C6 - Solution

A 小懒獭与反向点积

难度	考点
1	数组

题目分析

按照题目中给出的公式计算即可，注意一定要找好下标的对应关系，分析数据范围发现 `int` 就足够解决这道题。

可以封装一个函数使主要代码更加简洁。

示例代码

```c
#include <stdio.h>

int a[1005], b[1005];

int calc(int n) {
    int ans = 0;
    for(int i = 1; i <= n ;i++)
        ans += a[i] * b[n + 1 - i];
    return ans;
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        for(int i = 1; i <= n ;i++)
            scanf("%d", &a[i]);
        for(int i = 1; i <= n ;i++)
            scanf("%d", &b[i]);

        printf("%d\n", calc(n));
    }

    return 0;
}
```

B Moca 与水獭排队

难度	考点
2	冒泡排序

题目分析

简单来说，前半部分就是输入 n 个数字，输出 n 个数字排序后的结果。我们可以利用今天课上学习的冒泡排序解决这个问题。

在计算中位数的时候，注意涉及到两个整数的加法，且这两个整数的和的最大值可能超过 `int`。

示例代码

```c
#include <stdio.h>

int a[2005];

int main() {
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n ;i++)
        scanf("%d", &a[i]);

    // 无优化的冒泡排序，也可采用 PPT 上的优化版本提前跳出循环
    for(int i = 1; i <= n - 1 ;i++) {
        for(int j = 1; j <= n-i ;j++) {
            // 顺序不对就交换两个数字
            if(a[j] > a[j + 1]) {
                int temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }

    // 输出排序后的结果
    for(int i = 1; i <= n ;i++)
        printf("%d ", a[i]);
    printf("\n");

    // 输出中位数：轻量化的类型转换
    if(n % 2) {
        printf("%.1lf", a[(n + 1) / 2] * 1.0);
    }
    else {
        printf("%.1lf", (1.0 * a[n / 2] + a[n / 2 + 1]) / 2);
    }

    return 0;
}
```

C 魔法师水獭与悬浮咒

难度	考点
2	字符串、分支结构

题目分析

由于我们要根据长度是奇数还是偶数来采用不同的操作，所以第一步，我们需要获取字符串的长度；可以利用 `strlen` 来获取字符串的长度。

当长度为偶数时，判断字符串是否是一个回文串，也就是判断是不是对于每个 i，都有 `str[len - 1 - i] == str[i]`（下标从 0 开始），我们可以利用流程控制那节课中学到的 `flag` 技巧，记录这一结果。

对于长度为奇数也是类似，只是需要判断是不是对于每一位都不相等。

示例代码

```
#include <stdio.h>
#include <string.h>

char str[100005];

int main() {
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n ;i++) {
        scanf("%s", str);
        // 获取到字符串的长度
        int len = strlen(str), flag = 0;
        if(len % 2) {
            // 长度为奇数，看看是不是个回文串
            // 优化小技巧：可以只检查前半部分，想一想为什么
            for(int i = 0; i < len ;i++) {
                if(str[len - 1 - i] != str[i]) {
                    flag = 1;
                    break;
                }
            }
        } else {
            // 长度是偶数，看看是不是每一位都不相等
            for(int i = 0; i < len ;i++) {
                if(str[len - 1 - i] == str[i]) {
                    flag = 1;
                    break;
                }
            }
        }

        // 也可以用三目运算符简化代码
        if(!flag)
            printf("Wingardium Leviosa\n");
        else
            printf("The spell fails\n");
    }
    return 0;
}
```

D 最大池化

难度	考点
3	二维数组

题目分析

题目较为直接，就是将原矩阵分割为一个个大小为 $p \times q$ 的小矩阵，取每个小矩阵的最大值，作为池化后对应位置的元素值，不难分析出，池化后的矩阵大小为 $(n/p) \times (m/q)$。

在寻找最大值的过程中，我们可以封装一个 `max` 函数，同时注意到由于原矩阵元素的范围是整个 `int`，所以初始化元素值的时候必须要初始化为 `INT_MIN`，这样才能保证正确更新最大值。如果初始化为 0 的话，假如对应范围内的原矩阵元素都是负数，那么最大值就会错误地被记为 0。

