Algorithm

chenRenning

算法模板整理

更新：2024 年 8 月 4 日

目录

代码基本框架:

```cpp
#include <bits/stdc++.h>

using namespace std ;

void solve()
{
  // do something
}

int main()
{
  cin.tie(nullptr)->ios::sync_wtih_stdio(false) ;
  //关闭同步流-> 加快cin

  int t = 1 ;
  // cin >> t ; 多测的时候将注释去掉
  while ( t-- ) {
    solve() ;
  }
  return 0 ;
}
```

1 基础算法

1.1 排序

常用排序算法

- quick_sort $O(nlogn)$
- merge_sort $O(nlogn)$

```cpp
// quick_sort
const int N = 1e5 + 10 ;

int a[N] , n ;

void quick_sort(int l,int r) {
  if ( l >= r ) return ;
  int mid = (l + r) >> 1 ;
  int k = a[mid] , i = l - 1 , j = r + 1 ;
  while ( j > i ) {
    do ++ i ; while ( a[i] < k ) ;
    do -- j ; while ( a[j] > k ) ;
    if ( j > i ) {
      std::swap(a[i] , a[j]) ;
    }
  }
  quick_sort(l , j) , quick_sort(j + 1 , r) ;
}
```

```cpp
// merge_sort
const int N = 1e5 + 10 ;

int a[N] , tmp[N] , n ;

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 ( a[i] < a[j] ) {
      tmp[k++] = a[i++] ;
    } else {
      tmp[k++] = a[j++] ;
    }
  }
```

```
  while (i <= mid) tmp[k++] = a[i++] ;
  while (j <= r) tmp[k++] = a[j++] ;
  for(i = 0 , j = l ; j <= r ; ++ j , ++ i) a[j] = tmp[i] ;
}
```

1.2 二分 binary search

常用函数:

lower_bound(left,right,value)

upper_bound(left,right,value)

.....(以及 stl 容器自带方法等)

建立在有序序列之中 具体用法 https://oi-wiki.org/basic/binary/

```
// 边界值根据所需改变
// 整数二分
int l = 0 , r = 1e12 , ans = 0 ;
while ( r >= l ) {
  int mid = (l + r) >> 1 ; // 等同于 (l + r) / 2
  if ( check(mid) ) {
    ans = mid ;
    // r = mid - 1 or l = mid + 1 根据需求来
  } else {
    // l = mid + 1 or r = mid - 1 同理
  }
}
// ans 即所需 关键在check函数的设计
// 实数二分
const double eps = 1e-8 ; // 精度
double l = 0 , r = 1e6 , ans = 0 ;
while ( r - l >= eps )
{
  double mid = (l + r) / 2 ;
  if (check(mid)) {
    ans = mid ;
    // r = mid or l = mid
  } else {
    // l = mid or r = mid
```

```
  }
}
```

