徐老师的广度优先搜索

题解

方法一：找规律。形成的正方形可以看成两块间隔的形状。如下图，`1` 的这块有n^2，`.` 的这块有$(n-1)^2$

```
0001000
001.100
01.1.10
001.100
0001000
```

方法二：当然，你也可以考虑用等差数列求和来做。显然上面和下面两部分都是等差数列。不妨把最中间一行划分到上面的部分去，那么有：

1. 上面的部分是$1+3+5+...+(2*n-1)=n^2$
2. 下面的部分是$1+3+5+...+(2*n-3)=(n-1)^2$

同样可以得到答案。

注意使用 $longlong$

标程

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    long long n; cin >> n;
    cout << (n * n + (n - 1) * (n - 1)) << endl;
    return 0;
}
```

徐老师的仓管系统

题解

使用一个布尔类型数组 `check[]` 来表示集合中的元素， `check[i]` 表示元素 i 是否存在于集合中。每次操作从输入中读取指令和元素，然后根据指令进行相应的操作。操作结果通过输出 `YES` 或 `NO` 来实现。

标程

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e6 + 10;
bool check[N];
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        char s[10];
        int x;
        scanf("%s%d", s, &x);
        if (s[0] == 'A') {
            if (check[x]) {
                printf("Error\n");
            } else {
                check[x] = 1;
                printf("Yes\n");
            }
        } else if (s[0] == 'D') {
            if (!check[x]) {
                printf("Error\n");
            } else {
                check[x] = 0;
                printf("Yes\n");
            }
        } else {
            if (check[x]) {
                printf("Yes\n");
            } else {
                printf("No\n");
            }
```

```
        }
    }
    return 0;
}
```

徐老师的求和方案

题解

数据范围很小，直接暴力 dfs 即可

依次遍历每一位数字，考虑两种方案，要么拼到前一位去，要么这个数字自己成为一个新数字的开头

注意判断 0 不能成为新数字的开头即可

标程

```
#include <bits/stdc++.h>
using namespace std;
char s[20];
long long n, ans, mx, mi, ret;

void dfs(int pos, long long lst, long long sum) {
    if(pos == n) {
        sum += lst;
        ans++;
        if (ans == 1){
            mx = mi = sum;
        }
        mx = max(mx, sum);
        mi = min(mi, sum);
        ret += sum;
        return;
    }
    dfs(pos + 1, lst * 10 + s[pos] - '0', sum);
    if (s[pos] != '0'){
        dfs(pos + 1, s[pos] - '0', sum + lst);
```

```
    }
    return;
}

int main() {
    scanf("%s", s);
    n = strlen(s);
    dfs(1, s[0] - '0', 0);
    printf("%lld\n", ans);
    printf("%lld %lld %lld\n", mx, mi, ret);
    return 0;
}
```

徐老师的羊腿高塔

题解

思考最后的形状，显然是一个类似于三角形的递减形态，则可以考虑暴力：

1. 枚举一个高度 H
2. 对于这个高度 H，枚举这个高度出现的位置 i
3. 验证 i 向两边递减判断能否出现这个高度

这样的做法是 $O(n^3)$ 的，只能通过 50 分，考虑如何优化

首先很容易发现，这个高度 H 存在单调性，显然可以第一步枚举高度可以改成二分高度，那么复杂度降低到了 $O(n^2logn)$，能通过 70 分

对于满分的优化，我们需要思考如何让 $check$ 部分的复杂度降低

我们可以发现，对于验证某个高度 H 来说，难点在于并不能直接计算一个三角形需要使用的羊腿再减去现有的羊腿

而这里无法直接计算的原因则是因为可能会有一些位置的羊腿比需求的要高，例如对于下标 3 来说设置高度 5 来说，需要的实际高度为 $3,4,5,4,3$

而如果原本的高度是 $5,5,5,5,5$，则无法直接计算

那么从这里就可以发现，影响计算的其实是对于某个下标 x 来说，需要它的高度是 y，但是它的现有高度已经超过 y 了，则会导致无法计算

但是如果它的高度已经超过 y 了，不就代表它的左侧或者右侧高度如何已经不影响了吗?

如上例子所示，如果对于 len_2 已经是 5 了，则不需要考虑 len_1 的问题即可让 len_3 达到 5

在枚举的过程中，可以计算出 i 最多能够有效影响的位置即为 $i+H-len_i$，在 $[i+H-len_i, i+H-1]$ 这个区间内的位置在成为三角形顶端的时候是不需要考虑 i 之前的

那么这样就可以用 $O(n)$ 的复杂度计算出每个点作为三角形顶端时左半边需要用的羊腿数，然后倒过来再算一次右边即可

标程

```
#include <cstdio>
#include <vector>
using namespace std;
const long long inf = (long long)2e18;
const int N = 1234567;
int h[N], n;
long long m;
vector <int> ev[N];
long long ans[N];
long long sum[N];

int check(int mid){
    for (int i = 0; i < n; i++) ans[i] = 0;
    //正反两次操作，计算左边和右边
    for (int rot = 0; rot < 2; rot++) {
        sum[0] = h[0];
        for (int i = 1; i < n; i++)
            sum[i] = sum[i - 1] + h[i];
        for (int i = 0; i < n; i++)
            ev[i].clear();
        for (int i = 0; i < n; i++) {
            int j = i + mid - h[i];
            if (j < n){
                ev[j].push_back(i);
            }
        }
        int mx = -1;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < (int) ev[i].size(); j++) {
```

```
                mx = max(mx, ev[i][j]);
            }
            if (mx >= 0) {
                int from = mid - 1;
                int to = mid - (i - mx) + 1;
                ans[i] += (to + from) * 1LL * (from - to + 1) / 2;
                ans[i] -= (sum[i - 1] - sum[mx]);
            } else{
                ans[i] += inf;
            }

        }
        for (int i = 0; i < n - i - 1; i++) {
            swap(h[i], h[n - i - 1]);
            swap(ans[i], ans[n - i - 1]);
        }
    }
    for (int i = 0; i < n; i++) {
        ans[i] += mid - h[i];
        if (ans[i] <= m) {
            return true;
            break;
        }
    }
    return false;
}
int main() {
    scanf("%d %lld", &n, &m);
    int low = 0;
    for (int i = 0; i < n; i++) {
        scanf("%d", h + i);
        low = max(low, h[i]);
    }
    int final_ans = 0;
    int high = (int)1e9;
    while (low <= high) {
        int mid = (low + high) >> 1;
        if (check(mid)){
            low = mid + 1;
            final_ans = mid;
        } else {
            high = mid - 1;
        }
    }
    printf("%d\n", final_ans);
```

```
    return 0;
}
```