示例代码

```
#include <stdio.h>

// 三目表达式简化，同时为了避免与库函数名字冲突起名为 myMax
int myMax(int a, int b) {
    return a > b ? a : b;
}

int a[105][105];
int ans[105][105];

int main() {
    int n, m, p, q;
    scanf("%d%d%d%d", &n, &m, &p, &q);
    for(int i = 1; i <= n ;i++)
        for(int j = 1; j <= m ;j++)
            scanf("%d", &a[i][j]);

    // 外层循环枚举起点
    for(int i = 1; i <= n ;i += p) {
        for(int j = 1; j <= m ;j += q) {
            // 正确地初始化
            ans[i/p + 1][j/q + 1] = -2147483648;
            // 内层循环更新
            for(int k = 0; k < p ;k++) {
               for(int l = 0; l < q ;l++)
                    ans[i/p + 1][j/q + 1] = myMax(ans[i/p + 1][j/q + 1], a[i + k]
[j + l]);
            }
        }
    }


    for(int i = 1; i <= n / p ;i++) {
        for(int j = 1; j <= m / q ;j++)
            printf("%d ", ans[i][j]);
        printf("\n");
    }
```

```
    return 0;
}
```

E 水獭迷宫

难度	考点
4	模拟、二维数组

题目分析

本题是一道比较直接的模拟问题，开始时我们将记录一个执行序列，然后遍历其中的每一个指令，如果该指令的目标位置是一个障碍物，那就不动；否则移动到目标位置。移动到出口就算成功（不一定要最后一个指令结束刚好到达终点，可以参考第一个样例）。

存在的一个问题是如何判断目标位置处是否有障碍物，有两种解决办法：

- 开一个二维数组 `vis`，如果位置 (i,j) 处有障碍物，就把 `vis[i][j]` 的值标记为 1，否则它的值就是 0。对于移动到的目标位置 (i,j)，只要看 `vis[i][j]` 的值就可以。这种方法要注意的一个点就是由于移动到的目标位置可能是个负数，或者是个很大的正数（因为最多只有 500 个操作所以也不会有多大），导致数组越界，出现问题。可以选择的处理方法是访问 `vis[i][j]` 前先看看下标的范围，或者给下表加上一个偏移。
- 直接把不合法的位置记录在一个数组中，每次要移动的目标位置在这个数组中顺序查询看看有没有。这种方式比较直接，也不需要特殊处理（题解采用的是这种方式）。

示例代码

```
#include <stdio.h>
#include <string.h>

char str[505];
int visX[15], visY[15];

int main() {
    int k, a, b;

    while(~scanf("%d%d%d", &k, &a, &b)) {
        scanf("%s", str);
        int len = strlen(str);

        // 读入 k 个障碍
        for(int i = 1; i <= k ;i++)
            scanf("%d%d", &visX[i], &visY[i]);

        int x = 0, y = 0;

        for(int i = 0; i < len ;i++) {
            // 标记移动的方向
            int dx = 0, dy = 0;
            if(str[i] == 'N')
                dy++;
```

```
            else if(str[i] == 'E')
                dx++;
            else if(str[i] == 'S')
                dy--;
            else if(str[i] == 'W')
                dx--;

            // 看看目标位置是不是障碍
            int flag = 0;
            for(int i = 1; i <= k ;i++) {
               if(visX[i] == x + dx && visY[i] == y + dy) {
                    flag = 1;
                    break;
                }
            }

            // 如果不是障碍就移动
            if(!flag)
                x += dx, y += dy;

            // 如果移动到出口了就出去
            if(x == a && y == b)
                break;
        }

        if(x == a && y == b)
            printf("Otter Success\n");
        else
            printf("Otter Failed\n");
    }
    return 0;
}
```

F Moca 与水獭排队 2

难度	考点
4	二分

题目分析

这道题的核心任务是实现一个二分查找算法，来查询数组中大于等于某个值的位置，给定一个已排序的高度数组 `heights`，以及一组查询高度 `h`，对于每一个查询高度，找到数组中第一个大于等于 `h` 的元素位置，并返回该位置（从1开始计数）。如果不存在大于等于 `h` 的元素，返回 `-1`。

