 'Q'){
            ans = 0;
            query(1,read());
            Pri((int)(ans / 100));
        }else{
            double S,P; scanf("%lf%lf",&S,&P);
            S -= P;
            update(1,S,P);
        }
    }
```

```
63      return 0;
64  }
65  //模板2
66  //1.加入x1,y1,x2,y2为端点的线段
67  //2.查询k点最大值
68  //注意x1,x2相同的时候需要特判
69  const int maxn = 4e4 + 10;
70  const int maxm = 1e5 + 10;
71  int N,M,K;
72  double S[maxm],P[maxm];
73  struct Tree{
74      int l,r;
75      int id;
76  }tree[maxn << 2];
77  void Build(int t,int l,int r){
78      tree[t].l = l; tree[t].r = r;
79      tree[t].id = 0;
80      if(l == r) return;
81      int m = l + r >> 1;
82      Build(t << 1,l,m);
83      Build(t << 1 | 1,m + 1,r);
84  }
85  bool dcmp(double a){
86      return fabs(a) > eps;
87  }
88  void update(int t,int l,int r,int id){
89      int m = (tree[t].l + tree[t].r) >> 1;
90      if(l <= tree[t].l && tree[t].r <= r){
91          int &a = id;
92          int &b = tree[t].id;
93          if(P[a] > P[b]) swap(a,b);
94          if(m * P[a] + S[a] > m * P[b] + S[b] || (!dcmp(m * P[a] + S[a] - m * P[b] + S[b
        ]) && a < b)){
95              swap(a,b);
96              if(tree[t].l == tree[t].r) return;
97              update(t << 1 | 1,l,r,id);
98          }else{
99              if(tree[t].l == tree[t].r) return;
100             update(t << 1,l,r,id);
101         }
102         return;
103     }
104     if(r <= m) update(t << 1,l,r,id);
105     else if(l > m) update(t << 1 | 1,l,r,id);
```

```
106     else{
107         update(t << 1,l,m,id);
108         update(t << 1 | 1,m + 1,r,id);
109     }
110 }
111 int Id;
112 void query(int t,int k){
113     int a = tree[t].id;
114     double Ans = P[Id] * k + S[Id];
115     if(Ans < P[a] * k + S[a] || (!dcmp(Ans - P[a] * k - S[a]) && Id > a)){
116         Id = a;
117     }
118     if(tree[t].l == tree[t].r) return;
119     int m = (tree[t].l + tree[t].r) >> 1;
120     if(k <= m) query(t << 1,k);
121     else query(t << 1 | 1,k);
122 }
123 int main(){
124     Sca(N); Id = 0; int tot = 0;
125     Build(1,1,40000);
126     for(int i = 1; i <= N ; i ++){
127         int op = read();
128         if(!op){
129             int k = (read() + Id - 1) % 39989 + 1;
130             Id = 0; query(1,k);
131             Pri(Id);
132         }else{
133             int x0 = (read() + Id - 1) % 39989 + 1;
134             int y0 = (read() + Id - 1) % 1000000000 + 1;
135             int x1 = (read() + Id - 1) % 39989 + 1;
136             int y1 = (read() + Id - 1) % 1000000000 + 1;
137             tot++;
138             if(x0 == x1){
139                 P[tot] = 0;
140                 S[tot] = max(y1,y0);
141             }else{
142                 P[tot] = 1.0 * (y1 - y0) / (x1 - x0);
143                 S[tot] = y1 - x1 * P[tot];
144             }
145             update(1,min(x0,x1),max(x0,x1),tot);
146         }
147     }
148     return 0;
149 }
```

```
150 //模板3
151 /*
152 题意: 树链上每个点添加形如a * dis + b的值的一个点, 每次查询树链上最小点
153 树链剖分 + 李超树
154 关于李超树的区间查询:
155 用一个Min维护区间最小值,Min的组成来源为两个子树的Min的较小值和本结点优势线段在两个端点的较小
        值
156 如果查询包含整个区间则直接返回Min, 否则来源为正常区间最小值查找 + 本结点优势线段在查询两端的
        较小值
157 */
158 const int maxn = 1e5 + 10;
159 const LL INF = 123456789123456789;
160 int N,M,K;
161 struct Edge{
162     int to,next;
163     LL dis;
164 }edge[maxn * 2];
165 int head[maxn],tot;
166 void init(){
167     for(int i = 0 ; i <= N ; i ++) head[i] = -1;
168     tot = 0;
169 }
170 void add(int u,int v,LL w){
171     edge[tot].to = v;
172     edge[tot].next = head[u];
173     edge[tot].dis = w;
174     head[u] = tot++;
175 }
176 int dep[maxn],top[maxn],fa[maxn],pos[maxn];
177 int sz[maxn],son[maxn],to[maxn];
178 LL dis[maxn];
179 void dfs1(int t,int la){
180     sz[t] = 1; son[t] = t;
181     int heavy = 0;
182     for(int i = head[t]; ~i ; i = edge[i].next){
183         int v = edge[i].to;
184         if(v == la) continue;
185         dis[v] = dis[t] + edge[i].dis;
186         dep[v] = dep[t] + 1;
187         fa[v] = t;
188         dfs1(v,t);
189         if(sz[v] > heavy){
190             heavy = sz[v];
191             son[t] = v;
```

```
        }
        sz[t] += sz[v];
    }
}
int cnt;
void dfs2(int t,int la){
    top[t] = la;
    pos[t] = ++cnt; to[cnt] = t;
    if(son[t] == t) return;
    dfs2(son[t],la);
    for(int i = head[t]; ~i ; i = edge[i].next){
        int v = edge[i].to;
        if((fa[t] == v) || (v == son[t])) continue;
        dfs2(v,v);
    }
}
LL S[maxn * 2],P[maxn * 2];
int ttt;
struct Tree{
    int l,r,id;
    LL Min;
}tree[maxn << 2];
void Pushup(int t){
    if(tree[t].l == tree[t].r) return;
    tree[t].Min = min(tree[t].Min,min(tree[t << 1].Min,tree[t << 1 | 1].Min));
}
void Build(int t,int l,int r){
    tree[t].l = l; tree[t].r = r;
    tree[t].id = 0; tree[t].Min = INF;
    if(l == r) return;
    int m = l + r >> 1;
    Build(t << 1,l,m); Build(t << 1 | 1,m + 1,r);
}
void update(int t,int l,int r,int id){  //区间查询
    int m = tree[t].l + tree[t].r >> 1;
    if(l <= tree[t].l && tree[t].r <= r){
        int &a = id; int &b = tree[t].id;
        if(P[a] > P[b]) swap(a,b);
        if(dis[to[m]] * P[a] + S[a] < dis[to[m]] * P[b] + S[b]){
            swap(a,b);
            if(tree[t].l != tree[t].r) update(t << 1,l,r,id);
        }else{
            if(tree[t].l != tree[t].r) update(t << 1 | 1,l,r,id);
        }
```

```
236         Pushup(t);
237         tree[t].Min = min(tree[t].Min,dis[to[tree[t].l]] * P[b] + S[b]);
238         tree[t].Min = min(tree[t].Min,dis[to[tree[t].r]] * P[b] + S[b]);
239         return;
240     }
241     if(r <= m) update(t << 1,l,r,id);
242     else if(l > m) update(t << 1 | 1,l,r,id);
243     else{
244         update(t << 1,l,m,id);
245         update(t << 1 | 1,m + 1,r,id);
246     }
247     Pushup(t);
248 }
249 LL query(int t,int l,int r){
250     if(l <= tree[t].l && tree[t].r <= r) return tree[t].Min;
251     int m = (tree[t].l + tree[t].r) >> 1;
252     LL ans = INF;
253     ans = min(ans,dis[to[l]] * P[tree[t].id] + S[tree[t].id]);
254     ans = min(ans,dis[to[r]] * P[tree[t].id] + S[tree[t].id]);
255     if(r <= m) return min(ans,query(t << 1,l,r));
256     else if(l > m) return min(ans,query(t << 1 | 1,l,r));
257     else{
258         return min(ans,min(query(t << 1,l,m),query(t << 1 | 1,m + 1,r)));
259     }
260 }
261 int lca(int x,int y){
262     while(top[x] != top[y]) dep[top[x]] > dep[top[y]]?x = fa[top[x]]:y = fa[top[y]];
263     return dep[x] < dep[y]?x:y;
264 }
265 void update(int u,int v,int id){
266     while(top[u] != top[v]){
267         if(dep[top[u]] < dep[top[v]]) swap(u,v);
268         update(1,pos[top[u]],pos[u],id);
269         u = fa[top[u]];
270     }
271     if(dep[u] > dep[v]) swap(u,v);
272     update(1,pos[u],pos[v],id);
273 }
274 LL query(int u,int v){
275     LL ans = INF;
276     while(top[u] != top[v]){
277         if(dep[top[u]] < dep[top[v]]) swap(u,v);
278         ans = min(ans,query(1,pos[top[u]],pos[u]));
279         u = fa[top[u]];
```

```
280     }
281     if(dep[u] > dep[v]) swap(u,v);
282     ans = min(ans,query(1,pos[u],pos[v]));
283     return ans;
284 }
285 int main(){
286     Sca2(N,M); init();
287     P[0] = 0; S[0] = INF;
288     for(int i = 1; i <= N - 1; i ++){
289         int u = read(),v = read();
290         LL w = read();
291         add(u,v,w); add(v,u,w);
292     }
293     int root = 1;
294     dfs1(root,-1); cnt = 0;
295     dfs2(root,root);
296     Build(1,1,N); ttt = 0;
297     while(M--){
298         int op = read();
299         if(op == 1){
300             int s = read(),t = read();
301             LL a = read(),b = read();
302             int x = lca(s,t);
303             ttt++; P[ttt] = -a; S[ttt] = a * dis[s] + b;
304             update(s,x,ttt);
305             ttt++; P[ttt] = a; S[ttt] = a * (dis[s] - 2 * dis[x]) + b;
306             update(x,t,ttt);
307         }else{
308             int s = read(),t = read();
309             Prl(query(s,t));
310         }
311     }
312     return 0;
313 }
```

7.11 左偏树

</> 代码 7.17: /数据结构/左偏树

