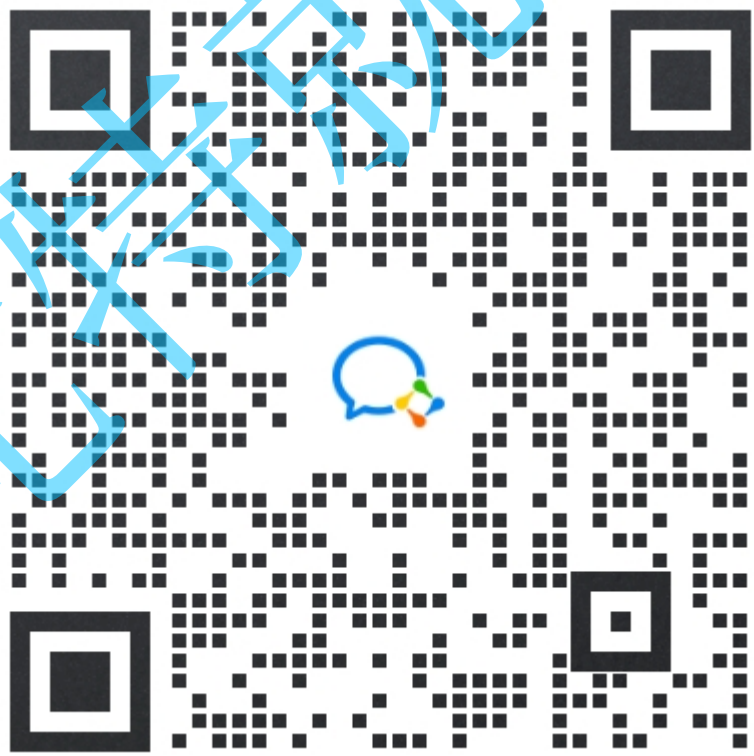
笔试强训第 07 周

版权说明

对比特算法感兴趣，可以联系这个微信。



板书链接

https://gitee.com/wu-xiaozhe/written-exam-training

Day37

1. 旋转字符串（字符串）

（题号：1024728）

1. **题目链接：** 旋转字符串

2. **题目描述：**

> 题目描述：字符串旋转:
> 给定两字符串A和B，如果能将A从中间某个位置分割为左右两部分字符串（可以为空串），并将左边的字符串移动到右边字符串后面组成新的字符串可以变为字符串B时返回true。
>
> 例如：如果A='youzan'，B='zanyou'，A按'you''zan'切割换位后得到'zanyou'和B相同，返回true。
> 再如：如果A='abcd'，B='abcd'，A切成'abcd'和''（空串），换位后可以得到B，返回true。
>
> 数据范围：A,B字符串长度满足 $n \leq 1000$，保证字符串中仅包含小写英文字母和阿拉伯数字
> 进阶： 时间复杂度 $O(n)$，空间复杂度 $O(n)$
>
> 补充说明:

```
示例1
输入："youzan","zanyou"
输出：true
说明：

示例2
输入："youzan","zyouan"
输出：false
说明：

示例3
输入："nowcoder","nowcoder"
输出：true
说明：
```

3. **解法（字符串查找）：**

算法思路：

第一种解法：

按照题目的要求模拟，每次旋转一下 A 字符串，看看是否和 B 字符串相同。

第二种解法：需要找到字符串旋转之后能匹配所满足的性质。

如果 A 字符串能够旋转之后得到 B 字符串的话，在 A 字符串倍增之后的新串中，一定是可以找到 B 字符串的。

因此，我们仅需让 A 字符串倍增，然后查找 B 字符串即可。

C++ 算法代码：

```cpp
class Solution
{
public:
    bool solve(string A, string B)
    {
        if(A.size() != B.size()) return false;
        return (A + A).find(B) != -1;
    }
};
```

Java 算法代码：

```java
import java.util.*;

public class Solution
{
    public boolean solve (String A, String B)
    {
        if(A.length() != B.length()) return false;
        return (A + A).contains(B);
    }
}
```

2. 合并k个已排序的链表（链表）

（题号：724）

1. **题目链接：** **NC51 合并k个已排序的链表**
2. **题目描述：**

题目描述：合并 k 个升序的链表并将结果作为一个升序的链表返回其头节点。

数据范围：节点总数 $0 \leq n \leq 5000$，每个节点的val满足 $|val| <= 1000$
要求：时间复杂度 $O(nlogn)$

补充说明：

示例1
输入：[{1,2,3},{4,5,6,7}]
输出：{1,2,3,4,5,6,7}
说明：

示例2
输入：[{1,2},{1,4,5},{6}]
输出：{1,1,2,4,5,6}
说明：

3. 解法：

算法思路：

利用堆，模拟题~

C++ 算法代码：

```cpp
/**
 * struct ListNode {
 *  int val;
 *  struct ListNode *next;
 *  ListNode(int x) : val(x), next(nullptr) {}
 * };
 */
class Solution
{
    struct cmp
    {
        bool operator()(ListNode* l1, ListNode* l2)
        {
            return l1->val > l2->val;
        }
    };

public:
    ListNode* mergeKLists(vector<ListNode*>& lists)
    {
        priority_queue<ListNode*, vector<ListNode*>, cmp> heap; // 小根堆
        for(auto head : lists)
```

```
        {
            if(head != nullptr)
            {
                heap.push(head);
            }
        }

        ListNode* ret = new ListNode(0);
        ListNode* prev = ret;
        while(heap.size())
        {
            ListNode* t = heap.top();
            heap.pop();
            prev = prev->next = t;
            if(t->next != nullptr)
            {
                heap.push(t->next);
            }
        }
        return ret->next;
    }
};
```