PPT 上的代码可以实现一个无重复数的二分查找，在有重复数的查找过程中需要考虑边界条件，当 `heights[mid] >= h` 时，当前元素可能是符合条件的解，但我们还需继续向左查找，以确保找到第一个符合条件的位置。

查找的流程如下：

- 初始化 `left` 为 0，`right` 为 `n - 1`。

- 当满足 `left <= right` 时，在循环中不断更新 `mid` 为 `(left + right) / 2`。如果 `heights[mid] >= h`，说明当前元素符合要求，将 `result` 记录为 `mid`（为了从1计数），并**继续向左查找**，将 `right` 更新为 `mid - 1`；如果 `heights[mid] < h`，将 `left` 更新为 `mid + 1`，在右侧查找更大的元素。
- 循环结束后，如果找到符合要求的元素，则返回 `result + 1`（因为我们要从 1 开始），否则返回 `-1`。

示例代码

```
#include <stdio.h>

int binary_search(int arr[], int n, int h) {
    int left = 0, right = n - 1, result = -1;
    while (left <= right) {
        int mid = (left + right) / 2;
        if (arr[mid] >= h) {
            result = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    // 如果 result != -1，说明找到了，因为编号从 1 开始，我们要给 result + 1
    return result == -1 ? -1 : result + 1;
}

int heights[500005];

int main() {
    int n, m;
    scanf("%d%d", &n, &m);

    // 读入身高
    for (int i = 0; i < n; i++)
        scanf("%d", &heights[i]);

    while (m--) {
        int h;
        scanf("%d", &h);
        // 进行查找
        printf("%d\n", binary_search(heights, n, h));
    }

    return 0;
}
```

G 多项式加法 2024

难度	考点
5	合并数组

题目分析

对于本道题，如果多项式的每一项指数部分都在 $[0, 10^5]$ 这样的范围内，则我们只需要设置一个数组，每次读入的时候以指数部分作为数组下标，将系数记录进数组中即可完成任务。但本题中指数范围过大，直接采取上述方法进行计算将会**导致数组越界**，因此需要考虑其他方法。

注意到两个函数都是以严格升序给出的，因此可以考虑这样一种做法：

1. 用两个变量 i, j 记录当前正在处理 $f(x)$ 的第 i 项和 $g(x)$ 的第 j 项，最初 $i = j = 1$。
2. W比较 $f(x)$ 的第 i 项和 $g(x)$ 的第 j 项的指数：

   a. 若指数部分相同，说明这两项应该合并，系数应相减，输出，然后将 i, j 都向后移动一位；

   b. 若 $f(x)$ 的第 i 项的指数较小，说明只有 $f(x)$ 的第 i 项在当前计算的这一项中出现，输出，并将 i 向后移动一位；

   c. 若 $g(x)$ 的第 j 项的指数较小，说明只有 $g(x)$ 的第 j 项在当前计算的这一项中出现，输出，并将 j 向后移动一位。

当 i, j 将 $f(x), g(x)$ 的每一项都计算之后，最后结果的多项式也被计算出来了。可以发现，通过该方法得到的多项式，其指数部分也是严格递增的。

由于只会移动最多 $m + n$ 次，因此总循环次数不超过 $m + n$ 次，可以在题目要求的时间范围内完成计算。

示例代码

```c
#include <stdio.h>
#include<math.h>
long long a[100006], b[100006], A[100006], B[100006];
long long c[200006], C[200006];
int main() {
    int m, n;
    int t;
    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);
        for (int i = 1; i <= n; i++) {
            scanf("%lld", &a[i]);
        }
        for (int i = 1; i <= n; i++) {
            scanf("%lld", &A[i]);
        }
        scanf("%d", &m);
        for (int i = 1; i <= m; i++) {
            scanf("%lld", &b[i]);
        }
        for (int i = 1; i <= m; i++) {
            scanf("%lld", &B[i]);
        }
```

```
        int i = 1, j = 1, k = 1;
        while (i <= m || j <= n) {
            if (i == m + 1) {
                c[k] = b[j], C[k] = B[j];
                j++;
                k++;
            } else if (j == n + 1) {
                c[k] = a[i], C[k] = A[i];
                i++;
                k++;
            } else if (A[i] < B[j]) {
                c[k] = a[i], C[k] = A[i];
                i++;
                k++;
            } else if (B[j] < A[i]) {
                c[k] = b[j], C[k] = B[j];
                j++;
                k++;
            } else {
                if (a[i] + b[j] != 0) {
                    c[k] = a[i] + b[j], C[k] = B[j];
                }
                i++;
                j++;
                k++;
            }
        }
        printf("%d\n", k - 1);
        for (int i = 1; i <= k - 1; i++) {
            printf("%lld ", c[i]);
        }
        printf("\n");
        for (int i = 1; i <= k - 1; i++) {
            printf("%lld ", C[i]);
        }
        printf("\n");
    }
    return 0;
}
```