```
1 //左偏树(可并堆)
2 //用来合并两个堆(优先队列)的数据结构
3 /*
4 模板 op1:合并两个下标所在的堆
5 op:2输出下标所在堆的最小值并pop, key值相同先pop下标小的那个
```

```
做法：维护每个小根堆
*/
const int maxn = 1e5 +10;

int N,M;
struct node{
    int l,r,dis,key;
}tree[maxn];
int fa[maxn];
int find(int x){
    if(fa[x] == x) return x;
    return fa[x] = find(fa[x]);   //路径压缩
}
int merge(int a,int b){
    if(!a) return b;
    if(!b) return a;
    if(tree[a].key >tree[b].key) swap(a,b);
    if((tree[a].key == tree[b].key) && (a > b)) swap(a,b);
    tree[a].r = merge(tree[a].r,b);
    fa[tree[a].r] = a;
    if(tree[tree[a].l].dis < tree[tree[a].r].dis)
        swap(tree[a].l,tree[a].r);
    if(tree[a].r) tree[a].dis = tree[tree[a].r].dis + 1;
    else tree[a].dis = 0;
    return a;
}
bool vis[maxn];
int pop(int a){
    int l = tree[a].l;
    int r = tree[a].r;
    fa[l] = l; fa[r] = r;
    vis[a] = 1;
    return fa[a]= merge(l,r);  //少了这部不能路径压缩
}
int a[maxn];

int main(){
    scanf("%d%d",&N,&M);
    for(int i = 1; i <= N ; i ++){
        scanf("%d",&tree[i].key); fa[i] = i;
        tree[i].l = tree[i].r = tree[i].dis = 0;
    }
    for(int i = 1; i <= M ; i ++){
        int op; scanf("%d",&op);
```

```
50          if(op == 1){
51              int x,y; scanf("%d%d",&x,&y);
52              if(vis[x] || vis[y]) continue;
53              x = find(x); y = find(y);
54              if(x == y) continue;
55              merge(x,y);
56          }else{
57              int x; scanf("%d",&x);
58              if(vis[x]){
59                  puts("-1");continue;
60              }
61              x = find(x);
62              printf("%d\n",tree[x].key);
63              pop(x);
64          }
65      }
66      return 0;
67  }
```

7.12 Splay

</> 代码 7.18: /数据结构/Splay

```
1  //Splay 伸展树
2  /*
3  再二叉搜索树的基础上增加了伸展(splay)功能，降低了查询和插入的时间复杂度。
4  本质上是通过将每一次查询的节点都进行伸展使其靠近根节点
5   */
6   //luogu 普通平衡树
7  const int maxn = 1e5 + 10;
8  const int INF = 0x3f3f3f3f;
9  const int mod = 1e9 + 7;
10 int N,M,K;
11 struct Splay{
12     #define root e[0].ch[1]
13     struct node{
14         int ch[2];
15         int sum,num;
16         int v,fa;
17     }e[maxn];
18     int n,points;
19     void update(int x){
20         e[x].sum = e[e[x].ch[0]].sum + e[e[x].ch[1]].sum + e[x].num;
21     }
```

```
22      int id(int x){
23          return x == e[e[x].fa].ch[0]?0:1;
24      }
25      void connect(int x,int y,int p){
26          e[x].fa = y;
27          e[y].ch[p] = x;
28      }
29      int find(int v){
30          int now = root;
31          while(1){
32              if(e[now].v == v){
33                  splay(now,root);
34                  return now;
35              }
36              int next = v < e[now].v?0:1;
37              if(!e[now].ch[next]) return 0;
38              now = e[now].ch[next];
39          }
40          return 0;
41      }
42      void rotate(int x){
43          int y = e[x].fa;
44          int z = e[y].fa;
45          int ix = id(x),iy = id(y);
46          connect(e[x].ch[ix ^ 1],y,ix);
47          connect(y,x,ix ^ 1);
48          connect(x,z,iy);
49          update(y); update(x);
50      }
51      void splay(int u,int v){
52          v = e[v].fa;
53          while(e[u].fa != v){
54              int fu = e[u].fa;
55              if(e[fu].fa == v) rotate(u);
56              else if(id(u) == id(fu)){
57                  rotate(fu);
58                  rotate(u);
59              }else{
60                  rotate(u);
61                  rotate(u);
62              }
63          }
64      }
65      int crenode(int v,int father){
```

```
66          n++;
67          e[n].ch[0] = e[n].ch[1] = 0;
68          e[n].fa = father;
69          e[n].num = e[n].sum = 1;
70          e[n].v = v;
71          return n;
72      }
73      void destroy(int x){
74          e[x].v = e[x].fa = e[x].num = e[x].sum = e[x].v = 0;
75          if (x == n) n--;
76      }
77      int insert(int v){
78          points++;
79          if(points == 1){
80              n = 0;
81              root = 1;
82              crenode(v,0);
83              return 1;
84          }else{
85              int now = root;
86              while(1){
87                  e[now].sum++;
88                  if(v == e[now].v){
89                      e[now].num++;
90                      return now;
91                  }
92                  int next = v < e[now].v?0:1;
93                  if(!e[now].ch[next]){
94                      crenode(v,now);
95                      e[now].ch[next] = n;
96                      return n;
97                  }
98                  now = e[now].ch[next];
99              }
100         }
101     }
102     void push(int v){  //添加元素
103         int add = insert(v);
104         splay(add,root);
105     }
106     void pop(int x){
107         int pos = find(x);  //找到节点并作为根
108         if(!pos) return;
109         points--;
```

```
110         if(e[pos].num > 1){
111             e[pos].num--;
112             e[pos].sum--;   //他已经是根节点了，不需要更新祖先的sum了
113             return;
114         }
115         if(!e[pos].ch[0]){       //没有左孩子就直接删除，右孩子作为根
116             root = e[pos].ch[1];
117             e[root].fa = 0;
118         }else{          //有左孩子：  将左子树最大的点转到根，将右子树连到它上
119             int lef = e[pos].ch[0];
120             while(e[lef].ch[1]) lef = e[lef].ch[1];
121             splay(lef,e[pos].ch[0]);
122             int rig = e[pos].ch[1];
123             connect(rig,lef,1); connect(lef,0,1);
124             update(lef);
125         }
126         destroy(pos);
127     }
128     int atrank(int x){
129         if(x > points) return -INF;
130         int now = root;
131         while(1){
132             int mid = e[now].sum - e[e[now].ch[1]].sum;
133             if(x > mid){
134                 x -= mid;
135                 now = e[now].ch[1];
136             }else if(x <= e[e[now].ch[0]].sum){
137                 now = e[now].ch[0];
138             }else break;
139         }
140         splay(now,root);
141         return e[now].v;
142     }
143     int rank(int x){
144         int now = find(x);
145         if(!now) return 0;
146         return e[e[now].ch[0]].sum + 1;
147     }
148     int upper(int v){
149         int now = root;
150         int ans = INF;
151         while(now){
152             if(e[now].v > v && e[now].v < ans) ans = e[now].v;
153             if(v < e[now].v) now = e[now].ch[0];
```

```
154             else now = e[now].ch[1];
155         }
156         return ans;
157     }
158     int lower(int v){
159         int now = root;
160         int ans = -INF;
161         while(now){
162             if(e[now].v < v && e[now].v > ans) ans = e[now].v;
163             if(v > e[now].v) now = e[now].ch[1];
164             else now = e[now].ch[0];
165         }
166         return ans;
167     }
168     #undef root
169 }F;
170 int main(){
171     Sca(N);
172     while(N--){
173         int x,op; Sca2(op,x);
174         if(op == 1) F.push(x);     //插入一个元素
175         else if(op == 2) F.pop(x);   //删除一个元素
176         else if(op == 3) Pri(F.rank(x));   //查询x数的排名(排名定义为比当前数小的数的个数
        +1+1。若有多个相同的数，因输出最小的排名)
177         else if(op == 4) Pri(F.atrank(x)); //查询排名为x的数
178         else if(op == 5) Pri(F.lower(x));  //求xx的前驱(前驱定义为小于xx，且最大的数)
179         else Pri(F.upper(x)); //求xx的后继(后继定义为大于xx，且最小的数)
180     }
181     return 0;
182 }
183
184 //Splay合并咋整鸭
185 //启发式合并，将小的合并到大的上去，用中序遍历的方式暴力插入，时间复杂度lnlogn级别
186 /* 题意
187 永无乡包含n座岛，编号从 1到 n，
188 每座岛都有自己的独一无二的重要度，按照重要度可以将这 n座岛排名，
189 名次用 1到 n来表示。某些岛之间由巨大的桥连接，通过桥可以从一个岛到达另一个岛。
190 如果从岛 a出发经过若干座（含 0座）桥可以到达岛 b,则称岛 a和岛 b是连通的。
191 B x y 表示在岛x与岛y之间修建一座新桥。
192 Q x k 表示询问当前与岛x连通的所有岛中第k重要的是哪座岛，
193 即所有与岛x连通的岛中重要度排名第k小的岛是哪座，请你输出那个岛的编号。
194  */
195  const int maxn = 5e5 + 10;
196 const int maxm = 3e5 + 10;
```