1.3 常见高精度

```
//高精度 + 高精度 O(n + m)
std::vector<int> add(std::vector<int> A,std::vector<int> B)
{
    std::vector<int> C ;
    int t = 0 ;
    for(int i = 0 , j = 0 ; i < (int)A.size() || j < (int)B.
       size() ; ++ i , ++ j)
    {
        if ( i < (int)A.size() ) t += A[i] ;
        if ( i < (int)B.size() ) t += B[i] ;
        C.push_back(t % 10) ;
        t /= 10 ;
    }
    while ( t ) C.push_back(t % 10) , t /= 10 ;
    while ( C.size() > 1 && C.back() == 0 ) C.pop_back() ;
    return C ;
}
```

```
// 高精度 - 高精度 O(n + m)
#include <bits/stdc++.h>

bool cmp(std::string a,std::string b) {
    if ( a.size() != b.size() ) return a.size() < b.size() ;
    for(int i = 0 ; i < (int)a.size() ; ++ i) if ( a[i] != b[i
       ] ) return a[i] < b[i] ;
    return false ;
}
std::vector<int> sub(std::vector<int> A,std::vector<int> B) {
    std::vector<int> C ;
    int t = 0 ;
    for(int i = 0 ; i < (int)A.size() ; ++ i)
    {
        t += A[i] ;
```

```
        if (i < (int)B.size()) t -= B[i] ;
        C.push_back((t % 10 + 10) %10) ;
        if ( t < 0 ) t = -1 ;
        else t = 0 ;
    }
    while (t) C.push_back((t % 10 + 10) % 10) , t /= 10;
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}

int main()
{
  //字符串直接比较是按照字典序，因此需要手写一个cmp函数比较二者数值大小
    std::string a , b ;
    std::cin >> a >> b ;
    if ( cmp(a , b) ) {
        std::cout << '-' ;
        swap(a , b) ;
    }
    std::vector<int> A , B ;
    for(int i = (int)a.size() - 1 ; i >= 0 ; -- i) A.push_back(a[i] - '0') ;
    for(int i = (int)b.size() - 1 ; i >= 0 ; -- i) B.push_back(b[i] - '0') ;
    auto C = sub(A , B) ;

    for(int i = (int)C.size() - 1 ; i >= 0 ; -- i) std::cout << C[i] ;

    return 0 ;
}
```

```
// 高精度 * 低精度
std::vector<int> mul(std::vector<int> A,int B) {
  std::vector<int> C ;
  int t = 0 ;
  for(int i = 0 ; i < (int)A.size() ; ++ i)
  {
```

```
        t += A[i] * B ;
        C.push_back(t % 10) ;
        t /= 10;
    }
    while ( t ) C.push_back(t % 10) , t /= 10  ;
    while ( C.size() > 1 && C.back() == 0 ) C.pop_back() ;
    return C ;
}
```

```
// 高精度 ÷ 低精度
std::vector<int> div(std::vector<int> A,int B,int &r)
{
    std::vector<int> C ;
    r = 0 ;
    for(int i = (int)A.size() - 1 ; i >= 0 ; -- i)
    {
        r = r * 10 + A[i] ;
        C.push_back(r / B) ;
        r %= B ;
    }
    reverse(C.begin(),C.end()) ;
    while ( C.size() > 1 && C.back() == 0 ) C.pop_back();
    return C ;
}
```

1.4 前缀和

```
// 一维
int n , a[N] , sum[N] ;
void solve() {
    std::cin >> n ;
    for(int i = 1 ; i <= n ; ++ i)
    {
        std::cin >> a[i] ;
        sum[i] = sum[i - 1] + a[i] ;
    }
}
// sum_i 表示前 i 个元素之和
```

```cpp
// 二维
int n , a[N][N] , sum[N][N] ;
void solve() {
  std::cin >> n ;
  for(int i = 1 ; i <= n ; ++ i)
  {
    for(int j = 1 ; j <= n ; ++ j)
    {
      std::cin >> a[i][j] ;
      sum[i][j] = sum[i - 1][j] + sum[i][j - 1] - sum[i - 1][j
          - 1] + a[i][j] ;
    }
  }
}
// sum_{i ,j} 表示以(i , j) 为右下端点的矩阵元素之和
```

1.5 差分

```cpp
// 一维
int n , q , a[N] , b[N] ;
void solve() {
  std::cin >> n >> q ;
  for(int i = 1 ; i <= n ; ++ i)
  {
    std::cin >> a[i] ;
  }
  while (q --)
  {
    // 每一次操作给定 l , r , v 表示在数组 a 区间[l , r] 上的
       元素加上 v
    int l , r , v ;
    std::cin >> l >> r >> v ;
    b[l] += v ;
    b[r + 1] -= v ;
  }
  for(int i = 1 ; i <= n ; ++ i)
  {
```

```
    b[i] += b[i - 1] ;
    a[i] += b[i] ;
  }
}
```

```
// 二维
#include <bits/stdc++.h>

const int N = 1010 ;

int n , m , q , a[N][N] , b[N][N] ;

void insert(int x1,int y1,int x2,int y2,int v)
{
    b[x1][y1] += v;
    b[x2 + 1][y1] -= v ;
    b[x1][y2 + 1] -= v ;
    b[x2 + 1][y2 + 1] += v ;
}

int main()
{
    std::cin >> n >> m >> q ;
    for(int i = 1 ; i <= n ; ++ i)
    {
        for(int j = 1 ; j <= m ; ++ j)
        {
            std::cin >> a[i][j] ;
        }
    }
    while ( q-- )
    {
        int x1 , y1 , x2 , y2 , v ;
        std::cin >> x1 >> y1 >> x2 >> y2 >> v ;
        insert(x1 , y1 , x2 , y2 , v) ;
    }
    for(int i = 1 ; i <= n ; ++ i)
    {
        for(int j = 1 ; j <= m ; ++ j)
```

```
35          {
36              b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j
                    - 1] ;
37              std::cout << b[i][j] + a[i][j] << ' ' ;
38          }
39          std::cout << '\n' ;
40      }
41      return 0 ;
42  }
```

