结果填空：π的近似值

题解

简单模拟。

标程

```
#include <iostream>
using namespace std;
int main() {
    double pi = 0;
    int fg = 1;
    for (int i = 1; i <= 99; i += 2) {
        pi += 1.0 / i * fg;
        fg *= -1;
    }
    pi *= 4;
    printf("%.6lf\n", pi);
    return 0;
}
```

结果填空：实习生的工资

题解

可以一天一天模拟，用 cnt 记录一下当月能出勤的天数。如果到下一个月了，累计当月出勤，然后清空 cnt。注意不要忘记了最后一个二月的出勤。

标程

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

int days[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int main() {
    int y = 2019, m = 2, d = 11, w = 1;
    double sum = 0;
    int cnt = 0;
    while (1) {
        if (y == 2020 && m == 2 && d == 11) {
            sum += min(1.0 * cnt, 21.5);
            break;
        }
        if (w != 6 && w != 7) {
            cnt++;
        }
        d++;
        w++;
        if (w > 7) {
            w = 1;
        }
        if (d > days[m]) {
            sum += min(1.0 * cnt, 21.5);
            cnt = 0;
            d -= days[m];
            m++;
        }
        if (m > 12) {
            m = 1;
            y++;
        }
    }
    cout << sum << endl;
    return 0;
}
```

结果填空：明明的压岁钱

题解

我们按照给的压岁钱数从小到大的顺序暴力枚举即可。最后一个不需要枚举，可以直接计算出来，然后判断是否合法就可以了。

标程

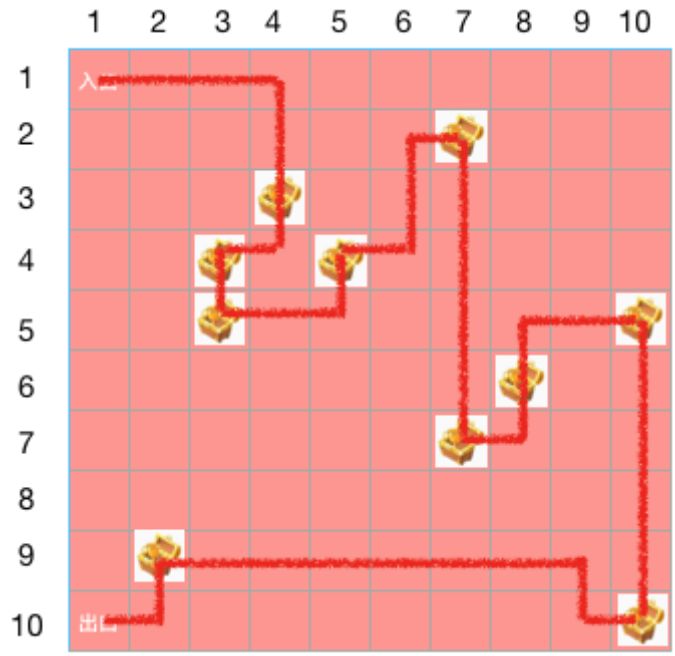
```cpp
#include <iostream>
using namespace std;
int main() {
    int ans = 0;
    for (int a2 = 2; a2 <= 20; a2++) {
        for (int a3 = 2; a3 <= 20; a3++) {
            for (int a5 = max(a2, a3) + 1; a5 <= 20; a5++) {
                for (int a6 = a5 + 1; a6 <= 20; a6++) {
                    for (int a7 = a5 + 1; a7 <= 20; a7++) {
                        for (int a1 = 2; a1 <= 20; a1++) {
                            for (int a4 = a1 + 1; a4 <= 20; a4++) {
                                int s = a1 + a2 + a3 + a4 + a5 + a6 + a7;
                                if (s < 100 && 100 - s >= 2 && 100 - s <= 2
0) {
                                    ++ans;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    cout << ans << endl;
    return 0;
}
```

结果填空：故宫元宵节寻宝

题解

我们可以直接用 dfs 搜索下一次去取哪个宝藏。由于地图上没有障碍，两个点之间的最短路就是 X 方向的坐标差加上 Y 方向的坐标差。

一种方法



标程

```cpp
#include <iostream>
using namespace std;
int x[11], y[11];
int ans = 1e9;
bool vis[11];
int n;
void dfs(int id, int cnt, int cost) {
    if (cnt == n) {
        ans = min(ans, cost + abs(x[id] - 10) + abs(y[id] - 1));
        return;
    }
    if (cost >= ans) {
        return;
    }
    vis[id] = 1;
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            dfs(i, cnt + 1, cost + abs(x[id] - x[i]) + abs(y[id] - y[i]));
        }
    }
    vis[id] = 0;
}
int main() {
    cin >> n;
    x[0] = y[0] = 1;
    for (int i = 1; i <= n; i++) {
        cin >> x[i] >> y[i];
    }
    dfs(0, 0, 0);
    cout << ans << endl;
    return 0;
}
```

代码填空：Excel列编号

题解

我们发现长度为 1 的编号一共有 26 个，长度为 2 的编号一共有 26^2，长度为 l 的编号一共有 26^l 个，所以我们可以先暴力找到 n 对应的编号的长度假设为 l。然后 n 减去前面长度小于的 l 的所有的编号个数就是在长度为 l 的所有编号中 n 的编号。

接下来我们递归的寻找 l 个位置具体每一位置是什么。我们先找到最前面的字母，假设为 `'A' + x`，那么去掉首字母后后面 $l-1$ 个字母组成的编号在长度为 $l-1$ 的编号中的排名为 $n - x \cdot 26^{l-1}$，然后递归找后面 $l-1$ 个字母即可。

答案