Java 算法代码：

```
import java.util.*;

/*
 * public class ListNode {
 *   int val;
 *   ListNode next = null;
 *   public ListNode(int val) {
 *     this.val = val;
 *   }
 * }
 */

public class Solution
{
    public ListNode mergeKLists (ArrayList<ListNode> lists)
    {
        PriorityQueue<ListNode> heap = new PriorityQueue<>((l1, l2) ->
        {
            return l1.val - l2.val;
```

```
        });

        for(int i = 0; i < lists.size(); i++)
        {
            if(lists.get(i) != null)
            {
                heap.offer(lists.get(i));
            }
        }

        ListNode ret = new ListNode(0);
        ListNode prev = ret;
        while(!heap.isEmpty())
        {
            ListNode t = heap.poll();
            prev = prev.next = t;
            if(t.next != null)
            {
                heap.offer(t.next);
            }
        }

        return ret.next;
    }
}

```

3. 滑雪（记忆化搜索）

（题号：2362693）

1. **题目链接：** DP18 滑雪
2. **题目描述：**

题目描述：给定一个$n \times m$ 的矩阵，矩阵中的数字表示滑雪场各个区域的高度，你可以选择从任意一个区域出发，并滑向任意一个周边的高度严格更低的区域（周边的定义是上下左右相邻的区域）。请问整个滑雪场中最长的滑道有多长？(滑道的定义是从一个点出发的一条高度递减的路线）。
(本题和矩阵最长递增路径类似，该题是当年NOIP的一道经典题)

数据范围：$1 \leq n, m \leq 100$，矩阵中的数字满足 $1 \leq val \leq 1000$

输入描述：第一行输入两个正整数 n 和 m 表示矩阵的长宽。
后续 n 行输入中每行有 m 个正整数，表示矩阵的各个元素大小。

输出描述：输出最长递减路线。

补充说明：

示例1
输入：5 5
1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
输出：25
说明：从25出发，每次滑向周边比当前小 1 的区域。 25->24->23->22->.....->1

3. 解法：

算法思路：

矩阵最长递增路径变成矩阵最长递减路径~

C++ 算法代码：

```cpp
#include <iostream>
using namespace std;

const int N = 110;

int n, m;
int arr[N][N];
int dp[N][N];

int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0, 0};

int dfs(int i, int j)
{
    if(dp[i][j]) return dp[i][j];

    int len = 1;
    for(int k = 0; k < 4; k++)
    {
```

```
        int x = i + dx[k], y = j + dy[k];
        if(x >= 1 && x <= n && y >= 1 && y <= m && arr[x][y] < arr[i][j])
        {
            len = max(len, 1 + dfs(x, y));
        }
    }
    dp[i][j] = len;
    return len;
}

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            cin >> arr[i][j];
        }
    }

    int ret = 1;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            ret = max(ret, dfs(i, j));
        }
    }

    cout << ret << endl;

    return 0;
}

```


Java 算法代码：

```
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static int N = 110;
```

```java

    public static int n, m;
    public static int[][] arr = new int[N][N];
    public static int[][] dp = new int[N][N];

    public static int[] dx = {0, 0, 1, -1};
    public static int[] dy = {1, -1, 0, 0};

    public static int dfs(int i, int j)
    {
        if(dp[i][j] != 0) return dp[i][j];

        int len = 1;
        for(int k = 0; k < 4; k++)
        {
            int x = i + dx[k], y = j + dy[k];
            if(x >= 0 && x < n && y >= 0 && y < m && arr[x][y] < arr[i][j])
            {
                len = Math.max(len, 1 + dfs(x, y));
            }
        }
        dp[i][j] = len;
        return len;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt(); m = in.nextInt();
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                arr[i][j] = in.nextInt();
            }
        }

        int ret = 0;
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                ret = Math.max(ret, dfs(i, j));
            }
        }

        System.out.println(ret);
```

```
54      }
55  }
```

Day38

1. 天使果冻（递推）

（题号：1435206）

1. **题目链接：** 天使果冻

2. **题目描述：**

题目描述：

> Angelic Jelly

有 n 个果冻排成一排。第 i 个果冻的美味度是 a_i。
天使非常喜欢吃果冻，但她想把最好吃的果冻留到最后收藏。天使想知道前 x 个果冻中，美味度第二大的果冻有多少美味度？
一共有 q 次询问。
注：如果最大的数有两个以上，默认第二大的等于最大的。例如，$[2,3,4,2,4]$ 这个序列，第二大的数是4。

输入描述：第一行一个正整数 n 。
第二行 n 个正整数 a_i，用空格隔开。
第三行一个正整数 q 。
接下来的 q 行，每行一个正整数 x ，代表一次询问。
数据范围：1≤q≤1e5，1≤ai≤1e9，2≤x≤n≤1e5

输出描述：输出 q 行，每行一个正整数，代表一次询问，输出前 x 个果冻中美味度第二大的值。

补充说明：

示例1
输入：5
1 2 5 3 5
4
2
3
4
5
输出：1
2
3
5
说明：前2个数，第二大的是1。
前3个数，第二大的是2。
前4个数，第二大的是3。
前5个数，第二大的是5。

3. **解法：**

算法思路：

需要两个数组绑定后排序，因此可以搞一个 pair 存一下，然后排序。

C++ 算法代码：

```cpp
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

const int N = 1e5 + 10;

pair<int, string> sp[N];

int main()
{
    int n;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> sp[i].second >> sp[i].first;
    }

    sort(sp, sp + n);

    for(int i = 0; i < n; i++)
    {
        cout << sp[i].second << endl;
    }

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
```

```
        int[] arr = new int[n + 1];
        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
        }

        // 更新最大值以及次大值shuzu
        int[] f = new int[n + 1];
        int[] g = new int[n + 1];
        f[1] = arr[1];
        for(int i = 2; i <= n; i++)
        {
            int x = arr[i];
            f[i] = Math.max(f[i - 1], x);
            if(x >= f[i - 1]) g[i] = f[i - 1];
            else if(x >= g[i - 1]) g[i] = x;
            else g[i] = g[i - 1];
        }

        int q = in.nextInt();
        while(q-- != 0)
        {
            int x = in.nextInt();
            System.out.println(g[x]);
        }
    }
}
```