H一种简单的语言

难度	考点
6	字符串解析

题目分析

这道题关键的地方是要识别并存下来定义过的变量对应的值，然后输出的时候去查找并输出即可。

存变量可以用两个数组去存，一个是`char names[][]`，一个是`values[]`，`names[i]`表示`i`位置的变量的名字的字符串，`values[i]`表示`i`位置的变量的值。对于多次赋值会覆盖的问题，可以每次存一个变量时，都尾接在数组的末尾，然后查询的时候从末尾往前查询，这样的话第一个查询到的就是变量被赋过的最新的值。

可能用到一些函数：`strcmp strcpy strlen atoi(将字符串类型的数字转为整型，在 <stdlib.h> 库中)...`，当然也可以不用，自己写对应的需要的逻辑。

示例代码

```
#include <stdio.h>
#include <string.h>

char names[5100][7];
int values[5100];
int cnt;
char buf[200];

int isExist; // 表示查找的变量是否存在
int findValOf(char s[], int l1) {
    isExist = 1;
    for (int i = cnt - 1; i >= 0; i--) {
        int l2 = strlen(names[i]);
        if (l1 != l2) continue;
        else {
            int ok = 1;
            for (int j = 0; j < l1; ++j) {
                if (names[i][j] != s[j]) ok = 0;
            }
            if (ok) return values[i];
        }
    }
    isExist = 0; // 表示不存在
    return 0;
}

int main() {
    int line = 1;
    while (scanf("%s", buf) != EOF) {
        int len = strlen(buf);
        if (len == 5 && buf[0] == 'p' && buf[1] == 'r' && buf[2] == 'i' && buf[3]
== 'n' && buf[4] == 't') {
            char n[7];
            scanf("%s", n);
            int val = findValOf(n, strlen(n));
            if (!isExist) printf("line %d: print invalid name \"%s\"!\n", line,
n);
            else printf("%d\n", val);
        } else { // 存下来变量
            char c;
            long long val = 0;
            // 解析出来要被赋的值val
            while ((c = getchar()) == ' ');
            if (c <= '9' && c >= '0') {
                val = c - '0';
```

```
                while ((c = getchar()) <= '9' && c >= '0') val = 10 * val + c -
'0';
            } else if (c == '-') {
                while ((c = getchar()) <= '9' && c >= '0') val = 10 * val + c -
'0';
                val = -val;
            } else {
                // 查找变量名对应的变量
                char nn[7], index = 0;
                nn[index++] = c;
                while (((c = getchar()) <= 'z' && c >= 'a') || (c <= 'Z' && c >=
'A')) nn[index++] = c;
                val = findValOf(nn, index);
            }
            // 存进去新的变量
            strcpy(names[cnt], buf);
            values[cnt] = val;
            cnt++;
        }
        line++;
    }
}
```

I 英语大师

难度	考点
4	字符串、递归

题目分析

n 最多有 10 位，根据英文规则，按 3 位一组进行划分，递归得到每一组的英文。根据每组数字所在的范围，具体操作如下：

- $n < 20$，直接转换成英文；
- $20 \le n < 100$，先将十位转换成英文，然后对个位递归地转换成英文；
- $n \ge 100$，先将百位转换成英文，然后对十位和个位递归地转换成英文。