```
n - cnt
```

程序设计：楼房的距离

题解

对于 30% 的数据：由于 w 等于 1，答案就是 $|n-m|$。

对于 60% 的数据：我们可以模拟一遍，找到每个点的具体坐标，然后他们之间最短距离就是 x 坐标的距离加上 y 坐标的距离。

对于 100% 的数据：实际上可以直接计算，如果我们是按照

```
1  2  3  4  5  6
7  8  9  10 11 12
13 14 15 .....
```

这样排列的，那么 n 的坐标为 $(\frac{(n-1)}{w},(n-1)\%w)$。

现在按照原题的排列，我们发现只有奇数行(下标从 0 开始)是反过来的，那么如果是奇数行，只需要把 y 翻转一下就可以了。

标程

```cpp
#include <iostream>
using namespace std;
typedef long long LL;
struct Point {
    LL x, y;
};
LL w;
Point getpoint(LL id) {
    id--;
    Point res;
    res.x = id / w;
    res.y = id % w;
    if (res.x % 2 == 1) {
        res.y = w - res.y - 1;
    }
    return res;
}
int main() {
    LL n, m;
    cin >> w >> n >> m;
    Point a1 = getpoint(n), a2 = getpoint(m);
    cout << abs(a1.x - a2.x) + abs(a1.y - a2.y) << endl;
    return 0;
}
```

程序设计：版本号控制

题解

用一个结构体套用 `vector` 保存每个版本号，然后用 `sort` 排序。

标程

```cpp
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>
#include <assert.h>
using namespace std;
struct node {
    string version;
    vector<int> v;
} G[110];
bool cmp(const node &a, const node &b) {
    for (int i = 0; ; i++) {
        if (i == (int)a.v.size()) {
            return 1;
        }
        if (i == (int)b.v.size()) {
            return 0;
        }
        if (a.v[i] < b.v[i]) {
            return 1;
        }
        if (a.v[i] > b.v[i]) {
            return 0;
        }
    }
}
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        string ss;
        cin >> ss;
        G[i].version = ss;
        int cnt = 0;
        for (int j = 0; j < ss.size(); j++) {
            if (ss[j] == '.') {
                G[i].v.push_back(cnt);
                cnt = 0;
            } else {
                cnt = cnt * 10 + ss[j] - '0';
            }
        }
        G[i].v.push_back(cnt);
    }
    sort(G, G + n, cmp);
    for (int i = 0; i < n; i++) {
        cout << G[i].version << endl;
    }
    return 0;
}
```

程序设计：排列还原

题解

这是一个非常明显的 bfs 的题目。我们可以直接一个字符串来记录每个状态，转移的时候直接枚举翻转位置。

标程

```cpp
#include <iostream>
#include <string>
#include <map>
#include <queue>
using namespace std;
map<string, int> mp;
int n;
int bfs(string st, string en) {
    if (st == en) {
        return 0;
    }
    queue<string> q;
    q.push(st);
    mp[st] = 0;
    while (!q.empty()) {
        string s = q.front();
        q.pop();
        for (int i = 1; i < n; i++) {
            string tt = s;
            for (int j = 0, k = i; j < k; j++, k--) {
                swap(tt[j], tt[k]);
            }
            if (tt == en) {
                return mp[s] + 1;
            }
            if (!mp.count(tt)) {
                mp[tt] = mp[s] + 1;
                q.push(tt);
            }
        }
    }
}
int main() {
    string s = "", e = "";
    cin >> n;
    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;
        s += to_string(x);
        e += to_string(i);
    }
    cout << bfs(s, e) << endl;
    return 0;
}
```

程序设计：数列乘积

题解

对于 30% 的数据，可以直接暴力求解。

对于另外 20% 的数据，我们发现答案就是 $2^n \bmod (10^9+7)$ ，用一个快速幂求解就可以。

对于 100% 的数据，把公式展开后发现

$$\begin{aligned}\prod_{i=1}^{n}(a_i+1) &= (a_1+1)(a_1^2+1)\cdots(a_1^{2^{n-1}}+1)\\ &= 1+a_1+a_1^2+a_1^3+a_1^4+\cdots+a_1^{2^n-1}\\ &= \frac{a_1^{2^n}-1}{a_1-1}\end{aligned}$$

可以用快速幂加上逆元求解。其中根据欧拉定理

$$a_1^{2^n} \bmod (10^9+7) = a_1^{2^n \bmod (10^9+6)} \bmod (10^9+7)$$

标程