2. dd爱旋转（模拟）

（题号：1703525）

1. **题目链接：** dd爱旋转
2. **题目描述：**

题目描述：读入一个$n*n$的矩阵，对于一个矩阵有以下两种操作
1:顺时针旋180°
2:关于行镜像
如
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$变成$\begin{bmatrix}3 & 4\\1 & 2\end{bmatrix}$
给出q个操作，输出操作完的矩阵

输入描述：第一行一个数n(1≤n≤1000)，表示矩阵大小
接下来n行，每行n个数，描述矩阵，其中数字范围为[1,2000]
一下来一行一个数q(1≤q≤100000)，表示询问次数
接下来q行，每行一个数x(x=1或x=2)，描述每次询问

输出描述：n行，每行n个数，描述操作后的矩阵

补充说明：

示例1
输入：2
1 2
3 4
1
1
输出：4 3
2 1
说明：

示例2
输入：2
1 2
3 4
1
2
输出：3 4
1 2
说明：

3. 解法：

算法思路：

模拟题，但是需要不能直接无脑模拟，要思考一下规律~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 1010;

int n;
int g[N][N];

void setRow() // 行对称
```

```
11 {
12     for(int i = 0; i < n / 2; i++)
13     {
14         for(int j = 0; j < n; j++)
15         {
16             swap(g[i][j], g[n - 1 - i][j]);
17         }
18     }
19 }
20
21 void setCol() // 列对称
22 {
23     for(int j = 0; j < n / 2; j++)
24     {
25         for(int i = 0; i < n; i++)
26         {
27             swap(g[i][j], g[i][n - 1 - j]);
28         }
29     }
30 }
31
32 int main()
33 {
34     cin >> n;
35     for(int i = 0; i < n; i++)
36     {
37         for(int j = 0; j < n; j++)
38         {
39             cin >> g[i][j];
40         }
41     }
42
43     int q, x;
44     cin >> q;
45
46     int row = 0, col = 0;
47     while(q--)
48     {
49         cin >> x;
50         if(x == 1) row++, col++;
51         else row++;
52     }
53
54     row %= 2; col %= 2;
55     if(row) setRow();
56     if(col) setCol();
57
```

```
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            cout << g[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.*;
import java.io.*;

public class Main
{
    public static PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(System.out)));
    public static Read in = new Read();

    public static int n;
    public static int[][] arr;

    public static void setRow()
    {
        for(int i = 0; i < n / 2; i++)
        {
            for(int j = 0; j < n; j++)
            {
                // arr[i][j] 与 arr[n - 1 - i][j]
                int tmp = arr[i][j];
                arr[i][j] = arr[n - 1 - i][j];
                arr[n - 1 - i][j] = tmp;
            }
        }
    }

    public static void setCol()
    {
        for(int j = 0; j < n / 2; j++)
        {
```

```
30              for(int i = 0; i < n; i++)
31              {
32                  // arr[i][j] 与 arr[i][n - 1 - j]
33                  int tmp = arr[i][j];
34                  arr[i][j] = arr[i][n - 1 - j];
35                  arr[i][n - 1 - j] = tmp;
36              }
37          }
38      }
39
40      public static void main(String[] args) throws IOException
41      {
42          n = in.nextInt();
43          arr = new int[n][n];
44          for(int i = 0; i < n; i++)
45          {
46              for(int j = 0; j < n; j++)
47              {
48                  arr[i][j] = in.nextInt();
49              }
50          }
51
52          int q = in.nextInt();
53          int row = 0, col = 0;
54          while(q-- != 0)
55          {
56              int x = in.nextInt();
57              if(x == 1)
58              {
59                  row++; col++;
60              }
61              else
62              {
63                  row++;
64              }
65          }
66
67          // 优化
68          row %= 2; col %= 2;
69          if(row != 0) setRow();
70          if(col != 0) setCol();
71
72          for(int i = 0; i < n; i++)
73          {
74              for(int j = 0; j < n; j++)
75              {
76                  out.print(arr[i][j] + " ");
```

```
            }
            out.println("");
        }

        out.close();
    }
}


class Read // 自定义快速读入
{
    StringTokenizer st = new StringTokenizer("");
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    String next() throws IOException
    {
        while(!st.hasMoreTokens())
        {
            st = new StringTokenizer(bf.readLine());
        }
        return st.nextToken();
    }

    String nextLine() throws IOException
    {
        return bf.readLine();
    }

    int nextInt() throws IOException
    {
        return Integer.parseInt(next());
    }

    long nextLong() throws IOException
    {
        return Long.parseLong(next());
    }

    double nextDouble() throws IOException
    {
        return Double.parseDouble(next());
    }
}
```


3. 小红取数（动态规划 - 01 背包 + 同余）

（题号：1831976）

1. 题目链接： DP40 小红取数

2. 题目描述：

> 题目描述：小红拿到了一个数组，她想取一些数使得取的数之和尽可能大，但要求这个和必须是 k 的倍数。
> 你能帮帮她吗?
>
> 输入描述：第一行输入两个正整数 n 和 k
> 第二行输入 n 个正整数 a_i
> $1 \leq n, k \leq 10^3$
> $1 \leq a_i \leq 10^{10}$
>
> 输出描述：如果没有合法方案，输出 -1。
> 否则输出最大的和。
>
> 补充说明：
>
> 示例1
> 输入：5 5
> 4 8 2 9 1
> 输出：20
> 说明：取后四个数即可

3. 解法：

算法思路：

这道题是不能用空间优化的~

C++ 算法代码：

```cpp
#include <iostream>
#include <cstring>
using namespace std;

typedef long long LL;

const int N = 1010;

int n, k;
LL a[N];
LL dp[N][N];

int main()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> a[i];

    memset(dp, -0x3f, sizeof dp);
    dp[0][0] = 0;
```

```
20
21      for(int i = 1; i <= n; i++)
22      {
23          for(int j = 0; j < k; j++)
24          {
25              dp[i][j] = max(dp[i - 1][j], dp[i - 1][(j - a[i] % k + k) % k] +
    a[i]);
26          }
27      }
28
29      if(dp[n][0] <= 0) cout << -1 << endl;
30      else cout << dp[n][0] << endl;
31
32      return 0;
33  }
```

Java 算法代码：