1.6 双指针

```
1   // 滑动窗口（同向双指针）
2   // 给定一个长度为 n 的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。
3   #include <bits/stdc++.h>
4
5   int main()
6   {
7       int n ;
8       std::cin >> n ;
9       std::vector<int> a(n) , cnt(100010 , 0) ;
10      for(auto & p : a) std::cin >> p ;
11      int l = 0 , r = 0 , ans = 0 ;
12      while ( r < n )
13      {
14          ++ cnt[a[r]] ;
15          while (r > l && cnt[a[r]] > 1) {
16              -- cnt[a[l++]] ;
17          }
18          ans = std::max(ans , r - l + 1) ;
19          ++ r ;
20      }
21      std::cout << ans << "\n" ;
22      return 0 ;
23  }
```

```
1   // 双向双指针
```

```cpp
// 例题 lc1.两数之和
vector<int> twoSum(vector<int>& nums, int target) {
  vector<pair<int , int>> a;
  for(int i = 0 ; i < (int)nums.size() ; ++ i) a.push_back({
     nums[i] , i}) ;
  sort(a.begin(),a.end()) ;
  int l = 0 , r = (int)a.size() - 1 ;

  while (r > l)
  {
      if ( a[l].first + a[r].first == target ) {
          return {a[l].second , a[r].second} ;
      } else if ( a[l].first + a[r].first > target ) {
          --r ;
      } else {
          ++l ;
      }
  }

  return {} ;
}
```

1.7 离散化

```cpp
//例题acwing802
#include <bits/stdc++.h>

int main()
{
    std::cin.tie(nullptr)->std::ios::sync_with_stdio(false) ;

    int n , m ;
    std::cin >> n >> m ;

    std::vector<std::pair<int , int>> a;
    std::vector<int> bin , b , sum ;
    for(int i = 1 ; i <= n ; ++ i)
```

```
14      {
15          int x , v ;
16          std::cin >> x >> v ;
17          a.push_back({x , v}) ;
18          bin.push_back(x) ;
19      }
20      sort(a.begin(),a.end()) ;
21      sort(bin.begin(),bin.end()) ;
22      bin.erase(unique(bin.begin(),bin.end()),bin.end()) ;
23      // unique 去重函数返回去重后的尾地址， erase(l , r) 删除下
            标
24      b.resize(bin.size() , 0) ;
25      sum.resize(bin.size() + 1 , 0) ;
26      for(auto p : a)
27      {
28          int x = lower_bound(bin.begin(),bin.end(),p.first) -
                bin.begin() ;
29          b[x] += p.second ;
30      }
31
32      for(int i = 0 ; i < (int)b.size() ; ++ i)
33      {
34          sum[i + 1] = sum[i] + b[i] ;
35      }
36      while (m--)
37      {
38          int l , r ;
39          std::cin >> l >> r ;
40          int L = upper_bound(bin.begin(),bin.end(),l) - bin.
                begin() - 1 ;
41          int R = upper_bound(bin.begin(),bin.end(),r) - bin.
                begin() - 1 ;
42          if ( bin[L] < l ) ++ L ;
43          if ( L > R )
44          {
45              std::cout << 0 << "\n" ;
46              continue ;
47          }
```

```
48          std::cout << sum[R + 1] - sum[L] << '\n' ;
49      }
50      return 0 ;
51  }
```

1.8 区间合并

```
1  //例题acwing803
2  std::vector<std::pair<int , int>> merge(std::vector<std::pair<
       int , int>> a)
3  {
4      std::vector<std::pair<int , int>> C ;
5      int l , r ;
6      l = r = -2e9 ;
7      for(auto p : a)
8      {
9          if ( p.first > r )
10         {
11             if ( r != -2e9 ) {
12                 C.push_back({l , r}) ;
13             }
14             l = p.first , r = p.second ;
15         } else {
16             r = std::max(p.second , r) ;
17         }
18     }
19     C.push_back({l , r}) ;
20     return C ;
21 }
```