```
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct node{
    int ch[2];
    int sum,v,fa,id;
}e[maxn];
int n,points[maxn];
int root[maxn],tree[maxn];
int crenode(int v,int id,int fa){
    n++;
    e[n].ch[0] = e[n].ch[1] = 0;
    e[n].fa = fa;
    e[n].sum = 1;
    e[n].v = v; e[n].id = id;
    return n;
}
int id(int x){
    return x == e[e[x].fa].ch[0]?0:1;
}
void connect(int u,int v,int p){
    e[u].fa = v; e[v].ch[p] = u;
}
void update(int x){
    e[x].sum = e[e[x].ch[0]].sum + e[e[x].ch[1]].sum + 1;
}
int insert(int t,int num){
    points[num]++;
    int now = root[num];
    int v = e[t].v;
    while(1){
        e[now].sum++;
        int nxt = v < e[now].v?0:1;
        if(!e[now].ch[nxt]){
            connect(t,now,nxt);
            e[t].ch[0] = e[t].ch[1] = 0;
            e[t].sum = 1;
            return t;
        }
        now = e[now].ch[nxt];
    }
}

void rotate(int x){
```

```
241     int y = e[x].fa;
242     int z = e[y].fa;
243     int ix = id(x),iy = id(y);
244     connect(e[x].ch[ix ^ 1],y,ix);
245     connect(y,x,ix ^ 1);
246     connect(x,z,iy);
247     update(y); update(x);
248 }
249 void splay(int u,int v){
250     v = e[v].fa;
251     while(e[u].fa != v){
252         int fu = e[u].fa;
253         if(e[fu].fa == v) rotate(u);
254         else if(id(u) == id(fu)){
255             rotate(fu);
256             rotate(u);
257         }else{
258             rotate(u);
259             rotate(u);
260         }
261     }
262     if(v <= N) root[v] = u;
263 }
264
265 void push(int t,int num){
266     int x = insert(t,num);
267     splay(x,root[num]);
268 }
269 int find(int p){
270     return p == tree[p]?p:tree[p] = find(tree[p]);
271 }
272 void dfs(int t,int num){
273     int l = e[t].ch[0],r = e[t].ch[1];
274     if(l) dfs(l,num);
275     push(t,num);        //中序遍历合并是log复杂度
276     if(r) dfs(r,num);
277 }
278 void Union(int u,int v){
279     int x = find(u),y = find(v);
280     if(x == y) return;
281     if(points[x] > points[y]) swap(x,y);
282     tree[x] = y;
283     dfs(root[x],y);
284 }
```

```
285 int Kth(int id,int k){
286     if(points[id] < k) return -1;
287     int now = root[id];
288     while(1){
289         int mid = e[now].sum - e[e[now].ch[1]].sum;
290         if(k > mid){
291             k -= mid;
292             now = e[now].ch[1];
293         }else if(k <= e[e[now].ch[0]].sum){
294             now = e[now].ch[0];
295         }else break;
296     }
297     splay(now,root[id]);
298     return e[now].id;
299 }
300 int main(){
301     Sca2(N,M);
302     n = N;      //1 ~ N表示每颗平衡树的0点
303     for(int i = 1; i <= N ; i ++){
304         tree[i] = i;
305         int v; Sca(v);
306         points[i] = 1;
307         root[i] = crenode(v,i,i);
308     }
309     for(int i = 1; i <= M; i ++){
310         int u,v; Sca2(u,v);
311         Union(v,u);
312     }
313     int Q; Sca(Q);
314     while(Q--){
315         char op[2]; int x,y;
316         scanf("%s%d%d",op,&x,&y);
317         if(op[0] == 'B') Union(x,y);
318         else Pri(Kth(find(x),y));
319     }
320     return 0;
321 }
```

7.13 LCT

</> 代码 7.19: /数据结构/LCT

```
1 //link cut tree (LCT)
2 /* lct的性质
```

```
    1.lct是将一棵树分为若干的Splay
    每一个Splay维护的是一条从上到下按在原树中深度严格递增的路径，且中序遍历Splay得到的每个点的深度序列严格递增。
    2.每个节点包含且仅包含于一个Splay中
    3.边分为实边和虚边，实边包含在Splay中，而虚边总是由一棵Splay指向另一个节点（指向该Splay中中序遍历最靠前的点在原树中的父亲）。因为性质2，当某点在原树中有多个儿子时，只能向其中一个儿子拉一条实链（只认一个儿子），而其它儿子是不能在这个Splay中的。那么为了保持树的形状，我们要让到其它儿子的边变为虚边，由对应儿子所属的Splay的根节点的父亲指向该点，而从该点并不能直接访问该儿子（认父不认子）。
 */
const int maxn = 3e5 + 10;
int N,M,K;
#define lc e[x].ch[0]
#define rc e[x].ch[1]
#define fc e[x].fa
struct node{
    int fa,ch[2];
    int v,s;   //v存储当前节点信息，s存储子树的异或和
    bool rev;
}e[maxn];
int Stack[maxn];
//1轻边重边主要看父亲认不认儿子（儿子一定认父亲）
void pushup(int x){
    e[x].s = e[lc].s ^ e[rc].s ^ e[x].v;
}
bool nroot(int x){   //判断节点是否为一个Splay的根 ,返回1代表不是根
    return e[fc].ch[0] == x || e[fc].ch[1] == x; //如果是根，他的父亲和他之间是轻边（父亲不认儿子）
}
void rev(int x){     //翻转子树
    swap(lc,rc); e[x].rev ^= 1;
}
void pushdown(int x){   //向下更新
    if(e[x].rev){
        if(lc) rev(lc);
        if(rc) rev(rc);
        e[x].rev = 0;
    }
}
void rotate(int x){    //在x在自己的splay里向上旋转
    int y = e[x].fa; int z = e[y].fa;
    int ix = e[y].ch[1] == x; int iy = e[z].ch[1] == y;
    int w = e[x].ch[ix ^ 1];
    if(nroot(y)) e[z].ch[iy] = x;   //不判断会错
```

```
41      if(w) e[w].fa = y;
42      e[x].ch[ix ^ 1] = y; e[y].ch[ix] = w;
43      e[y].fa = x; e[x].fa = z;
44      pushup(y);
45  }
46  void splay(int x){ //只传了一个参数的原因是所有目标都是该Splay的根
47      int y = x,top = 0;
48      Stack[++top] = y;
49      while(nroot(y)) Stack[++top] = y = e[y].fa;
50      while(top) pushdown(Stack[top--]);  //先下方整条路径上的信息
51      while(nroot(x)){
52          y = e[x].fa; int z = e[y].fa;
53          if(nroot(y)) rotate((e[y].ch[0] == x) ^ (e[z].ch[0] == y)?x:y);  //直连就转父亲,
    否则转自己
54          rotate(x);
55      }
56      pushup(x);
57  }
58  void access(int x){  //联通x到根，使他们在同一个Splay
59      for(int y = 0;x;x = e[x].fa){
60          splay(x);
61          e[x].ch[1] = y;
62          pushup(x);
63          y = x;
64      }
65  }
66  //换原树的根
67  void makeroot(int x){
68      access(x); splay(x); rev(x);
69  }
70  //找根,即x所属的splay中深度最浅的那个点
71  //indroot(x)==findroot(y)表明x,y在同一棵树中
72  int findroot(int x){
73      access(x); splay(x);
74      while(lc) pushdown(x),x = lc;
75      splay(x);
76      return x;
77  }
78  //拉出x-y的路径成为一个Splay（以y作为该Splay的根）
79  void split(int x,int y){
80      makeroot(x);
81      access(y); splay(y);
82  }
83  //连一条x-y的边（使x的父亲指向y，连一条轻边）
```

```
84  void link(int x,int y){
85      makeroot(x);
86      if(findroot(y) != x) e[x].fa = y;
87  }
88  //将x-y的边断开
89  void cut(int x,int y){
90      makeroot(x);
91      if(findroot(y) == x && e[y].fa == x && !e[y].ch[0]){
92          e[y].fa = e[x].ch[1] = 0;
93          pushup(x);
94      }
95  }
96  /* 如果保证断的边都存在可以这么写
97  void cut(int x,int y){
98      split(y,x);
99      e[y].fa = e[x].ch[1] = 0;
100     pushup(x);
101 }
102  */
103 int main(){
104     Sca2(N,M);
105     for(int i = 1; i <= N; i ++) Sca(e[i].v);
106     while(M--){
107         int op,x,y; Sca3(op,x,y);
108         switch(op){
109         case 0:split(x,y); Pri(e[y].s); break;
110         case 1:link(x,y); break;
111         case 2:cut(x,y); break;
112         case 3:splay(x); e[x].v = y; break;
113         }
114     }
115     return 0;
116 }
117
118 //LCT的边转点
119 /*
120 树链剖分等算法的边转点一般是将边转化为深度交深的点
121 当用LCT维护类似于树链上最大的边的时候，可以采用(u,v)这条边转化为(u,id),(id,v)的方式，其中
122 id为u,v之间的边，即将边看作是一个点
123 模板：luoguP4172 水管局长
124 给一个n个点m条边的图，支持以下操作：
125 1.询问x到y所有路径中，路径上最大边权的最小值
126 2.删去一条边
127 做法:离线之后变为加边，用LCT维护一个最小生成树
```

```
*/

const double eps = 1e-9;
const int maxn = 2e5 + 10;
const int maxm = 1500000;
const int maxq = 1e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K,Q;
struct Edge{
    int u,v,t;
}edge[maxm + maxn];
struct Q{
    int k,x,y;
}q[maxm];
bool vis[maxm + maxn];
map<PII,int>ide;
int fa[maxm];
void init(){for(int i = 0 ; i <= N ; i ++) fa[i] = i;}
int find(int x){return x == fa[x]?x:fa[x] = find(fa[x]);}
void Union(int a,int b){
    a = find(a); b = find(b);
    fa[a] = b;
}
#define lc e[x].ch[0]
#define rc e[x].ch[1]
#define fc e[x].fa
struct node{
    int ch[2],fa;
    int iv,imax;
    bool rev;
}e[maxn + maxm];
void pushup(int x){
    int a = e[x].iv,b = e[lc].imax,c = e[rc].imax;
    if(edge[a].t < edge[b].t) a = b;
    if(edge[a].t < edge[c].t) a = c;
    e[x].imax = a;
}
bool nroot(int x){
    return e[fc].ch[0] == x || e[fc].ch[1] == x;
}
void rev(int x){
    swap(lc,rc); e[x].rev ^= 1;
}
```