```
1   import java.util.*;
2
3   // 注意类名必须为 Main, 不要有任何 package xxx 信息
4   public class Main
5   {
6       public static void main(String[] args)
7       {
8           Scanner in = new Scanner(System.in);
9           int n = in.nextInt(), k = in.nextInt();
10          long[] a = new long[n + 1];
11          for(int i = 1; i <= n; i++)
12          {
13              a[i] = in.nextLong();
14          }
15
16          long[][] dp = new long[n + 1][k];
17          for(int i = 0 ; i <= n; i++)
18          {
19              for(int j = 0; j < k; j++)
20              {
21                  dp[i][j] = (long)-1e16;
22              }
23          }
24          dp[0][0] = 0;
25
26          for(int i = 1; i <= n; i++)
```

```
27          {
28              for(int j = 0; j < k; j++)
29              {
30                  dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][(int)((j - a[i] %
    (long)k + (long)k) % (long)k)] + a[i]);
31              }
32          }
33
34          if(dp[n][0] > 0) System.out.println(dp[n][0]);
35          else System.out.println(-1);
36      }
37  }
38
```

Day39

1. 神奇的字母（二）（哈希 + 字符串）

（题号：955181）

1. **题目链接：** 神奇的字母（二）
2. **题目描述：**

> 题目描述：我有个神奇的字母。
> 但我不告诉你是什么字母。
> 什么？你想知道这个字母是什么？那你就来猜呀~
> 字母的范围从'a'到'z'。
> 我会给你一段话，神奇的字母就是出现次数最多的那个字母哦~
>
> 输入描述：一段话，仅由英文小写字母和空格组成。这段话可能有很多行。（保证存在出现次数最多的一个字母）
> 数据范围：所有字符串总长度之和不超过1000。
>
> 输出描述：出现次数最多的那个神奇的字母。
>
> 补充说明：今天不是愚人节哦~

> 示例1
> 输入：ranko sekai ichiban kawaii
> ranko saikou
> 输出：a
> 说明：这段话只有a出现了7次，其他字母都小于7次。

3. **解法：**

算法思路：

哈希表。

C++ 算法代码：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s;
    int hash[26] = { 0 };
    char ret = 0;
    int maxCount = 0;

    while(cin >> s)
    {
        for(auto ch : s)
        {
            if(++hash[ch - 'a'] > maxCount)
            {
                ret = ch;
                maxCount = hash[ch - 'a'];
            }
        }
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int[] hash = new int[26];
        char ret = 0;
        int maxCount = 0;

```

```
        while(in.hasNext())
        {
            char[] s = in.next().toCharArray();
            for(char ch : s)
            {
                if(++hash[ch - 'a'] > maxCount)
                {
                    ret = ch;
                    maxCount = hash[ch - 'a'];
                }
            }
        }

        System.out.println(ret);
    }
}
```

2. 字符编码（哈夫曼编码）

（题号：26165）

1. **题目链接：** MT7 字符编码
2. **题目描述：**

> 题目描述：请设计一个算法，给一个字符串进行二进制编码（哈夫曼编码），使得编码后字符串的长度最短。
>
> 数据范围：字符串长度满足 $1 < n \leq 1000$ ，本题有多组输入
>
> 输入描述：每组数据一行，为待编码的字符串。保证字符串长度小于等于1000。
>
> 输出描述：一行输出最短的编码后长度。
>
> 补充说明：
>
> 示例1
> 输入：MT-TECH-TEAM
> 输出：33
> 说明：

3. **解法：**

算法思路：

哈夫曼编码模板题~

C++ 算法代码：

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <queue>
using namespace std;

int main()
{
    string s;
    while(cin >> s)
    {
        // 1. 先统计每个字符的频次
        int hash[300] = { 0 };
        for(auto ch : s)
        {
            hash[ch]++;
        }

        // 2. 把所有的频次放入堆里面
        priority_queue<int, vector<int>, greater<int>> heap;
        for(int i = 0; i < 300; i++)
        {
            if(hash[i]) heap.push(hash[i]);
        }

        // 3. 哈夫曼编码
        int ret = 0;
        while(heap.size() > 1)
        {
            int t1 = heap.top(); heap.pop();
            int t2 = heap.top(); heap.pop();
            ret += t1 + t2;
            heap.push(t1 + t2);
        }

        cout << ret << endl;
    }

    return 0;
}
```


Java 算法代码：

```java
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        while(in.hasNext())
        {
            char[] s = in.next().toCharArray();
            // 1. 统计所有字符的频次
            int[] hash = new int[300];
            for(char ch : s)
            {
                hash[ch]++;
            }

            // 2. 把所有的频次放入堆里面
            PriorityQueue<Integer> heap = new PriorityQueue<>();
            for(int i = 0; i < 300; i++)
            {
                if(hash[i] != 0)
                {
                    heap.offer(hash[i]);
                }
            }

            // 3. 哈夫曼编码
            int ret = 0;
            while(heap.size() > 1)
            {
                int t1 = heap.poll();
                int t2 = heap.poll();
                ret += t1 + t2;
                heap.offer(t1 + t2);
            }
            System.out.println(ret);
        }