1.9 应用

求逆序对的个数 (归并、线段树)

```
1  // O(nlogn) 归并排序过程中求得逆序对数量
2  i64 merge_sort(int l, int r) {
3  if ( l == r ) return 0 ;
```

```
int mid = (l + r) >> 1 ;
i64 x = merge_sort(l , mid) , y = merge_sort(mid + 1 , r) ,
   ans = 0 ;

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

```
// O(nlogn) 利用线段树(单点修改、区间查询)求得逆序对数量
#define ls (x << 1)
#define rs (x << 1 | 1)

int n , a[N] , sum[N << 2] , bin[N] ;
inline void update(int x) { sum[x] = sum[ls] + sum[rs] ; }
inline void modify(int pos,int l,int r, int x) {
    if ( l == r ) return sum[x] ++ , void() ;
    int mid = (l + r) >> 1 ;
    if ( pos <= mid ) modify(pos , l , mid , ls) ;
    else modify(pos , mid + 1 , r , rs) ;
    update(x) ;
}
inline int query(int A,int B,int l,int r,int x) {
    if ( A > B ) return 0 ;
    if ( A <= l && r <= B ) return sum[x] ;
    int mid = (l + r) >> 1 , ans = 0 ;
    if (A <= mid) ans += query(A , B , l , mid , ls) ;
    if ( mid < B ) ans += query(A , B , mid + 1 , r , rs) ;
    return ans ;
}
```

```
22 void solve() {
23   std::cin >> n ;
24   for(int i = 1 ; i <= n ; ++ i)
25   {
26     std::cin >> a[i] ;
27     bin[i] = a[i] ;
28   }
29   std::sort(bin + 1 , bin + 1 + n) ;
30   int m = std::unique(bin + 1 , bin + 1 + n) - bin - 1 ;
31   i64 ans = 0 ;
32   for(int i = 1 ; i <= n ; ++ i)
33   {
34       int x = std::lower_bound(bin + 1 , bin + 1 + m , a[i]) -
              bin ;
35       ans += query(x + 1 , m , 1 , n , 1) ;
36       modify(x , 1 , n , 1) ;
37   }
38   std::cout << ans << '\n' ;
39 }
```

2 数据结构

2.1 链表

```
//数组模拟单链表
//head需要初始化为 -1 表示末尾
int n , cnt , head , ne[N] , e[N] ;
inline void insert(int k,int x) {
  // 在第 k 个插入的元素后面插入一个元素 x
  e[cnt] = x , ne[cnt] = ne[k] , ne[k] = cnt++ ;
}
inline void del(int k) {
  // 删除第 k 个插入的元素后面的元素
  ne[k] = ne[ne[k]] ;
}
inline void head_insert(int x) {
  // 头部插入一个元素 x
  e[cnt] = x , ne[cnt] = head , head = cnt++ ;
}
inline void print() {
  // 输出整个链表
  for(int i = head ; i != -1 ; i = ne[i])
  {
    std::cout << e[i] << " " ;
  }
}
```

```
// 数组模拟双链表 例题acwing827
#include <bits/stdc++.h>

using namespace std ;

const int N = 2e5 + 10 ;

int n , lft[N] , rgt[N] , cnt , e[N] ;
void insert(int k, int x) {
    e[cnt] = x ;
    rgt[cnt] = rgt[k] ;
```

```
12      lft[cnt] = k ;
13      lft[rgt[cnt]] = cnt ;
14      rgt[k] = cnt++ ;
15  }
16  void del(int k) {
17      lft[rgt[k]] = lft[k] ;
18      rgt[lft[k]] = rgt[k] ;
19  }
20  int main()
21  {
22      cin >> n ;
23      cnt = 2 ;
24      lft[1] = 0 , rgt[0] = 1 ;
25      for(int i = 1 ; i <= n ; ++ i)
26      {
27          string op ;
28          cin >> op ;
29          if (op == "L")
30          {
31              int x ;
32              cin >> x ;
33              insert(0 , x) ;
34          } else if (op == "R") {
35              int x ;
36              cin >> x ;
37              insert(lft[1] , x) ;
38          } else if ( op == "D") {
39              int k ;
40              cin >> k ;
41              del(k + 1) ;
42          } else if ( op == "IL" ) {
43              int k , x ;
44              cin >> k >> x ;
45              insert(lft[k + 1] , x) ;
46          } else if ( op == "IR" ) {
47              int k , x ;
48              cin >> k >> x ;
49              insert(k + 1 , x) ;
```