```
172 void pushdown(int x){
173     if(e[x].rev){
174         if(lc) rev(lc);
175         if(rc) rev(rc);
176         e[x].rev = 0;
177     }
178 }
179 void rotate(int x){
180     int y = e[x].fa;int z = e[y].fa;
181     int ix = e[y].ch[1] == x;
182     int iy = e[z].ch[1] == y;
183     int w = e[x].ch[ix ^ 1];
184     if(nroot(y)) e[z].ch[iy] = x;
185     if(w) e[w].fa = y;
186     e[x].ch[ix ^ 1] = y; e[y].ch[ix] = w;
187     e[y].fa = x;  e[x].fa = z;
188     pushup(y);
189 }
190 int Stack[maxn + maxm];
191 void splay(int x){
192     int y = x,top = 0;
193     Stack[++top] = y;
194     while(nroot(y)) Stack[++top] = y = e[y].fa;
195     while(top) pushdown(Stack[top--]);
196     while(nroot(x)){
197         y = e[x].fa; int z = e[y].fa;
198         if(nroot(y)) rotate((e[y].ch[0] == x) ^ (e[z].ch[0] == y)?x:y);
199         rotate(x);
200     }
201     pushup(x);
202 }
203
204 void access(int x){
205     for(int y = 0;x;x = e[x].fa){
206         splay(x);
207         e[x].ch[1] = y;
208         pushup(x);
209         y = x;
210     }
211 }
212 void makeroot(int x){
213     access(x); splay(x); rev(x);
214 }
215 int findroot(int x){
```

```
216     access(x); splay(x);
217     while(lc) pushdown(x),x = lc;
218     splay(x);
219     return x;
220 }
221 void split(int x,int y){
222     makeroot(x);
223     access(y); splay(y);
224 }
225 void link(int x,int y){
226     makeroot(x);
227     if(findroot(y) != x) e[x].fa = y;
228 }
229 void cut(int x,int y){
230     split(y,x);
231     e[y].fa = e[x].ch[1] = 0;
232     pushup(x);
233 }
234 void add(int u,int v,int id){
235     if(find(u) != find(v)){
236         Union(u,v);
237         link(u,id); link(id,v);
238         return;
239     }
240     split(u,v);
241     int tmp = e[v].imax;
242     if(edge[tmp].t > edge[id].t){
243         cut(edge[tmp].u,tmp);
244         cut(tmp,edge[tmp].v);
245         link(u,id); link(id,v);
246     }
247 }
248 int ans[maxm];
249 bool cmp(Edge a,Edge b){
250     return a.t < b.t;
251 }
252 int main(){
253     N = read(),M = read(),Q = read();init();
254     edge[0].t = -1;
255     for(int i = 1; i <= N; i ++) e[i].imax = e[i].iv = 0;
256     for(int i = 1 + N; i <= M + N; i ++){
257         edge[i].u = read(); edge[i].v = read(); edge[i].t = read();
258         if(edge[i].u > edge[i].v) swap(edge[i].u,edge[i].v);
259     }
```

```
    sort(edge + 1 + N,edge + 1 + M + N,cmp);
    for(int i = 1 + N; i <= M + N; i ++){
        e[i].iv = e[i].imax = i;
        ide[mp(edge[i].u,edge[i].v)] = i;
    }
    for(int i = 1; i <= Q ; i ++){
        q[i].k = read(); q[i].x = read(); q[i].y = read();
        if(q[i].x > q[i].y) swap(q[i].x,q[i].y);
        if(q[i].k == 2) vis[ide[mp(q[i].x,q[i].y)]] = 1;
    }
    int cnt = 0;
    for(int i = 1 + N; i <= M + N && cnt < N - 1; i ++){
        if(vis[i]) continue;
        int u = edge[i].u,v = edge[i].v;
        if(find(u) == find(v)) continue;
        cnt++;
        Union(u,v); link(u,i); link(i,v);
    }
    int tot = 0;
    for(int i = Q; i >= 1; i --){
        if(q[i].k == 1){
            split(q[i].x,q[i].y);
            ans[++tot] = edge[e[q[i].y].imax].t;
        }else{
            add(q[i].x,q[i].y,ide[mp(q[i].x,q[i].y)]);
        }
    }
    for(int i = tot; i >= 1; i --) Pri(ans[i]);
    return 0;
}
```

7.14 主席树

7.14.1 静态区间第 k 小

</> 代码 7.20: /数据结构/静态区间第 k 小

```
//静态区间第K小 nlogn
const int maxn = 2e5 + 10;
int N,M,K;
int Hash[maxn],a[maxn];
int T[maxn];
struct Tree{
    int lt,rt,sum;
    void init(){
```

```
9           lt = rt = sum = 0;
10      }
11  }tree[maxn * 60];
12  int tot;
13  void newnode(int &t){
14      t = ++tot;
15      tree[t].init();
16  }
17  void Build(int &t,int l,int r){
18      newnode(t);
19      if(l == r) return;
20      int m = l + r >> 1;
21      Build(tree[t].lt,l,m); Build(tree[t].rt,m + 1,r);
22  }
23  void update(int &t,int pre,int l,int r,int p){
24      newnode(t);
25      tree[t] = tree[pre];
26      tree[t].sum++;
27      if(l == r) return;
28      int m = l + r >> 1;
29      if(p <= m) update(tree[t].lt,tree[pre].lt,l,m,p);
30      else update(tree[t].rt,tree[pre].rt,m + 1,r,p);
31  }
32  int query(int L,int R,int l,int r,int k){
33      if(l >= r) return l;
34      int num = tree[tree[R].lt].sum - tree[tree[L].lt].sum;
35      int m = l + r >> 1;
36      if(num >= k) return query(tree[L].lt,tree[R].lt,l,m,k);
37      else return query(tree[L].rt,tree[R].rt,m + 1,r,k - num);
38  }
39  int main(){
40      Sca2(N,M);
41      for(int i = 1; i <= N ; i ++) Hash[i] = a[i] = read();
42      sort(Hash + 1,Hash + 1 + N);
43      int cnt = unique(Hash + 1,Hash + 1 + N) - Hash - 1;
44      Build(T[0],1,cnt);
45      for(int i = 1; i <= N ; i ++){
46          int p = lower_bound(Hash + 1,Hash + 1 + cnt,a[i]) - Hash;
47          update(T[i],T[i - 1],1,cnt,p);
48      }
49      for(int i = 1; i <= M ; i ++){
50          int l = read(),r = read(),k = read();
51          Pri(Hash[query(T[l - 1],T[r],1,cnt,k)]);
52      }
```

```
53      return 0;
54  }
```

7.15 cdq 分治

7.15.1 三维偏序问题

</> 代码 7.21: /数据结构/三维偏序

```
1   //cdq分治：陌上花开
2   //定义node的等级为xyz全不大于他的node的种数，求每个等级的node有多少种
3   //注意需要先将xyz完全相同的node合并才行
4   const double eps = 1e-9;
5   const int maxn = 1e5 + 10;
6   const int maxm = 2e5 + 10;
7   int N,M,K;
8   struct Node{
9       int x,y,z,ans,cnt;
10      Node(){}
11      Node(int x,int y,int z,int ans,int cnt):x(x),y(y),z(z),ans(ans),cnt(cnt){}
12      bool operator == (Node a){
13          return x == a.x && y == a.y && z == a.z;
14      }
15  }node[maxn],tmp[maxn];
16  int ans[maxn],sum[maxn];
17  int tree[maxm];
18  void add(int x,int y){
19      for(;x <= M; x += x & -x) tree[x] += y;
20  }
21  int getsum(int x){
22      int ans = 0;
23      for(;x > 0; x -= x & -x) ans += tree[x];
24      return ans;
25  }
26  bool cmp(Node a,Node b){
27      if(a.x != b.x) return a.x < b.x;
28      if(a.y != b.y) return a.y < b.y;
29      return a.z < b.z;
30  }
31  void cdq(int l,int r){
32      if(l >= r) return;
33      int m = l + r >> 1;
34      cdq(l,m); cdq(m + 1,r);
35      int cnt1 = l,cnt2 = m + 1;
36      for(int i = l; i <= r; i ++){
```

```
        if((cnt2 > r) || ((node[cnt1].y <= node[cnt2].y) && cnt1 <= m)){
            add(node[cnt1].z,node[cnt1].cnt);
            tmp[i] = node[cnt1++];
        }else{
            node[cnt2].ans += getsum(node[cnt2].z);
            tmp[i] = node[cnt2++];
        }
    }
    for(int i = l; i <= m; i ++) add(node[i].z,-node[i].cnt);
    for(int i = l; i <= r; i ++) node[i] = tmp[i];
}
int main(){
    Sca2(N,M); int cnt = 1;
    for(int i = 1; i <= N ; i ++) node[i] = Node(read(),read(),read(),0,1);
    sort(node + 1,node + 1 + N,cmp);
    for(int i = 2; i <= N ; i ++){
        if(node[i - 1] == node[i]) node[cnt].cnt++;
        else node[++cnt] = node[i];
    }
    cdq(1,cnt);
    for(int i = 1; i <= cnt ; i ++) sum[node[i].ans + node[i].cnt - 1] += node[i].cnt;
    for(int i = 0; i < N ; i ++) Pri(sum[i]);
    return 0;
}
```

7.16 kd 树

</> 代码 7.22: /数据结构/kd 树

```
/* K-DTree是用来解决K维空间中数点问题强有力的数据结构，可以在 (NlogN)-(N√N)的时间复杂度内完成查询和修改。
建树类似于二叉平衡树，只是排序的依据每个维度轮流，第一层取所有节点0维度的中位数点作为结点，比他小的放左边，比他大的放右边
第二层就按照1维度来排序....
每个节点除了表示一个当前节点外，还存储子树信息，查询的时候如果有子树显然不符合条件，就skip掉(类似于剪枝)
*/
//模板1HDU4347.给出N个K维度点，查询的时候给个点和M，询问N个点中前M靠近给定点的点
const int maxn = 50010;
const int maxk = 7;
int N,M,K;
int idx;
struct point{
    int x[maxk];
```