    }
}
```


3. 最少的完全平方数（动态规划）

（题号：2380254）

1. 题目链接： DP43 最少的完全平方数

2. 题目描述：

> 题目描述：给定一个正整数n，请找出最少个数的完全平方数，使得这些完全平方数的和等于n。
> 完全平方指用一个整数乘以自己例如1*1，2*2，3*3等，依此类推。若一个数能表示成某个整数的平方的形式，则称这个数为完全平方数。例如:1，4，9，和16都是完全平方数，但是2，3，5，8，11等等不是
>
> 数据范围:$1 \leq n \leq 10^4$
>
> 输入描述：仅一行，输入一个正整数 n
>
> 输出描述：按题目要求输出完全平方数之和为n的最少个数
>
> 补充说明：
>
> 示例1
> 输入：5
> 输出：2
> 说明：1+4=5
>
> 示例2
> 输入：8
> 输出：2
> 说明：4+4=8
>
> 示例3
> 输入：9
> 输出：1
> 说明：$3^2 = 9$

3. 解法：

算法思路：

完全背包问题。

C++ 算法代码：

```cpp
#include <iostream>
#include <cstring>
using namespace std;

const int N = 1e4 + 10;

int n;
int dp[N];

int main()
{
    cin >> n;
    memset(dp, 0x3f, sizeof dp);
```

```
    dp[0] = 0;

    for(int i = 1; i * i <= n; i++)
    {
        for(int j = i * i; j <= n; j++)
        {
            dp[j] = min(dp[j], dp[j - i * i] + 1);
        }
    }

    cout << dp[n] << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] dp = new int[n + 1];
        for(int i = 0; i <= n; i++)
        {
            dp[i] = 0x3f3f3f3f;
        }
        dp[0] = 0;

        for(int i = 1; i * i <= n; i++)
        {
            for(int j = i * i; j <= n; j++)
            {
                dp[j] = Math.min(dp[j], dp[j - i * i] + 1);
            }
        }

        System.out.println(dp[n]);
    }
}
```

Day40

1. 游游的字母串（枚举）

（题号：10274355）

1. **题目链接：** 游游的字母串

2. **题目描述：**

> 题目描述：对于一个小写字母而言，游游可以通过一次操作把这个字母变成相邻的字母。'a'和'b'相邻，'b'和'c'相邻，以此类推。特殊的，'a'和'z'也是相邻的。可以认为，小写字母的相邻规则为一个环。
> 游游拿到了一个仅包含小写字母的字符串，她想知道，使得所有字母都相等至少要多少次操作?
>
> 输入描述：一个仅包含小写字母，长度不超过100000的字符串。
>
> 输出描述：一个整数，代表最小的操作次数。
>
> 补充说明：
>
> 示例1
> 输入：yab
> 输出：3
> 说明：第一次操作，把'y'变成'z'，字符串变成了"zab"
> 第二次操作，把'b'变成'a'，字符串变成了"zaa"
> 第三次操作，把'z'变成'a'，字符串变成了"aaa"

3. **解法：**

算法思路：

枚举所有可能变成的字符情况。

C++ 算法代码：

```cpp
#include <iostream>
#include <cmath>
#include <string>

using namespace std;

int main()
{
    string s;
    cin >> s;

    int ret = 1e9;
```

```
    for(char ch = 'a'; ch <= 'z'; ch++)
    {
        int sum = 0;
        for(auto x : s)
        {
            sum += min(abs(x - ch), 26 - abs(x - ch));
        }
        ret = min(ret, sum);
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        char[] s = in.next().toCharArray();

        int ret = (int)1e9;
        for(char ch = 'a'; ch <= 'z'; ch++)
        {
            int sum = 0;
            for(int i = 0; i < s.length; i++)
            {
                sum += Math.min(Math.abs(s[i] - ch), 26 - Math.abs(s[i] - ch));
            }
            ret = Math.min(ret, sum);
        }

        System.out.println(ret);
    }
}
```

2. 体育课测验(二)（拓扑排序）

（题号：2382473）

1. 题目链接： NC316 体育课测验(二)

2. 题目描述：

题目描述：体育课共有$numProject$个考核项目，编号为0到$numProject-1$，考核中每两个项目被划分为一组得到分组数组$groups_i$，现规定若想完成项目$groups_i[0]$，必须先完成$groups_i[1]$。保证所有分组互不相同，若分组情况能顺利完成考核，请返回任意的一个完成顺序,否则返回空数组 。

数据范围：

$1 \leq numProject \leq 2000$

$1 \leq groups_i.length \leq numProject * (numProject-1)$

补充说明：

示例1

输入：3,[[2,1]]

输出：[1,2,0]

说明：要先完成1号项目，再完成2号项目，而0号项目不受约束，故可以以1 2 0的顺序完成项目。

示例2

输入：3,[[1,0], [0,1]]

输出：[]

说明：第一个分组要求先完成0号项目，再完成1号项目；而第二个分组要求先完成1号项目，再完成0号项目，自相矛盾，故不可以完成项目。

3. 解法：

算法思路：

简单拓扑排序的应用。

C++ 算法代码：

```cpp
class Solution
{
public:
    vector<int> findOrder(int n, vector<vector<int> >& groups)
    {
        vector<vector<int>> edges(n); // 存储边
        vector<int> in(n); // 入度

        // 1. 建图
        for(auto v : groups)
        {
            int a = v[0], b = v[1];  // b -> a
            edges[b].push_back(a);
            in[a]++;
        }

        queue<int> q;
```

```
        // 2. 入度为 0 的点，加入到队列里面
        for(int i = 0; i < n; i++)
        {
            if(in[i] == 0)
            {
                q.push(i);
            }
        }

        // 3. 拓扑排序
        vector<int> ret;
        while(q.size())
        {
            int a = q.front();
            q.pop();
            ret.push_back(a);
            for(auto b : edges[a]) // a -> b
            {
                if(--in[b] == 0)
                {
                    q.push(b);
                }
            }
        }

        if(ret.size() == n) return ret;
        else return {};
    }
};

```

Java 算法代码：

```
import java.util.*;

public class Solution {
    public ArrayList<Integer> findOrder (int n, ArrayList<ArrayList<Integer>>
groups)
    {
        List<List<Integer>> edges = new ArrayList<>(); // 存储边
        for(int i = 0; i < n; i++)
        {
            edges.add(new ArrayList<>());
        }
```