```
        }
    }
    for(int i = rgt[0] ; i != 1 ; i = rgt[i])
    {
        cout << e[i] << " " ;
    }
    return 0 ;
}
```

2.2 栈

```
//经典例题 : 表达式求值
#include <bits/stdc++.h>

using namespace std ;

stack<char> op ;
stack<int> num ;

void eval()
{
    auto a = num.top() ; num.pop() ;
    auto b = num.top() ; num.pop() ;
    auto c = op.top() ; op.pop() ;
    if ( c == '*' ) {
        num.push(a * b) ;
    } else if ( c == '/' ) {
        num.push(b / a) ;
    } else if ( c == '+' ) {
        num.push(a + b) ;
    } else {
        num.push(b - a) ;
    }
}

int main()
{
```

```
    unordered_map<char , int> pr {{'+',1},{'-',1},{'*',2},{'/',2}} ;
    string str ;
    cin >> str ;

    for(int i = 0 ; i < (int)str.size() ; ++ i)
    {
        if ( isdigit(str[i]) ) {
            int j = str[i] - '0' ;
            ++ i;
            while ( i < (int)str.size() && isdigit(str[i]) ) {
                j = j * 10 + str[i++] - '0' ;
            }
            -- i;
            num.push(j) ;
        } else if (str[i] == '(') {
            op.push(str[i]) ;
        } else if (str[i] == ')') {
            while ( op.top() != '(' ) eval();
            op.pop() ;
        } else {
            while ( !op.empty() && pr[op.top()] >= pr[str[i]] ) eval() ;
            op.push(str[i]) ;
        }
    }
    while ( !op.empty() ) eval() ;
    cout << num.top() ;
    return 0 ;
}
```

2.3 单调栈、单调队列

```
//单调栈
#include <bits/stdc++.h>

using namespace std ;
```

```

int main()
{
    int n ;
    std::cin >> n ;
    stack<int> S ;
    for(int i = 1 ; i <= n ; ++ i)
    {
        int x ;
        cin >> x;
        while ( !S.empty() && S.top() >= x ) S.pop() ;
        if ( S.empty() ) {
            cout << -1 << ' ' ;
        } else {
            cout << S.top() << ' ' ;
        }
        S.push(x) ;
    }
    return 0 ;
}
```

```
//一维单调队列
#include <bits/stdc++.h>

using namespace std ;

int main()
{
    int n , k ;
    cin >> n >> k ;
    std::vector<int> a(n) ;
    for(auto & p : a) std::cin >> p ;

    deque<int> mn , mx ;
    std::vector<int> r1 , r2 ;
    for(int i = 0 ; i < n ; ++ i)
    {
        while ( !mn.empty() && a[mn.back()] > a[i] ) mn.
            pop_back();
```

```
        while ( !mx.empty() && a[mx.back()] < a[i] ) mx.
           pop_back() ;
        mx.push_back(i) ;
        mn.push_back(i) ;
        while ( i - mn.front() + 1 > k ) mn.pop_front() ;
        while ( i - mx.front() + 1 > k ) mx.pop_front() ;
        if ( i + 1 >= k ) {
            r1.push_back(a[mn.front()]) ;
            r2.push_back(a[mx.front()]) ;
        }
    }
    for(auto p : r1) cout << p << ' ' ;
    cout << '\n' ;
    for(auto p : r2) cout << p << ' ' ;
    return 0 ;
}
```

2.4 KMP

字符串重要算法之一 (模式匹配)

```
// O(n + m)
#include <bits/stdc++.h>

using namespace std ;

const int N = 1e6 + 10 ;

int n , m , ne[N] ;
string s , t ;

int main()
{
    cin >> n >> t >> m >> s ;
    s = '#' + s , t = '#' + t ;
    for(int i = 2 , j = 0 ; i <= n ; ++ i)
    {
        while ( j && t[j + 1] != t[i] ) j = ne[j] ;
```

```
        if ( t[j + 1] == t[i] ) ++ j ;
        ne[i] = j ;
    }
    for(int i = 1 , j = 0 ; i <= m ; ++ i)
    {
        while ( j && t[j + 1] != s[i] ) j = ne[j] ;
        if ( t[j + 1] == s[i] ) ++ j ;
        if ( j == n ) {
            cout << i - n << ' ' ;
            j = ne[j] ;
        }
    }
    return 0 ;
}
```

2.5 字典树 Trie