```
13      bool operator < (const point &u) const{
14          return x[idx] < u.x[idx];  //每层排序的维度都不一样,用全局变量idx标识
15      }
16  }p[maxn];
17  typedef pair<double,point>tp;
18  priority_queue<tp>nq;    //维护离查询点最近的点,fi表示距离,se表示点坐标
19  int pow2(int x){return x * x;}
20  struct kdTree{
21      point pt[maxn << 2]; //结点表示的点坐标
22      int son[maxn << 2];  //区间长度
23      void build(int t,int l,int r,int dep = 0){
24          if(l > r) return;
25          son[t] = r - l;
26          son[t << 1] = son[t << 1 | 1] = -1;
27          idx = dep % K;
28          int mid = l + r >> 1;
29          nth_element(p + l,p + mid,p + r + 1);  //当前idx维度排名为中位数的点作为这个结点表
        示的点
30          pt[t] = p[mid];
31          build(t << 1,l,mid - 1,dep + 1);
32          build(t << 1 | 1,mid + 1,r,dep + 1);
33      }
34      void query(int t,point q,int m,int dep = 0){
35          if(son[t] == -1) return;
36          tp nd(0,pt[t]);
37          for(int i = 0 ; i < K; i ++) nd.first += pow2(nd.se.x[i] - q.x[i]);
38          int dim = dep % K,x = t << 1,y = t << 1 | 1,flag = 0;
39          if(q.x[dim] >= pt[t].x[dim]) swap(x,y);   //如果待查寻的点这个维度比他大就查询右子
        树
40          if(~son[x]) query(x,q,m,dep + 1);
41          if(nq.size() < m) nq.push(nd),flag = 1;
42          else{
43              if(nd.first < nq.top().fi) nq.pop(),nq.push(nd);
44              if(pow2(q.x[dim] - pt[t].x[dim]) < nq.top().fi) flag = 1; //如果这个维度的距
        离已经比所有的都大了,就没有继续查的必要了
45          }
46          if(~son[y] && flag) query(y,q,m,dep + 1);
47      }
48  }kd;
49  void print(point &t){
50      for(int j = 0 ; j < K ; j ++) printf("%d%c",t.x[j],j == K - 1?'\n':' ');
51  }
52  int main(){
53      while(~Sca2(N,K)){
```

```
        for(int i = 0; i < N ; i ++){
            for(int j = 0 ; j < K; j ++) Sca(p[i].x[j]);
        }
        kd.build(1,0,N - 1);
        int t = read();
        while(t--){
            point ask;
            for(int j = 0 ; j < K ; j ++) Sca(ask.x[j]);
            Sca(M); kd.query(1,ask,M);
            printf("the closest %d points are:\n",M);
            point pt[20];
            for(int j = 0;!nq.empty(); j ++) pt[j] = nq.top().second,nq.pop();
            for(int j = M - 1; j >= 0 ; j --) print(pt[j]);
        }
    }
    return 0;
}
//当题目需要在线的时候,就不能用上面的方法构树了
/*insert函数动态加点,利用重构系数判断重构, 节点中Max,Min表示该子树中该维度的最大(小)值
重构kdtree的参数选择, 插入多查询少的情况, 最优参数是接近 0.5 + x / (x + y) 的
x是插入个数, y是查询个数
插入少查询多的话, 最优参数其实是更接近 0.7 + x / (x + y) 的, 查询再多也不建议参数低于0.7
当然最优参数的话, 有时间可以自己去测试调整
*/
//模板, 操作1在坐标(x,y)中加权值v,操作2查询子矩阵和
const int maxn = 1e5 + 10;
const int maxk = 2;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
int p[maxk];
int q[maxk][2];
struct Tree{
    int d[maxk];
    int l,r,sum,val,size;
    int Min[maxk],Max[maxk];
}tree[maxn];
int cnt = 0;
const double alpha = 0.75;  //重构系数, 当左(右)子树权重超过一定比例的时候将子树重构
int idx,num,tmp[maxn],A;
inline void erase(int &x){ //删除子树中的所有点,把这些点加入tmp
    if(!x) return;
    tmp[++num] = x;
    erase(tree[x].l); erase(tree[x].r);
```

```
98      x = 0;
99  }
100 bool cmp(const int &a,const int &b){    //按照idx维度的大小排序
101     return tree[a].d[idx] < tree[b].d[idx];
102 }
103 inline void update(int t){   //类似于Pushup
104     int l = tree[t].l,r = tree[t].r;
105     tree[t].size = tree[l].size + tree[r].size + 1;
106     tree[t].sum = tree[t].val + tree[l].sum + tree[r].sum;
107     for(int i = 0 ; i < K; i ++){
108         if(l){
109             tree[t].Max[i] = max(tree[t].Max[i],tree[l].Max[i]);
110             tree[t].Min[i] = min(tree[t].Min[i],tree[l].Min[i]);
111         }
112         if(r){
113             tree[t].Max[i] = max(tree[t].Max[i],tree[r].Max[i]);
114             tree[t].Min[i] = min(tree[t].Min[i],tree[r].Min[i]);
115         }
116     }
117 }
118 inline void build(int &t,int l,int r,int k){  //构造
119     if(l > r) return;
120     int mid = l + r >> 1; idx = k;
121     nth_element(tmp + l,tmp + 1 + mid,tmp + r + 1,cmp);
122     t = tmp[mid];
123     tree[t].sum = tree[t].val;
124     for(int i = 0; i < K; i ++) tree[t].Max[i] = tree[t].Min[i] = tree[t].d[i];
125     build(tree[t].l,l,mid - 1,(k + 1) % K);   //mid点就是t点,已经被加入了
126     build(tree[t].r,mid + 1,r,(k + 1) % K);
127     update(t);
128 }
129 inline void rebuild(int &t,int k){    //重构子树
130     tmp[num = 1] = ++cnt;
131     tree[cnt].size = 1;
132     for(int i = 0; i < K ; i ++) tree[cnt].d[i] = p[i];
133     tree[cnt].val = tree[cnt].sum = A;
134     erase(t);
135     build(t,1,num,k);
136 }
137 void insert(int& t,int k){
138     if(!t){
139         tree[t = ++cnt].size = 1;
140         tree[t].val = tree[t].sum = A;
141         for(int i = 0 ; i < K ; i ++){
```

```
142             tree[t].Max[i] = tree[t].Min[i] = tree[t].d[i] = p[i];
143         }
144         return;
145     }
146     if(p[k] < tree[t].d[k]){
147         if(tree[tree[t].l].size > tree[t].size * alpha) rebuild(t,k);
148         else insert(tree[t].l,(k + 1) % K);
149     }else{
150         if(tree[tree[t].r].size > tree[t].size * alpha) rebuild(t,k);
151         else insert(tree[t].r,(k + 1) % K);
152     }
153     update(t);
154 }
155 inline bool check_range(int t){  //检查这个区域是否被两个q完全包围
156     if(!t) return 0;
157     for(int i = 0 ; i < K ; i ++){
158         if(q[i][0] > tree[t].Min[i] || q[i][1] < tree[t].Max[i]) return 0;
159     }
160     return 1;
161 }
162 inline bool check_point(int t){   //检查点是否在子矩阵内
163     if(!t) return 0;
164     for(int i = 0 ; i < K ; i ++){
165         if(tree[t].d[i] < q[i][0] || tree[t].d[i] > q[i][1]) return 0;
166     }
167     return 1;
168 }
169 inline bool check(int t){ //检查是否两个区间是否有相交部分
170     if(!t) return false;
171     for(int i = 0 ; i < K ; i ++){
172         if(q[i][1] < tree[t].Min[i] || q[i][0] > tree[t].Max[i]) return false;
173     }
174     return true;
175 }
176 int ans;
177 inline void query(int t){
178     if(check_range(t)){
179         ans += tree[t].sum;
180         return;
181     }
182     if(check_point(t)) ans += tree[t].val;
183     if(check(tree[t].l)) query(tree[t].l);
184     if(check(tree[t].r)) query(tree[t].r);
185 }
```

```
186 int main(){
187     N = read(); K = 2;   //默认为平面(2维)
188     int root = 0;
189     while(1){
190         int op = read();
191         if(op == 1){
192             for(int i = 0 ; i < K ; i ++) p[i] = read() ^ ans;
193             A = read() ^ ans;
194             insert(root,0);
195         }else if(op == 2){
196             for(int i = 0; i <= 1; i ++){
197                 for(int j = 0 ; j < K ;j ++) q[j][i] = read() ^ ans;
198             }
199             ans = 0; query(root);
200             Pri(ans);
201         }else{
202             break;
203         }
204     }
205     return 0;
206 }
```

第八章　动态规划

8.1　悬线法 dp

</> 代码 8.1:　/动态规划/悬线法 dp

```
//悬线法dp，用于处理矩阵相关的dp，例如求解最大的满足某个限制的长方形（正方形）
//用Left Right两个数组记录这个位置最左端和最右端的满足题目限制的条件。
//用up数组记录 这个位置在满足了左右两端条件之后可向上扩充的最大长度
//然后使用up * (Right - Left + 1)的方法更新答案即可。

//模板：ZJOI2007 棋盘制作 寻找最大的满足01交错的长方形和正方形
int Left[maxn][maxn],Right[maxn][maxn],up[maxn][maxn];
int MAP[maxn][maxn];
int main()
{
    Sca2(N,M);
    for(int i = 1; i <= N ; i ++){
        for(int j = 1; j <= M ; j ++){
            Sca(MAP[i][j]);
            Left[i][j] = Right[i][j] = j;
            up[i][j] = 1;
        }
    }
    for(int i = 1; i <= N ; i ++){
        for(int j = 2; j <= M ; j ++){
            if(MAP[i][j] != MAP[i][j - 1]){
                Left[i][j] = Left[i][j - 1];
            }
        }
        for(int j = M - 1; j >= 1; j --){
            if(MAP[i][j] != MAP[i][j + 1]){
                Right[i][j] = Right[i][j + 1];
            }
        }
    }
    int ans1 = 0,ans2 = 0;
    for(int i = 1; i <= N ; i ++){
```