```
11          int[] in = new int[n]; // 入度
12
13          // 1. 建图
14          for(int i = 0; i < groups.size(); i++)
15          {
16              int a = groups.get(i).get(0), b = groups.get(i).get(1); // b -> a
17              in[a]++;
18              edges.get(b).add(a);
19          }
20
21          Queue<Integer> q = new LinkedList<>();
22          // 2. 入度为 0 的点加入到队列中
23          for(int i = 0; i < n; i++)
24          {
25              if(in[i] == 0)
26              {
27                  q.add(i);
28              }
29          }
30
31          ArrayList<Integer> ret = new ArrayList<>();
32          // 3. 拓扑排序（BFS）
33          while(!q.isEmpty())
34          {
35              int a = q.poll();
36              ret.add(a);
37              for(int b : edges.get(a)) // a -> b
38              {
39                  if(--in[b] == 0)
40                  {
41                      q.add(b);
42                  }
43              }
44          }
45
46          if(ret.size() == n) return ret;
47          else return new ArrayList<>();
48      }
49  }
50
```


3. 合唱队形（动态规划）

（题号：2361966）

1. 题目链接： DP16 合唱队形

2. 题目描述：

题目描述：N位同学站成一排，音乐老师要请其中的 (N-K) 位同学出列，使得剩下的K位同学排成合唱队形。
合唱队形是指这样的一种队形：设K位同学从左到右依次编号为 1，2…，K，他们的身高分别为 T_1，T_2，…，T_K， 则他们的身高满足 $t_1 < t_2 \ldots < t_i > t_{i+1} > \ldots > t_{k-1} > t_k (1 \le i \le k)$
你的任务是，已知所有 n 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

数据范围： $1 \le n \le 1000$ ，身高满足 $130 \le t_i \le 230$

输入描述：第一行输入一个正整数 n 表示同学的总数。
第二行有 n 个整数，用空格分隔，第 i 个整数 t_i 是第 i 位同学的身高(厘米)。

输出描述：输出仅有一个整数，即最少需要几个同学出列

补充说明：

示例1
输入：8
186 186 150 200 160 130 197 220
输出：4
说明：

3. 解法：

算法思路：

最长上升子序列模型。

C++ 算法代码：

```cpp
#include <iostream>
using namespace std;

const int N = 1010;

int n;
int arr[N], f[N], g[N];

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> arr[i];

    // 从前向后
    for(int i = 1; i <= n; i++)
    {
        f[i] = 1;
        for(int j = 1; j < i; j++)
        {
            if(arr[j] < arr[i])
```

```
            {
                f[i] = max(f[i], f[j] + 1);
            }
        }
    }

    // 从后向前
    for(int i = n; i >= 1; i--)
    {
        g[i] = 1;
        for(int j = i + 1; j <= n; j++)
        {
            if(arr[i] > arr[j])
            {
                g[i] = max(g[i], g[j] + 1);
            }
        }
    }

    int len = 0;
    for(int i = 1; i <= n; i++)
    {
        len = max(len, f[i] + g[i] - 1);
    }

    cout << n - len << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] arr = new int[n + 1];
        for(int i = 1; i <= n; i++)
        {
```

```
            arr[i] = in.nextInt();
        }
        int[] f = new int[n + 1];
        int[] g = new int[n + 1];

        // 从左往右
        for(int i = 1; i <= n; i++)
        {
            f[i] = 1;
            for(int j = 1; j < i; j++)
            {
                if(arr[i] > arr[j])
                {
                    f[i] = Math.max(f[i], f[j] + 1);
                }
            }
        }

        // 从右往左
        for(int i = n; i >= 1; i--)
        {
            g[i] = 1;
            for(int j = i + 1; j <= n; j++)
            {
                if(arr[i] > arr[j])
                {
                    g[i] = Math.max(g[i], g[j] + 1);
                }
            }
        }

        int len = 0;
        for(int i = 1; i <= n; i++)
        {
            len = Math.max(len, f[i] + g[i] - 1);
        }

        System.out.println(n - len);
    }
}

```


Day41

1. 棋子翻转（模拟）

（题号：2315634）

1. **题目链接：** MT2 棋子翻转

2. **题目描述：**

题目描述：在 4x4 的棋盘上摆满了黑白棋子，黑白两色棋子的位置和数目随机，其中0代表白色，1代表黑色；左上角坐标为 (1,1) ，右下角坐标为 (4,4) 。
现在依次有一些翻转操作，要对以给定翻转坐标(x,y)（也即第x行第y列）为中心的上下左右四个棋子的颜色进行翻转。
给定两个数组 A 和 f ，分别代表 初始棋盘 和 哪些要进行翻转的位置(x,y) ，请返回经过所有翻转操作后的棋盘。

例如输入[[0,0,1,1],[1,0,1,0],[0,1,1,0],[0,0,1,0]],[[2,2],[3,3],[4,4]]时，初始键盘如下图所示：

0	0	1	1
1	0	1	0
0	1	1	0
0	0	1	0

对应的输出为[[0,1,1,1],[0,0,1,0],[0,1,1,0],[0,0,1,0]]，如下图所示：

0	1	1	1
0	0	1	0
0	1	1	0
0	0	1	0

3. **解法：**

算法思路：

模拟即可。注意点：

- 如何访问上下左右四个方向；
- 访问的时候不要越界；
- 下标的对应关系；

- 如何优雅地翻转~

C++ 算法代码：

```
class Solution
{
public:

    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};

    vector<vector<int> > flipChess(vector<vector<int> >& A, vector<vector<int>
>& f)
    {
        for(auto& v : f)
        {
            int a = v[0] - 1, b = v[1] - 1;
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < 4 && y >= 0 && y < 4)
                {
                    A[x][y] ^= 1;
                }
            }
        }

        return A;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};

    public int[][] flipChess (int[][] A, int[][] f)
```

```
    {
        for(int[] i : f)
        {
            int a = i[0] - 1, b = i[1] - 1;
            for(int j = 0; j < 4; j++)
            {
                int x = a + dx[j], y = b + dy[j];
                if(x >= 0 && x < 4 && y >= 0 && y < 4)
                {
                    A[x][y] ^= 1;
                }
            }
        }

        return A;
    }
}
```

2. 宵暗的妖怪（动态规划）

（题号：1115884）

1. **题目链接：** 宵暗的妖怪
2. **题目描述：**

题目描述：露米娅作为宵暗的妖怪，非常喜欢吞噬黑暗。
这天，她来到了一条路上，准备吞噬这条路上的黑暗。
这条道路一共被分为n 部分，每个部分上的黑暗数量为a_i 。
露米娅每次可以任取 **连续的 未被吞噬过的** 三部分，将其中的黑暗全部吞噬，并获得中间部分的饱食度。
露米娅想知道，自己能获得的饱食度最大值是多少?

输入描述：第一行一个正整数n ，代表道路被分的份数。
第二行有n 个正整数a_i ，代表每一部分黑暗数量。
数据范围：$3 \leq n \leq 100000, 1 \leq a_i \leq 10^9$

输出描述：一个正整数，代表最终饱食度的最大值。

补充说明：

示例1
输入：7
2 4 1 4 2 1 8
输出：6
说明：选择[2,4,1]和[4,2,1]这两段即可。饱食度为4+2=6。

示例2
输入：7
2 4 1 7 2 1 8
输出：7
说明：选择[1,7,2]这一段即可。饱食度为7。
值得注意的是，若取两段进行吞噬，反而最多只能获得6的饱食度，并不是最大的。

3. 解法：

算法思路：

打家劫舍，但是别抢到最后一家就行了~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

typedef long long LL;
const int N = 1e5 + 10;

int n;
LL arr[N];
LL dp[N];

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> arr[i];
```

```
16
17      for(int i = 3; i <= n; i++)
18      {
19          dp[i] = max(dp[i - 1], dp[i - 3] + arr[i - 1]);
20      }
21
22      cout << dp[n] << endl;
23
24      return 0;
25  }
```

Java 算法代码：

```
1  import java.util.*;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          Scanner in = new Scanner(System.in);
8          int n = in.nextInt();
9          long[] arr = new long[n + 1];
10         for(int i = 1; i <= n; i++)
11         {
12             arr[i] = in.nextLong();
13         }
14         long[] dp = new long[n + 1];
15
16         for(int i = 3; i <= n; i++)
17         {
18             dp[i] = Math.max(dp[i - 3] + arr[i - 1], dp[i - 1]);
19         }
20
21         System.out.println(dp[n]);
22     }
23 }
```


3. 过桥（BFS）

（题号：2219848）

1. **题目链接：** 过桥

2. **题目描述：**

> 题目描述：dd被困在了一个迷幻森林，现在她面前有一条凶险的大河，河中央有n个神奇的浮块，浮块按$1 \sim n$顺序标号，但两两并不相接，第i个浮块上有一个数字$a[i]$，可能是正数，也可能是负数,每块浮块都附带一个魔法结界用于传送，当$a[i]$为正数时，dd可以选择传送到第$i+k(1 \leq k \leq a[i])$个浮块上,当dd抵达n号浮块时才可以顺利脱身，显然不管$a[n]$是多少，都没有任何意义，当$a[i]$为负时，dd只能选择标号小于等于$i+a[i]$的任意一块浮块进行传送，当$i+a[i]<1$时，默认只能传送到1的位置，每次传送都会花费$1s$的时间，随着时间的流逝，迷雾森林的空气会被逐渐榨干，她现在在1号浮块，她想知道，她最快多久能顺利脱身，如果始终无法逃脱，请输出-1
>
> 输入描述：第一行一个数n(2≤n≤2000)
> 接下来一行n个数a[i](1≤|a[i]|≤2000)表示浮块上的数字
>
> 输出描述：输出一行，表示对应的答案
>
> 补充说明：
>
> 示例1
> 输入：4
> 2 2 -1 2
> 输出：2
> 说明：1跳到2,1s
> 2跳到4,1s
> 共2s
>
> 示例2
> 输入：2
> -1 -2
> 输出：-1

3. **解法：**

算法思路：

类似层序遍历的思想。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 2010;

int n;
int arr[N];

int bfs()
{
    int left = 1, right = 1;
    int ret = 0;

```

```
    while(left <= right)
    {
        ret++;
        int r = right;
        for(int i = left; i <= right; i++)
        {
            r = max(r, arr[i] + i);
            if(r >= n)
            {
                return ret;
            }
        }
        left = right + 1;
        right = r;
    }
    return -1;
}

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> arr[i];

    cout << bfs() << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static int n;
    public static int[] arr = new int[2010];

    public static int bfs()
    {
        int left = 1, right = 1;
        int ret = 0;
        while(left <= right)
        {
            ret++;
```

```
            int r = right;
            for(int i = left; i <= right; i++)
            {
                r = Math.max(r, arr[i] + i);
                if(r >= n)
                {
                    return ret;
                }
            }
            left = right + 1;
            right = r;
        }
        return -1;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
        }

        System.out.println(bfs());
    }
}
```

Day42

1. 最大差值（模拟 + 贪心）

（题号：2314739）

1. 题目链接： MT1 最大差值
2. 题目描述：

题目描述：有一个长为 n 的数组 A，求满足 0 ≤ a ≤ b < n 的 A[b] - A[a] 的最大值。
给定数组 A 及它的大小 n，请返回最大差值。

数据范围：$2 < n \leq 2 * 10^5$ ，数组中的值满足 $0 \leq |val| \leq 5 * 10^8$

补充说明：

示例1
输入：[5,1],2
输出：0
说明：

示例2
输入：[5,6],2
输出：1
说明：

3. **解法：**

算法思路：

遍历数组的过程中，使用一个变量标记一下当前位置之前所有元素的最小值即可。

C++ 算法代码：