```
#include <bits/stdc++.h>

using namespace std ;

const int N = 2e5 + 10 ;

int son[N][26] , n , cnt[N] , tot ;
void insert(string str) {
    int p = 0 ;
    for(auto c : str) {
        if ( !son[p][c - 'a'] ) son[p][c - 'a'] = ++tot ;
        p = son[p][c - 'a'] ;
    }
    ++ cnt[p] ;
}
int query(string str) {
    int p = 0 ;
    for(auto c : str) {
        if ( !son[p][c - 'a'] ) return 0 ;
        p = son[p][c - 'a'] ;
```

```
    }
    return cnt[p] ;
}
int main()
{
    cin >> n ;
    for(int i = 1 ; i <= n ; ++ i)
    {
        char op ;
        string str ;
        cin >> op >> str ;
        if ( op == 'I' ) {
            insert(str) ;
        } else {
            cout << query(str) << '\n' ;
        }
    }
    return 0 ;
}
```

2.6 应用

```
//最大异或对 url: https://www.luogu.com.cn/problem/P10471
#include <bits/stdc++.h>

using namespace std ;

const int N = 3e6 + 10 ;

int n , son[N][2] , tot ;
void insert(string str)
{
    int p = 0 ;
    for(auto c : str){
        if ( !son[p][c - '0'] ) son[p][c - '0'] = ++tot ;
        p = son[p][c - '0'] ;
    }
```

```
}
int str_xor(string a,string b) {
    int ans = 0 ;
    for(int i = 0 ; i <= 30 ; ++ i)
    {
        if ( a[i] != b[i] ) {
            ans += 1 << (30 - i) ;
        }
    }
    return ans ;
}
string query(string str) {
    int p = 0 ;
    string ans ;
    for(auto c : str) {
        int t = (c - '0') ^ 1 ;
        if ( !son[p][t] ) {
            p = son[p][c - '0'] ;
            ans.push_back(c) ;
        } else {
            p = son[p][t] ;
            ans.push_back(char('0' + t)) ;
        }
    }
    return ans ;
}

int main()
{
    cin >> n ;
    std::vector<string> a(n) ;
    for(int i = 0 ; i < n ; ++ i)
    {
        int x ;
        cin >> x ;
        for(int j = 30 ; j >= 0 ; -- j){
            a[i].push_back(char('0' + (x >> j & 1))) ;
        }
```

```
54          insert(a[i]) ;
55      }
56
57      int ans = 0 ;
58      for(int i = 0 ; i < n ; ++ i)
59      {
60          ans = max(ans , str_xor(a[i] , query(a[i]))) ;
61      }
62      cout << ans << '\n' ;
63
64      return 0 ;
65  }
```

3 经典题

https://www.luogu.com.cn/problem/solution/P10837

扫描线、前缀和、二分

简要题意：对于每一个凋零玫瑰在 t_i 时刻会绽放延续 m 时刻，询问若此时可以修改 1 个凋零玫瑰的绽放时刻请问最多有多少时刻有且仅有 1 个凋零玫瑰绽放。

思路：显然我们需要知道一个关键信息，**改变一个凋零玫瑰绽放时刻对答案的影响会是什么？**设修改的凋零玫瑰绽放时刻区间为 $[t_l, t_r]$，若在此区间内包含某些时刻有且仅有 1 个凋零玫瑰绽放则对答案贡献是 -1，若有 2 个凋零玫瑰绽放则对答案贡献是 1，知道这个后只需要预处理出所有包含 1 个和 2 个凋零玫瑰绽放的时刻区间后枚举修改的凋零玫瑰通过**前缀和、二分**的手段计算求得最大即可。预处理可以通过**扫描线**的思想求得。

时间复杂度 $O(nlogn)$。

```cpp
#include <bits/stdc++.h>
#define int long long

void solve() {
  int n , m ;
  std::cin >> n >> m ;
  std::map<int , int> a;
  std::vector<int> arr(n + 1 , 0);
  for(int i = 1 ; i <= n ; ++ i)
  {
    std::cin >> arr[i] ;
    a[arr[i]] ++ ;
    a[arr[i] + m] -- ;
  }

  int cnt = 0 , x1 , x2 , ok1 = 0 , ok2 = 0 ;
  // 通过扫描线的思想预处理
  std::vector<std::pair<int , int>> b1 , b2;
```