```
        for(int j = 1; j <= M ; j ++){
            if(i > 1 && MAP[i][j] != MAP[i - 1][j]){
                Left[i][j] = max(Left[i][j],Left[i - 1][j]);
                Right[i][j] = min(Right[i][j],Right[i - 1][j]);
                up[i][j] = up[i - 1][j] + 1;
            }
            int a = Right[i][j] - Left[i][j] + 1;
            int b = min(a,up[i][j]);
            ans1 = max(ans1,b * b);
            ans2 = max(ans2,a * up[i][j]);
        }
    }
    Pri(ans1);
    Pri(ans2);
    return 0;
}
```

8.2 斜率优化 dp

</> 代码 8.2: /动态规划/斜率优化 dp

```
//斜率优化dp
/*方法1.
将dp转移方程化成斜率单调递增(减)的y = kx + b形式
其中b是需要被转移的dp[i],x是只与i有关的单调递增的东西, y是由只与i有关或只与j有关的式子
方法2(推荐).
假设在更新i的时候,对于i存在x > y且x比y更优
即dp[x] + fun(x,i) < dp[y] + fun(y,i) 假设求最小值
化为(dp[x] - dp[y]) / (fun(x) - fun(y)) < fun(i)
则fun(i)为斜率,单调队列维护即可。
*/
/*关于最大(小)值
求最大值:上凸包, 斜率单调递减
求最小值:下凸包, 斜率单调递增
上下凸包在单调队列判断的时候有点区别
*/

/*求dp[i]最小值,斜率单调递增(下凸包)
方法1.
最终状态转移方程 dp[i]+d[i]*sum[j]=dp[j]+pre[j]+d[i]*sum[i-1]-pre[i-1]+c[i]
其中b为dp[i],k为d[i](单调递增),x为sum[j],y为f[j] + pre[j]
其中d[i]*sum[i-1]-pre[i-1]+c[i]和j无关,算dp[i]的时候可以直接算上
方法2.
```

```
设存在x,y对于i使得x > y且x更优
dp[x]+pre[x]-d[i]*sum[x]<dp[y]+pre[y]-d[i]*sum[y]
->(dp[x]+pre[x]-dp[y]-pre[y]) / (sum[x]-sum[y]) < d[i]
*/
//模板是方法1
const int maxn = 1e6 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M;
struct Node{
    LL d,w,C;
}node[maxn];
LL dp[maxn],sum[maxn],pre[maxn];
int Q[maxn];
inline double Y(int i){return dp[i] + pre[i];}
inline double X(int i){return sum[i];}
inline double K(int i,int j){return (Y(j) - Y(i)) / (X(j) - X(i));}
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++ ){
        scanf("%lld%lld%lld",&node[i].d,&node[i].w,&node[i].C);
        sum[i] = sum[i - 1] + node[i].w;
        pre[i] = pre[i - 1] + node[i].w * node[i].d;
    }
    int tail = 0,head = 1;  //单调队列有没有初始数字要看题目决定
    for(int i = 1; i <= N ; i ++){
        dp[i] = node[i].d * sum[i - 1] - pre[i - 1] + node[i].C;
        while(tail > head && K(Q[head],Q[head + 1]) < node[i].d) head++; //这行基本一样
        int j = Q[head]; dp[i] = min(dp[j] + pre[j] + (sum[i - 1] - sum[j]) * node[i].d
    - pre[i - 1] + node[i].C,dp[i]);
        while(tail > head && K(Q[tail - 1],Q[tail]) > K(Q[tail],i)) tail--;  //这行基本一
    样
        Q[++tail] = i;
    }
    Prl(dp[N]);
    return 0;
}

/*依然是下凸包,模板是方法2
dp[i] = dp[j] + w[j + 1] * h[i];
方法2. 存在x > y且x优于y
dp[x] + w[x + 1] * h[i] < dp[y] + w[y + 1] * h[i];
(dp[x] - dp[y]) / (w[y + 1] - w[x + 1]) < h[i]
不等式左边为slope,右边为K
```

```
 */
const int maxn = 5e4 + 10;
const int INF = 0x3f3f3f3f;
int N,M;
int Q[maxn];
struct node{
    LL w,h;
}a[maxn];
LL dp[maxn];
double slope(int i,int j){return (dp[i] - dp[j]) / (a[j + 1].w - a[i + 1].w);}
double K(int i){return a[i].h;}
bool cmp(node a,node b){
    if(a.w == b.w) return a.h > b.h;
    return a.w > b.w;
}
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++) scanf("%lld%lld",&a[i].h,&a[i].w);
    sort(a + 1,a + 1 + N,cmp); int cnt = 1;
    for(int i = 2; i <= N ; i ++){
        if(a[i].h > a[cnt].h) a[++cnt] = a[i];
    }
    N = cnt;
    int tail = 1,head = 1;
    for(int i = 1; i <= N ; i ++){
        while(tail > head && slope(Q[head],Q[head + 1]) <= K(i)) head++;
        int j = Q[head]; dp[i] = dp[j] + a[j + 1].w * a[i].h;
        while(tail > head && slope(Q[tail - 1],Q[tail]) > slope(Q[tail],i)) tail--;
        Q[++tail] = i;
    }
    Prl(dp[N]);
    return 0;
}
/*上凸包(求最大值)
不等式为dp[i] = dp[j] + a * (sum[i] - sum[j])^2 + b * (sum[i] - sum[j]) + c (a < 0)
方法1.
dp[i] + 2 * a * sum[i] * sum[j] = dp[j] + a * sum[i]^2 + a * sum[j]^2 + b * sum[i] - b *
    sum[j] + c
斜率为单调递减的2 * a * sum[i]
方法2.
(dp[x] + a * sum[x]^2 - b * sum[x] - dp[y] - a * sum[y]^2 + b * sum[y]) / (sum[x] - sum[y
    ]) < 2 * a * sum[i]
不等式左边为slope(x,y)
*/
```

```
//方法1模板
const int maxn = 1e6 + 10;
int N,M;
LL sum[maxn],x[maxn],dp[maxn];
LL a,b,c;
int Q[maxn];
inline double X(int i){return sum[i];}
inline double Y(int j){return dp[j] + a * sum[j] * sum[j] - b * sum[j];}
inline double K(int i,int j){return (Y(i) - Y(j)) / (X(i) - X(j));}
int main(){
    Sca(N); scanf("%lld%lld%lld",&a,&b,&c);
    for(int i = 1; i <= N ; i ++){
        Scl(x[i]); sum[i] = sum[i - 1] + x[i];
    }
    int head = 1,tail = 1;
    for(int i = 1; i <= N  ; i++){
        while(tail > head && K(Q[head],Q[head + 1]) >= 2 * a * sum[i]) head++;
        int j = Q[head]; dp[i] = Y(j) - 2 * a * sum[i] * sum[j] + a * sum[i] * sum[i] +
    b * sum[i] + c;
        while(tail > head && K(Q[tail - 1],Q[tail]) < K (Q[tail],i)) tail--;
        Q[++tail] = i;
    }
    Prl(dp[N]);
    return 0;
}
//方法2模板
const int maxn = 1e6 + 10;
int N,M;
LL sum[maxn],x[maxn],dp[maxn];
LL a,b,c;
int Q[maxn];
inline double K(int i){return 2 * a * sum[i];}
inline double slope(int i,int j){return (dp[i] + a * sum[i] * sum[i] - b * sum[i] - dp[j
    ] - a * sum[j] * sum[j] + b * sum[j]) / (sum[i] - sum[j]);}
int main(){
    Sca(N); scanf("%lld%lld%lld",&a,&b,&c);
    for(int i = 1; i <= N ; i ++){
        Scl(x[i]); sum[i] = sum[i - 1] + x[i];
    }
    int head = 1,tail = 1;
    for(int i = 1; i <= N  ; i++){
        while(tail > head && slope(Q[head],Q[head + 1]) > K(i)) head++;
        int j = Q[head]; dp[i] = dp[j] + a * (sum[i] - sum[j]) * (sum[i] - sum[j]) + b *
     (sum[i] - sum[j]) + c;
```

```
149         while(tail > head && slope(Q[tail - 1],Q[tail]) < slope(Q[tail],i)) tail--;
150         Q[++tail] = i;
151     }
152     Prl(dp[N]);
153     return 0;
154 }
```

8.3 数位 dp

</> 代码 8.3: /动态规划/数位 dp

```
1  //数位dp，最普遍的题目：求A到B中满足若干条件的数有多少个
2  //模板：条件是相邻数字的差至少大于2
3  const int maxn = 12;
4  int N,M,K,cnt;
5  int str[maxn];
6  int dp[maxn][maxn];   //i位置的j数
7  int dfs(int pos,int num,int zero,int limit){
8      if(pos == 0) return (zero ^ 1);
9      if(~dp[pos][num] && !zero && !limit) return dp[pos][num];
10     int top = limit?str[pos - 1]:9;
11     int ans = 0;
12     for(int i = 0 ; i <= top; i ++){
13         if(abs(i - num) <= 1 && !zero) continue;
14         ans += dfs(pos - 1,i,zero && !i,limit && str[pos - 1] == i);
15     }
16     if(!zero && !limit) dp[pos][num] = ans;
17     return ans;
18 }
19 int solve(int x){
20     if(!x) return 0;
21     Mem(dp,-1);
22     cnt = 0;
23     while(x){
24         str[cnt++] = x % 10;
25         x /= 10;
26     }
27     str[cnt] = 0;
28     int ans = dfs(cnt,0,1,1);
29     return ans;
30 }
31 int main(){
32     int A = read(), B = read();
33     Pri(solve(B) - solve(A - 1));
```