```cpp
class Solution
{
public:
    int getDis(vector<int>& arr, int n)
    {
        int ret = 0;
        int minPrev = arr[0];
        for(int i = 1; i < n; i++)
        {
            minPrev = min(minPrev, arr[i]);
            ret = max(ret, arr[i] - minPrev);
        }
        return ret;
    }
};
```

Java 算法代码：

```java
import java.util.*;

```

```java
public class Solution
{
    public int getDis (int[] arr, int n)
    {
        int ret = 0;
        int minPrev = arr[0];
        for(int i = 1; i < n; i++)
        {
            minPrev = Math.min(minPrev, arr[i]);
            ret = Math.max(ret, arr[i] - minPrev);
        }
        return ret;
    }
}
```

2. 兑换零钱（动态规划 - 完全背包）

（题号：2383902）

1. **题目链接：** **DP44 兑换零钱**
2. **题目描述：**

题目描述：给定数组arr，arr中所有的值都为正整数且不重复。每个值代表一种面值的货币，每种面值的货币可以使用任意张，再给定一个aim，代表要找的钱数，求组成aim的最少货币数。
如果无解，请返回-1.

数据范围：数组大小满足 $0 \leq n \leq 10000$ ， 数组中每个数字都满足 $0 < val \leq 10000$，$0 \leq aim \leq 5000$

要求：时间复杂度 $O(n \times aim)$ ，空间复杂度 $O(aim)$。

输入描述：第一行给定两个正整数分别是 n 和 aim 分别表示数组 arr 的长度和要找的钱数。
第二行给定 n 个正整数表示 arr 数组中的所有元素

输出描述：输出组成 aim 的最少货币数

补充说明：

示例1
输入：3 20
5 2 3
输出：4
说明：最少用四个 5 元凑成 20 元

示例2
输入：3 0
5 2 3
输出：0
说明：

示例3
输入：2 2
3 5
输出：-1
说明：无解

3. 解法：

算法思路：

完全背包问题。

C++ 算法代码：

```cpp
#include <iostream>
#include <cstring>
using namespace std;

const int N = 10010, M = 5010;

int n, aim;
int arr[N];
int dp[M];

int main()
{
    cin >> n >> aim;
```

```
    for(int i = 1; i <= n; i++) cin >> arr[i];
    memset(dp, 0x3f, sizeof dp);
    dp[0] = 0;

    for(int i = 1; i <= n; i++)
    {
        for(int j = arr[i]; j <= aim; j++)
        {
            dp[j] = min(dp[j], dp[j - arr[i]] + 1);
        }
    }

    if(dp[aim] >= 0x3f3f3f3f) cout << -1 << endl;
    else cout << dp[aim] << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), aim = in.nextInt();
        int[] arr = new int[n + 1];
        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
        }
        int[] dp = new int[aim + 1];

        for(int i = 0; i <= aim; i++)
        {
            dp[i] = 0x3f3f3f3f;
        }
        dp[0] = 0;

        for(int i = 1; i <= n; i++)
        {
```

```
25              for(int j = arr[i]; j <= aim; j++)
26              {
27                  dp[j] = Math.min(dp[j], dp[j - arr[i]] + 1);
28              }
29          }
30
31          if(dp[aim] >= 0x3f3f3f3f) System.out.println(-1);
32          else System.out.println(dp[aim]);
33      }
34  }
```

3. 小红的子串（前缀和 + 双指针）

（题号：10745732）

1. **题目链接：** 小红的子串

2. **题目描述：**

题目描述：小红拿到了一个长度为n的字符串，她准备选取一段子串，满足该子串中字母的种类数量在[l,r]之间。小红想知道，一共有多少种选取方案?

输入描述：第一行输入三个正整数 n, l, r
第二行输入一个仅包含小写字母的字符串。
$1 \leq n \leq 200000$
$1 \leq l \leq r \leq 26$

输出描述：合法的方案数。

补充说明：

示例1
输入：3 2 2
aba
输出：3
说明：

示例2
输入：5 1 2
abcda
输出：9
说明：

示例3
输入：10 2 4
abcaacbdef
输出：33

3. **解法：**

算法思路：

利用前缀和的思想，求种类个数在 `[l, r]` 区间内子串的个数，等于求 `[1, r]` 区间内个数 - `[1, l - 1]` 区间内个数。

求种类个数在 `[1, count]` 区间内子串的个数，可以用滑动窗口来求解。

C++ 算法代码：

```cpp
#include <iostream>
#include <string>

using namespace std;

int n, l, r;
string s;

// 找出字符种类在 [1, x] 之间的子串的个数
long long find(int x)
{
    if(x == 0) return 0;
    // 滑动窗口
    int left = 0, right = 0;
    int hash[26] = { 0 }, kinds = 0; // 统计窗口内字符种类的个数
    long long ret = 0;

    while(right < n)
    {
        if(hash[s[right] - 'a']++ == 0) kinds++;
        while(kinds > x)
        {
            if(hash[s[left] - 'a']-- == 1) kinds--;
            left++;
        }
        ret += right - left + 1;
        right++;
    }

    return ret;
}

int main()
{
    cin >> n >> l >> r >> s;

    cout << find(r) - find(l - 1) << endl;

    return 0;
```

```
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static int n, l, r;
    public static char[] s;

    // 找出字符种类在 [1, x] 之间的子串的个数
    public static long find(int x)
    {
        // 滑动窗口
        int left = 0, right = 0;
        int[] hash = new int[26];
        int kinds = 0; // 统计窗口内字符的种类
        long ret = 0;

        while(right < n)
        {
            if(hash[s[right] - 'a']++ == 0) kinds++;
            while(kinds > x)
            {
                if(hash[s[left] - 'a']-- == 1) kinds--;
                left++;
            }
            ret += right - left + 1;
            right++;
        }

        return ret;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt(); l = in.nextInt(); r = in.nextInt();
        s = in.next().toCharArray();

        System.out.println(find(r) - find(l - 1));
    }
}
```