```
19
20  for(auto &p : a)
21  {
22    cnt += p.second ;
23    if ( !p.second ) {
24      if ( ok1 ) {
25        b1.push_back({x1 , p.first - 1}) ;
26        x1 = p.first ;
27      }
28      if ( ok2 ) {
29        b2.push_back({x2 , p.first - 1}) ;
30        x2 = p.first ;
31      }
32    } else if ( cnt == 1 ) {
33      if ( !ok1 ) {
34        x1 = p.first ;
35      }
36      if ( ok2 ) {
37        b2.push_back({x2 , p.first - 1}) ;
38        ok2 = 0 ;
39      }
40      ok1 = 1 ;
41    } else if ( cnt == 2 ) {
42      if ( !ok2 ) {
43        x2 = p.first ;
44      }
45      if ( ok1 ) {
46        b1.push_back({x1 , p.first - 1}) ;
47        ok1 = 0 ;
48      }
49      ok2 = 1 ;
50    } else {
51      if ( ok1 ) {
52        b1.push_back({x1 , p.first - 1}) ;
53        ok1 = 0;
54      }
55      if ( ok2 ) {
56        b2.push_back({x2 , p.first - 1}) ;
```

```
57          ok2 = 0 ;
58        }
59      }
60    }
61    // 预处理前缀和
62    std::vector<int> s1((int)b1.size() + 1 , 0) , s2((int)b2.
         size() + 1 , 0);
63    for(int i = 0 ; i < (int)b1.size() ; ++ i)
64    {
65      s1[i + 1] = s1[i] + b1[i].second - b1[i].first + 1 ;
66    }
67    for(int i = 0 ; i < (int)b2.size() ; ++ i)
68    {
69      s2[i + 1] = s2[i] + b2[i].second - b2[i].first + 1 ;
70    }
71
72    int ans = 0 ;
73    // 枚举
74    for(int i = 1 ; i <= n ; ++ i)
75    {
76      int x = arr[i] , y = arr[i] + m - 1 ; // 影响时刻区间[x ,
           y]
77
78      int l = 0 , r = b1.size() - 1 , lo = -1 , hg = -1 , r1 = 0
           , r2 = 0 ;
79      while ( r >= l ) {
80        int mid = ( l + r ) >> 1 ;
81        if ( b1[mid].first >= x ) {
82          r = mid - 1 ;
83          lo = mid ;
84        } else {
85          l = mid + 1 ;
86        }
87      }
88      l = 0 , r = b1.size() - 1 ;
89      while ( r >= l ) {
90        int mid = (l + r) >> 1 ;
91        if ( b1[mid].second <= y ) {
```

```
92          l = mid + 1 ;
93          hg = mid ;
94        } else {
95          r = mid - 1 ;
96        }
97      }
98      if ( lo != -1 ) {
99        r1 = s1[hg + 1] - s1[lo] ;
100     }
101     l = 0 , r = b2.size() - 1 , lo = -1 , hg = -1 ;
102     while ( r >= l ) {
103       int mid = ( l + r ) >> 1 ;
104       if ( b2[mid].first >= x ) {
105         r = mid - 1 ;
106         lo = mid ;
107       } else {
108         l = mid + 1 ;
109       }
110     }
111     l = 0 , r = b2.size() - 1 ;
112     while ( r >= l ) {
113       int mid = (l + r) >> 1 ;
114       if ( b2[mid].second <= y ) {
115         l = mid + 1 ;
116         hg = mid ;
117       } else {
118         r = mid - 1 ;
119       }
120     }
121     if ( lo != -1 ) {
122       r2 = s2[hg + 1] - s2[lo] ;
123     }
124     ans = std::max(ans , s1[b1.size()] - r1 + r2 + m) ;
125   }
126   std::cout << ans << '\n' ;
127 }
128
129 signed main()
```

```
130 {
131     std::cin.tie(nullptr)->std::ios::sync_with_stdio(false) ;
132 
133     int t = 1 ;
134     // std::cin >> t ;
135     while ( t-- ) {
136         solve() ;
137     }
138 
139     return 0 ;
140 }
```