```
34      return 0;
35  }
```

8.4 错排公式

</> 代码 8.4: /动态规划/错排公式

```
1  //n封信对应n个信封,求恰好全部装错了信封的方案数
2  //dp[i] = (i-1) * (dp[i-1] + dp[i-2])
3  int N,M,K;
4  LL dp[30];
5  int main(){
6      dp[1] = 0,dp[2] = 1;
7      for(int i = 3; i <= 20; i ++) dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]);
8      while(~Sca(N)) Prl(dp[N]);
9      return 0;
10 }
```

第九章　其他

9.1　STL

9.1.1　multiset

</> 代码 9.1: /其他/multiset

```
1  //mulitiset
2  mulitiset<int>c;
3  c.size()     //返回当前的元素数量
4  c.empty ()   //判断大小是否为零，等同于0 == size()，效率更高
5  count (elem) //返回元素值为elem的个数
6  find(elem)   //返回元素值为elem的第一个元素，如果没有返回end()
7  lower _bound(elem)  //返回元素值为elem的第一个可安插位置，也就是元素值 >= elem的第一个元素位置
8  upper _bound (elem) //返回元素值为elem的最后一个可安插位置，也就是元素值 > elem 的第一个元素位置
9  c.begin()  //返回一个随机存取迭代器，指向第一个元素
10 c.end()    //返回一个随机存取迭代器，指向最后一个元素的下一个位置
11 c.rbegin() //返回一个逆向迭代器，指向逆向迭代的第一个元素
12 c.rend()   //返回一个逆向迭代器，指向逆向迭代的最后一个元素的下一个位置
13 c.insert(elem) //插入一个elem副本，返回新元素位置，无论插入成功与否。
14 c.insert(pos, elem) //安插一个elem元素副本，返回新元素位置，pos为收索起点，提升插入速度。
15 c.insert(beg,end) //将区间[beg,end)所有的元素安插到c，无返回值。
16 c.erase(elem) //删除与elem相等的所有元素，返回被移除的元素个数。
17 c.erase(pos) //移除迭代器pos所指位置元素，无返回值。
18 c.erase(beg,end) //移除区间[beg,end)所有元素，无返回值。
19 c.clear() //移除所有元素，将容器清空
20 //注意事项:mulitiset是在set的基础上允许重复元素，如果需要删除某个值x的其中一个元素，不能通过c.erase(3)，这样会删除所有值为3的元素
21 //而是it = c.find(3); c.erase(it);
```

9.1.2　bitset

</> 代码 9.2: /其他/bitset

```
1  //bitset<maxn>a
2  /*
```

```
bitset是一种专门用来储存二进制的数组，使用前要先调用函数库。
他的每一个元素只占 1 bit空间，你可以将它当作bool类型的高精度。
他的优点很多，你可将他整体使用，也可单个访问
bitset的原理大概是将很多数压成一个，从而节省空间和时间（暴力出奇迹）
一般来说bitset会让你的算法复杂度 /32
 */
//构造函数
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

bitset<2> bitset1(12);  //12的二进制为1100（长度为 4），但bitset1的size=2，只取后面部分，
    即00
string s = "100101";
bitset<4> bitset2(s);  //s的size=6，而bitset的size=4，只取前面部分，即1001
char s2[] = "11101";
bitset<4> bitset3(s2);  //与bitset2同理，只取前面部分，即1110
cout << bitset1 << endl;  //00
cout << bitset2 << endl;  //1001
cout << bitset3 << endl;  //1110

//可用操作符
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)
```

```
46
47 //下标访问
48 bitset<4> foo ("1011");
49 cout << foo[0] << endl;  //1
50 cout << foo[1] << endl;  //1
51 cout << foo[2] << endl;  //0
52
53 //支持的函数
54 bitset<8> foo ("10011011");
55 cout << foo.count() << endl;  //5     (count函数用来求bitset中1的位数，foo中共有5个1
56 cout << foo.size() << endl;   //8     (size函数用来求bitset的大小，一共有8位
57 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
58 cout << foo.test(2) << endl;  //false     (同理，foo[2]为0，返回false
59 cout << foo.any() << endl;    //true     (any函数检查bitset中是否有1
60 cout << foo.none() << endl;   //false     (none函数检查bitset中是否没有1
61 cout << foo.all() << endl;    //false     (all函数检查bitset中是全部为1
62
63 //操作函数，相比于运用下标，这些会检查有没有越界
64 bitset<8> foo ("10011011");
65 cout << foo.flip(2) << endl;  //10011111     (flip函数传参数时，用于将参数位取反，本行代码
       将foo下标2处"反转"，即0变1，1变0
66 cout << foo.flip() << endl;   //01100000     (flip函数不指定参数时，将bitset每一位全部取反
67 cout << foo.set() << endl;        //11111111     (set函数不指定参数时，将bitset的每一位全
       部置为1
68 cout << foo.set(3,0) << endl;  //11110111     (set函数指定两位参数时，将第一参数位的元素置
       为第二参数的值，本行对foo的操作相当于foo[3]=0
69 cout << foo.set(3) << endl;    //11111111     (set函数只有一个参数时，将参数下标处置为1
70 cout << foo.reset(4) << endl;  //11101111     (reset函数传一个参数时将参数下标处置为0
71 cout << foo.reset() << endl;   //00000000     (reset函数不传参数时将bitset的每一位全部置为
       0
72
73 //转换类型
74 bitset<8> foo ("10011011");
75 string s = foo.to_string();  //将bitset转换成string类型
76 unsigned long a = foo.to_ulong();  //将bitset转换成unsigned long类型
77 unsigned long long b = foo.to_ullong();  //将bitset转换成unsigned long long类型
78 cout << s << endl;  //10011011
79 cout << a << endl;  //155
80 cout << b << endl;  //155
```

9.2 表达式树

</> 代码 9.3: /其他/表达式树

```cpp
#include <bits/stdc++.h>
using namespace std;
//表达式树
typedef pair<int, int>PII;
typedef long long ll;
const int MX = 3e5 + 5;

char S[MX];
int LT[MX], RT[MX], rear, n;
ll A[MX];
string op[MX];
string last[10];

int build(int L, int R) {
    int cur = L, u;
    ll tmp = 0;
    while(cur <= R && isdigit(S[cur])) tmp = tmp * 10 + S[cur] - '0', cur++;
    if(cur == R + 1) {
        u = rear++;
        op[u] = ".";
        A[u] = tmp;
        return u;
    }

    int p[10], cnt = 0;
    for(int i = 1; i <= 9; i++) p[i] = -1;
    while(cur <= R) {
        string oper = "";
        if(!isdigit(S[cur])) {
            oper += S[cur];
            if(cur + 1 <= R) {
                if(S[cur] == '>' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '<' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '=' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '!' && S[cur + 1] == '=') oper += S[++cur];
                if(S[cur] == '&' && S[cur + 1] == '&') oper += S[++cur];
                if(S[cur] == '|' && S[cur + 1] == '|') oper += S[++cur];
            }
        }

        if(oper == "(") cnt++;
        else if(oper == ")") cnt--;
        else if(cnt);
```

```
44          else if(oper == "*" || oper == "/" || oper == "%") last[1] = oper, p[1] = cur;
45          else if(oper == "+" || oper == "-") last[2] = oper, p[2] = cur;
46          else if(oper == ">" || oper == ">=" || oper == "<" || oper == "<=") last[3] =
        oper, p[3] = cur;
47          else if(oper == "==" || oper == "!=") last[4] = oper, p[4] = cur;
48          else if(oper == "&") last[5] = oper, p[5] = cur;
49          else if(oper == "^") last[6] = oper, p[6] = cur;
50          else if(oper == "|") last[7] = oper, p[7] = cur;
51          else if(oper == "&&") last[8] = oper, p[8] = cur;
52          else if(oper == "||") last[9] = oper, p[9] = cur;
53
54          cur++;
55      }
56      int w = -1, wi;
57      for(int i = 9; i >= 1; i--) {
58          if(p[i] != -1) {
59              w = p[i];
60              wi = i;
61              break;
62          }
63      }
64      if(w == -1) u = build(L + 1, R - 1);
65      else {
66          u = rear++;
67          op[u] = last[wi];
68          LT[u] = build(L, w - op[u].length());
69          RT[u] = build(w + 1, R);
70      }
71      return u;
72  }
73
74  ll solve(int u) {
75      if(op[u] == ".") return A[u];
76      ll l = solve(LT[u]), r = solve(RT[u]);
77      if(op[u] == "*") return l * r;
78      if(op[u] == "/") return l / r;
79      if(op[u] == "%") return l % r;
80      if(op[u] == "+") return l + r;
81      if(op[u] == "-") return l - r;
82      if(op[u] == ">") return l > r;
83      if(op[u] == ">=") return l >= r;
84      if(op[u] == "<") return l < r;
85      if(op[u] == "<=") return l <= r;
86      if(op[u] == "==") return l == r;
```

```
87      if(op[u] == "!=") return l != r;
88      if(op[u] == "&") return l & r;
89      if(op[u] == "^") return l ^ r;
90      if(op[u] == "|") return l | r;
91      if(op[u] == "&&") return l && r;
92      if(op[u] == "||") return l || r;
93      return 0;
94  }
95
96  int main() {
97      //FIN;
98      while(~scanf("%s", S)) {
99          n = strlen(S);
100         if(S[0] == '0' && n == 1) break;
101
102         rear = 0;
103         build(0, n - 1);
104         printf("%lld\n", solve(0));
105     }
106     return 0;
107 }
```

9.3 切比雪夫距离

</> 代码 9.4: /其他/切比雪夫距离