```cpp
#include <iostream>
using namespace std;
const int mod = 1000000007;
typedef long long LL;
LL pow_mod(LL x, LL y, int mod) {
    LL res = 1, temp = x % mod;
    for ( ; y; y /= 2) {
        if (y & 1) {
            res = res * temp % mod;
        }
        temp = temp * temp % mod;
    }
    return res;
}
int main() {
    LL n, a;
    cin >> n >> a;
    if (a == 1) {
        cout << pow_mod(2, n, mod) << endl;
    } else {
        LL p = pow_mod(2, n, mod - 1);
        LL res = pow_mod(a, p, mod) - 1;
        if (res < 0) {
            res += mod;
        }
        res *= pow_mod(a - 1, mod - 2, mod);
        res %= mod;
        cout << res << endl;
    }
    return 0;
}
```

程序设计：收集雨水

题解

对于 20% 的数据：直接暴力模拟即可。

对于另外 20% 的数据：由于没有负数，可以直接求出前缀和，然后通过前缀和求出区间和就可以了。

对于 80% 的数据：我们先预处理出来前缀和 sum_i。然后预处理出来一个 rid_i 表示如果在第 i 天开始搜集雨水，那么在第 rid_i 天的时候，**第一次** 出现了空杯（没有水了）。也就是说 $[i, rid_i]$ 这几天都白搜集了，没有任何收获。这个值实际上就是求 $[i, n]$ 上最小的 x 满足

$sum[x] \le sum[i-1]$ 的位置，我们可以通过用线段树维护区间最小值，然后二分答案的方法来预处理。这部分时间复杂度为 $\mathcal{O}(n \log^2 n)$。

预处理出来 rid 以后，我们把所有查询离线后先按照 l 从小到大排序，然后枚举所有以 i 为左端点的区间，如果右端点 $r \le rid_i$，那么答案就是 $sum[r] - sum[i-1]$。否则对于这个查询，从 i 开始和 $i+1$ 开始的效果是一样的，所以我们可以把这个查询放到后面处理。所以我们就用另外一个集合按照 r 从小到大的顺序维护查询的区间，只要集合中满足 $r \le rid_i$ 的查询都可以得到答案了。由于每个查询最多进一次集合，出一次集合，这部分的时间复杂度为 $\mathcal{O}(q \log q)$。

由于 n 和 q 是一个级别，所以总的时间复杂度主要在预处理 $\mathcal{O}(n \log^2 n)$。

对于 100% 的数据：我们可以用线段树直接预处理出 rid，不需要二分。具体方法见标程。最后总的时间复杂度为 $\mathcal{O}(n \log n + q \log q)$。

标程

```cpp
#include <iostream>
#include <algorithm>
#include <set>
using namespace std;
const int maxn = 500010;
const int inf = 0x3f3f3f3f;
typedef long long LL;
LL minv[4 * maxn];
LL a[maxn], sum[maxn];
void build(int id, int l, int r) {
    if (l == r) {
        minv[id] = sum[l];
        return;
    }
    int mid = (l + r) >> 1;
    build(id << 1, l, mid);
    build(id << 1 | 1, mid + 1, r);
    minv[id] = min(minv[id << 1], minv[id << 1 | 1]);
}
int query(int id, int l, int r, int x, int y, LL v) {
    if (l == r) {
        return l;
    }
    int mid = (l + r) >> 1;
    if (mid >= x) {
        if (minv[id << 1] <= v) {
            int t = query(id << 1, l, mid, x, y, v);
            if (t != inf) {
                return t;
            }
        }
    }
    if (mid < y) {
        if (minv[id << 1 | 1] <= v) {
            int t = query(id << 1 | 1, mid + 1, r, x, y, v);
            if (t != inf) {
                return t;
            }
        }
    }
    return inf;
}
int rid[maxn];
struct Que {
    int id, l, r;
} Q[maxn];
bool cmpl(Que q1, Que q2) {
    if (q1.l == q2.l) {
        return q1.id < q2.id;
    }
    return q1.l < q2.l;
```

```
}
class cmpr {
    public:
    bool operator() (const Que& q1, const Que& q2) const {
        if (q1.r == q2.r) {
            return q1.id < q2.id;
        }
        return q1.r < q2.r;
    }
};
LL ans[maxn];
set<Que, cmpr> s;
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
        sum[i] = sum[i - 1] + a[i];
    }
    build(1, 1, n);
    for (int i = 1; i <= n; i++) {
        rid[i] = query(1, 1, n, i, n, sum[i - 1]);
    }
    int q;
    scanf("%d", &q);
    for (int i = 0; i < q; i++) {
        scanf("%d %d", &Q[i].l, &Q[i].r);
        Q[i].id = i;
    }
    sort(Q, Q + q, cmpl);
    int p = 0;
    for (int i = 1; i <= n; i++) {
        while (p < q && Q[p].l == i) {
            s.insert(Q[p]);
            p++;
        }
        while (s.size() >= 1) {
            Que x = *s.begin();
            if (x.r <= rid[i]) {
                ans[x.id] = max(0LL, sum[x.r] - sum[i - 1]);
                s.erase(x);
            } else {
                break;
            }
        }
    }
    for (int i = 0; i < q; i++) {
        printf("%lld\n", ans[i]);
    }
    return 0;
}
```