得到每一组的英文后，需要对每一组加上对应的表示单位的词，然后拼接得到整数 n 的英文。

注意：

- 只有非零的组的英文才会拼接到整数 n 的英文中；
- n 为零时需要特判，直接返回 `zero`。

示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
const char SMALL[20][12] = {"Zero ", "One ", "Two ", "Three ", "Four ", "Five ",
"Six ", "Seven ", "Eight ", "Nine ", "Ten ", "Eleven ", "Twelve ", "Thirteen ",
"Fourteen ", "Fifteen ", "Sixteen ", "Seventeen ", "Eighteen ", "Nineteen "};
const char TEN[10][12] = {"Zero ", "Ten ", "Twenty ", "Thirty ", "Forty ", "Fifty
", "Sixty ", "Seventy ", "Eighty ", "Ninety "};
const char BIG[4][12] = {"Hundred ", "Thousand ", "Million ", "Billion "};
const int VAL[4] = {100, 1000, 1000000, 1000000000};

char res[10086];

void n2word(int n) {
    // 大于等于100的数，从最大的单位开始处理
    if (n >= 100) {
        for (int i = 3; i >= 0; --i) {
            if (n / VAL[i] > 0) {
                n2word(n / VAL[i]);
                strcat(res, BIG[i]);
                n%=VAL[i];
            }
        }
    }
    if (n == 0) {
        return;
    }
    // 小于20的数，直接查找 SMALL 数组
    else if (n < 20) {
        strcat(res, SMALL[n]);
    }
    // 小于100的数，查找 TENS 数组，然后递归处理个位
    else {
        strcat(res, TEN[n / 10]);
        n2word(n % 10);
    }
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        if (n == 0) {
            printf("Zero\n");
        } else {
            memset(res, 0, sizeof(res));
            n2word(n);
            printf("%s\n", res);
        }
    }
    return 0;
}
```

J 朝日捡石头

难度	考点
5	带权二分

题目分析

先考虑没有 m 怎么做。

可以得到一个简单递推：$f_i = max\{f_{i-1}, f_{i-k} + a_i\}$ 。其中 f_i 表示在前 i 个石头中选若干个的答案。

考虑二分一个附加值 x ，让所有 a_i 去加上这个 x ，再做上述的递推，同时记录一下 f_i 取到最值时取了多少个石头，记作 g_i 。

x 越大， g 越大，反之越小。[(1)]

因此，我们不断二分这个 x 直到 g 取到 m 。

有一些细节是， f_i 取到最值时 g_i 可能有不同的取值，相当于切到了凸包的边 [(2)] ，你要明确 g 此时求的是最大还是最小值。

示例代码

```
#include<stdio.h>
#define N 300010
const int inf=1000000007;
int n,m,k,a[N],g[N],A[N];
long long f[N];
int max(int x,int y){return x>y?x:y;}
int check(int d) {
    for(int i=1;i<=n;i++)A[i]=a[i]+d;
    for(int i=1;i<=n;i++) {
        int j=max(0,i-k);
        long long tmp=f[j]+A[i];
        if(tmp>=f[i-1])f[i]=tmp,g[i]=g[j]+1; //注意这里g求的是f取最优下的最大值，所以是
>=
        else f[i]=f[i-1],g[i]=g[i-1];
    }
    return g[n];
}
int main() {
    scanf("%d%d%d",&n,&m,&k);
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    int l=-inf,r=inf;
    while(l<r) {
        int mid=(l+r)>>1;
        if(check(mid)<m)l=mid+1;//如果g取到最大值了还是切不到m，那么二分右侧
        else r=mid;
    }
    check(l);
    printf("%lld",f[n]-1ll*l*m);
}
```

正确性

这里对上面 $(1)(2)$ 进行进一步解释。

关于这部分，请有兴趣的同学先行学习带权二分/wqs二分。

设 $F(m)$ 表示选了 m 个石头的答案，这里简单证明一下这个函数的凸性，也就是对 (1) 的解释。

考虑 $F(m-1)$ 和 $F(m+1)$ 的共 $2m$ 个石头，总可以构造出一种方案，将它们划分成两个大小为 m 的石头集合 A, B 满足距离限制。

所以有 $F(m-1)+F(m+1)=\sum_A a_i+\sum_B a_i\le 2F(m)$，所以 F 是上凸的。

我们二分的 x，其实就是二分一个斜率去切 F 这个凸包，一直切到 m 为止。

思考

如果仅把距离限制改成：相邻的石头之间距离要 $\le k$，还能这么做吗？