```
1  /* 曼哈顿距离和切比雪夫距离
2  曼哈顿距离:两点横纵坐标差之和 即dis = |x1 - x2| + |y1 - y2|;
3  切比雪夫距离: 两点横纵坐标差的最大值 即dis = max(|x1 - x2|,|y1 -y2|);
4  两距离可以相互转化
5  将一个点(x,y)的坐标变为(x+y,x-y)后,原坐标系中的曼哈顿距离 = 新坐标系中的切比雪夫距离
6  将一个点(x,y)的坐标变为((x+y)/2,(x-y)/2) 后,原坐标系中的切比雪夫距离 = 新坐标系中的曼哈顿距
       离
7  用处:
8  1.切比雪夫距离转曼哈顿距离
9  切比雪夫距离在计算的时候需要取max，往往不是很好优化，对于一个点，计算其他点到该的距离的复杂度
       为O(n)
10 而曼哈顿距离只有求和以及取绝对值两种运算，我们把坐标排序后可以去掉绝对值的影响，进而用前缀和
       优化，可以把复杂度降为O(1)
11 2.曼哈顿距离转切比雪夫距离
12 例如维护区间两点之间最大的曼哈顿距离，可以转为切比雪夫距离，分别维护区间x,y的最大(小)值，求
       他们之间的差再取x,y中较大的值就是原坐标中的最大的曼哈顿距离
13  */
14 //模板: 给一堆点，求一个点使得他到其他所有点的切比雪夫距离最小
```

```cpp
//做法:转曼哈顿距离之后排序维护前缀后缀x,y的和
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
const int mod = 1e9 + 7;
int N,M,K;
struct Node{
    ll x,y;
    int id;
}node[maxn];
bool cmp(Node a,Node b){return a.x < b.x;}
bool cmp2(Node a,Node b){return a.y < b.y;}
ll ans[maxn];
int main(){
    Sca(N);
    for(int i = 1; i <= N ; i ++){
        int x,y; Sca2(x,y);
        node[i].x = x + y; node[i].y = x - y;
        node[i].id = i;
    }
    sort(node + 1,node + 1 + N,cmp);
    ll pre = 0;
    for(int i = 1; i <= N ; i ++){
        ans[node[i].id] += (i - 1) * node[i].x - pre;
        pre += node[i].x;
    }
    pre = 0;
    for(int i = N ; i >= 1; i --){
        ans[node[i].id] += pre - (N - i) * node[i].x;
        pre += node[i].x;
    }
    sort(node + 1,node + 1 + N,cmp2);
    pre = 0;
    for(int i = 1; i <= N ; i ++){
        ans[node[i].id] += (i - 1) * node[i].y - pre;
        pre += node[i].y;
    }
    pre = 0;
    for(int i = N ; i >= 1; i --){
        ans[node[i].id] += pre - (N - i) * node[i].y;
        pre += node[i].y;
    }
    ll Min = 1e18;
    for(int i = 1; i <= N ; i ++) Min = min(Min,ans[i]);
    Min /= 2;
```

```
    Prl(Min);
    return 0;
}
```

9.4 离散化

</> 代码 9.5: /其他/离散化

```
//离散化，模板：离散N条线段
int Hash[maxn * 2];
int cnt = 0;
for(int i = 1; i <= N; i ++) {
    line[i].fi = read(); line[i].se = read();
    Hash[++cnt] = line[i].fi; Hash[++cnt] = line[i].se;
}
sort(Hash + 1,Hash + 1 + cnt);
cnt = unique(Hash + 1,Hash + 1 + cnt) - Hash - 1;
for(int i = 1; i <= N ; i ++) {
    line[i].fi = lower_bound(Hash + 1,Hash + 1 + cnt,line[i].fi) - Hash;
    line[i].se = lower_bound(Hash + 1,Hash + 1 + cnt,line[i].se) - Hash;
}
```

9.5 区间离散化

</> 代码 9.6: /其他/区间离散化

```
//区间离散化
/*
    涉及到区间修改的离散化是和普通离散化不同的
    假设修改20 - 30,查询10 - 40
    常规的离散化变成1，2，3，4之后线段树修改2,3两个点，询问1-4的sum
    但是修改的时候会分开修改2和3，这和我们想要的不一样
    因为离散化区间是不能直接相加的，[2,2] + [3,3] = [2,3]是没毛病，[20,20] + [30,30] =
    [20,30]就有问题了
    所以我们希望他的区间可以直接相加
    寻常的线段树是左右都闭，我们需要把它转化为左闭右开的区间
    让他变为[2,3) + [3,4) = [2,4)  => [20,30) + [30,31) = [20,31)
    所以说，我们离散化的时候需要离散的是l和r + 1而不是寻常的l,r
 */
ll read(){ll x = 0,f = 1;char c = getchar();while (c<'0' || c>'9'){if (c == '-') f = -1;
    c = getchar();}
while (c >= '0'&&c <= '9'){x = x * 10 + c - '0';c = getchar();}return x*f;}
const double eps = 1e-9;
const int maxn = 8e5 + 10;
```

```
17 ll N;
18 int M;
19 struct Query{
20     int op;
21     ll a,b,c;
22 }query[maxn];
23 ll Hash[maxn * 2];
24 struct Tree{
25     int l,r;
26     ull sum,lazy;
27 }tree[maxn * 4];
28 void Build(int t,int l,int r){
29     tree[t].l = l; tree[t].r = r;
30     tree[t].sum = tree[t].lazy = 0;
31     if(r - l <= 1) return;
32     int m = l + r >> 1;
33     Build(t << 1,l,m);
34     Build(t << 1 | 1,m,r);
35 }
36 void Pushup(int t){
37     tree[t].sum = tree[t << 1].sum + tree[t << 1 | 1].sum;
38 }
39 void solve(int t,ull p){
40     tree[t].lazy = p + tree[t].lazy;
41     tree[t].sum = tree[t].sum + ((ull)Hash[tree[t].r] - (ull)Hash[tree[t].l]) * p;
42 }
43 void Pushdown(int t){
44     if(tree[t].lazy){
45         solve(t << 1,tree[t].lazy);
46         solve(t << 1 | 1,tree[t].lazy);
47         tree[t].lazy = 0;
48     }
49 }
50 void update(int t,int l,int r,ull p){
51     if(l <= tree[t].l && tree[t].r <= r){
52         solve(t,p);
53         return;
54     }
55     Pushdown(t);
56     int m = (tree[t].l + tree[t].r) >> 1;
57     if(r <= m) update(t << 1,l,r,p);
58     else if(l >= m) update(t << 1 | 1,l,r,p);
59     else{
60         update(t << 1,l,m,p);
```

```
61          update(t << 1 | 1,m,r,p);
62      }
63      Pushup(t);
64  }
65  ull _query(int t,int l,int r){
66      if(l <= tree[t].l && tree[t].r <= r){
67          return tree[t].sum;
68      }
69      Pushdown(t);
70      int m = (tree[t].l + tree[t].r) >> 1;
71      if(r <= m) return _query(t << 1,l,r);
72      else if(l >= m) return _query(t << 1 | 1,l,r);
73      else{
74          return _query(t << 1,l,m) + _query(t << 1 | 1,m,r);
75      }
76  }
77  int main(){
78      cin >> N >> M;
79      int cnt = 0;
80      for(int i = 1; i <= M; i ++){
81          query[i].op = read();
82          query[i].a = read();
83          query[i].b = read();
84          if(query[i].op == 1) query[i].c = read();
85          Hash[++cnt] = query[i].a;
86         // Hash[++cnt] = query[i].b;
87          Hash[++cnt] = query[i].b + 1;
88      }
89      sort(Hash + 1,Hash + 1 + cnt);
90      cnt = unique(Hash + 1,Hash + 1 + cnt) - Hash - 1;
91      Build(1,1,cnt);
92      for(int i = 1; i <= M; i ++){
93          query[i].a = lower_bound(Hash + 1,Hash + 1 + cnt,query[i].a) - Hash;
94          query[i].b = lower_bound(Hash + 1,Hash + 1 + cnt,query[i].b + 1) - Hash;
95          if(query[i].op == 1) update(1,query[i].a,query[i].b,query[i].c);
96          else{
97              ull ans = _query(1,query[i].a,query[i].b);
98              cout << ans << endl;
99          }
100     }
101     return 0;
102 }
```

9.6 注意事项

!!!!!!!!!!

1. 注意精度问题, 尤其是过程中的精度问题，例如 int * int 的值大于 LL，即使赋值给 LL 也会炸例如 (1 « 50) 是有问题的，需要 (1LL « 50)。

2. 在交题前一定要重新审视一边整个代码，不要心急交题, 记得要测两个小样例和极端样例。

3. 初始化请务必 0 N+1,memset 容易被卡, 手动初始化容易遗漏边界 0 等问题, 以及 init 记得在主函数里调用，不要调用在 N 的输入之前。

4. 读入挂可以尝试但不要迷信, 包括 inline 和 register 这样的玄学优化，更加多的去寻找算法本身的问题。

5. 在遇到莫名其妙不知所云的错误的时候（例如一个值好好的莫名其妙就在中途改变了），很有可能是数组溢出之类的错误。

6. 如果在想 dp 状态转移方程的时候始终觉得状态太多存不下，找规律又不怎么找得到的时候，要记得有一那么一类题，当你枚举出起始或者初始的有限个状态的时候，可以递推出后面所有的结果，所以只要枚举 + 验证即可，例如知道了石头剪刀布的结果（是谁赢）就可以推出过程的两个状态。

7. 如果 DP 完了之后要求记忆路径的操作很麻烦或者很容易出错，可以考虑在 DP 的过程中就记录当前状态的前驱。

8. 如果为了压行去大括号，请务必看清楚大括号里有几句话!!!

9. 如果感觉被卡常了且函数里含有形如 a.a/a.b<b.a/b.b 之类的 cmp 函数，不妨将结构体的答案 a/b 直接计算出来比较，每次比较的时候都要计算一下会很费时间。

10. 结构体排序建议把所有成员的关键字都设置好使得这个排序是稳定的，否则可能会造成奇怪的问题 (至于为什么，暂时还不知道)。

11. 涉及到区间修改的离散化请查阅上面的离散化板子，需要左闭右开的离散化而不能跟个愣头青一样离散化。