Week3、week4_心得体会

1.进行了一次简单的 csp 模拟测试，很喜欢那种感觉，虽然制作出来前两道题（居然完美过了没有出问题哈哈），但是也比较满意，因为最后一个实在不会。

2.虽然 cin 很爽，但 **scanf 真香！**虽然 cin 简单帅，但 scanf 快啊，大批量数据不想 TLE 只能选 scanf。scanf 是格式化输入；cin 是输入流，效率稍低，但书写简便。格式化输入效率比较高，但是写代码麻烦。流输入操作效率稍低，但书写简便。cout 之所以效率低，是先把要输出的东西存入缓冲区，再输出，导致效率降低。cin 也是类似

https://zh.cppreference.com/w/cpp/io/cin

https://zh.cppreference.com/w/cpp/io/c/fscanf

3.注意题目给出的数据量和数据范围，合理估计要开的数组大小，太大了 MLE（可能性很小），太小了 RE。

4.二分法真的很省时间。排序一个 sort 就行了，最多重载个”<”。

5. 二分算法常常用来对暴力枚举算法进行优化以及求解最大值最小 / 最小值最大问题，最重要的是选取适当的上下界以及二分模板中对当前 mid 大小的判断。

6.查二分法相关资料时发现当对二分算法的边界难以把握时可以考虑使用 STL 中的 upper_bound 与 lower_bound 进行替代。

https://zh.cppreference.com/w/cpp/algorithm/upper_bound

https://zh.cppreference.com/w/cpp/algorithm/lower_bound

7.由于 DFS 本身不撞南墙不回头的特点，很多情况下要进行“剪枝”，也就是依据适当的条件及时 return 进行回溯,节省时间和空间。还有就是一般都要配合标记数组使用。

8.代码变量命名尽量易懂，不能图简单，不然过段时间回来就不知道这个变量用来干嘛的了，不用怕麻烦，代码补全挺好用的。

9.贪心算法要找到合适的贪心指标，一般都是类似相同情况价值选最高这样的，搭配优先级队列更好（不熟练，也就没用到）。

201800130125_孟龙雨

Week3

A-选数问题 •

Given nn positive numbers, ZJM can select exactly KK of them that sums to SS. Now ZJM wonders how many ways to get it!

Input

The first line, an integer T<=100T<=100, indicates the number of test cases. For each case, there are two lines. The first line, three integers indicate nn, KK and SS. The second line, nn integers indicate the positive numbers.

Output

For each case, an integer indicate the answer in a independent line.

Example

Input

```
1
10 3 10
1 2 3 4 5 6 7 8 9 10
```

Output

```
4
```

Note

Remember that k<=n<=16k<=n<=16 and all numbers can be stored in 32-bit integer

思路解释

题目总体意思就是从 n 个数里面选出 k 个和为 s 的数，然后输出 k。

想法是 dfs 深度优先搜索寻找合适的组合，当选中的数的个数达到 k 的时候但是由于 dfs 本身不撞南墙不回头的特点，需要及时回溯，也就是进行剪枝。

当选中的数和为 s 而且数字个数为 k 时（由于每选一次数都有 k--，此时 k=0）返回 1，代表产生一种组合。

当选中的数和不为 s 而数字个数为 k 时，也就是 k 达到要求但是 s 不满足，及时剪枝 return 0。

当数组遍历完毕而没有达到要求时，return 0.

当以上三种情况都不满足时，正常搜索合适的数字，当前数字加入组合，开始寻找下一个数字。

代码

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

int T, k, s, n, t;
vector<int> arr;
```

```
int solution(vector<int> a, int start, int k, int sum) {
    int length = a.size();
    if (k == 0 && sum == 0) return 1;
    if (k == 0 && sum != 0) return 0;
    if (start >= length) return 0;
    return solution(a, start + 1, k - 1, sum - a[start]) +
           solution(a, start + 1, k, sum);

int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> T >> k >> s;
        for (int i = 0; i < T; i++) {
            cin >> t;
            arr.push_back(t);
        }
        int ans = solution(arr, 0, k, s);
        cout << ans << endl;
        arr.clear();
    }

    return 0;
}
```

B - 区间选点（编译器选 GNU G++）Gym - 270437B

数轴上有 n 个闭区间 [a_i, b_i]。取尽量少的点，使得每个区间内都至少有一个点（不同区间内含的点可以是同一个）

Input
第一行 1 个整数 N（N<=100）
第 2~N+1 行，每行两个整数 a,b（a,b<=100）
Output
一个整数，代表选点的数目
Examples
Input
2
1 5
4 6
Output
1
Input
3
1 3
2 5

4 6
Output
2

思路

将区间存储为结构体，按照右端点进行升序排序右端点（右端点相同时按照左端点降序排序）。从一开始，令第一个区间右端点为一个待选点，向后遍历区间集合，直到某一个区间左端点大于待选点，也就是区间之间产生间断，此时当前待选点得到了最多的区间覆盖，更新待选点为新区间的右端点，继续循环，直到遍历完所有区间。

当区间按照上面的规则排列完毕时可以发现，选点越靠右，落到的区间越多，说明该贪心方法为最优解

代码

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

struct seg {
    int a;
    int b;
    bool operator<(const seg& S) const {
        return b < S.b || b == S.b && a > S.a;
    }
};
seg set[101];
int main() {
    int i, n, count;
    while (cin >> n) {
        count = 1;
        for (i = 0; i < n; i++) {
            cin >> set[i].a;
            cin >> set[i].b;
        }

        sort(set, set + n);  //按 b 从小到大排序

        int newEnd = set[0].b;
        // cout << set[0].a << "," << set[0].b << "  ";
        for (i = 1; i < n; i++) {
            if (newEnd < set[i].a) {
                count++;
                newEnd = set[i].b;
                // cout << set[i].a << "," << set[i].b << "  ";
            }
```

```
        }
        cout << count << endl;
    }
    return 0;
}
```

C - 区间覆盖（不支持 C++11） POJ - 2376

描述

数轴上有 n (1<=n<=25000)个闭区间 [ai, bi]，选择尽量少的区间覆盖一条指定线段 [1, t]（ 1<=t<=1,000,000）。

覆盖整点，即(1,2)+(3,4)可以覆盖(1,4)。

不可能办到输出-1

输入

第一行：N 和 T

第二行至 N+1 行：每一行一个闭区间。

输出

选择的区间的数目，不可能办到输出-1

样例输入

```
3 10
1 7
3 6
6 10
```

样例输出

```
2
```

提示

这道题输入数据很多，请用 scanf 而不是 cin

思路解释

将区间存储为结构体，按照左端点进行升序排序，向后遍历区间，寻找区间使得相邻选中的区间满足左区间的右端点和右区间的左端点之差小于等于 1（由于是证书覆盖，所以类似（1,2）、（2,3）也可以满足），也就是“set[i].a <= position + 1 ” 的循环条件。最终遍历完毕之后如果选中最右边区间没能覆盖 T 则失败，否则可以覆盖，输出区间数。

如果没有中间的判断步骤，会产生新选中的区间的右端点不够大的情况，也就是存在区间比这个区间覆盖的范围更靠大，后面也可能要消耗更多区间来填补。

cin 居然也行 2333

代码

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
```

```cpp
struct seg {
    int a, b;
    bool operator<(const seg &s) const {
        if (b != s.a) return a < s.a;
    }
};
seg set[30000];
int main() {
    int N, T;
    while (cin >> N >> T) {
        for (int i = 1; i <= N; i++) {
            cin >> set[i].a >> set[i].b;
        }
        sort(set + 1, set + N + 1);
        int position = 0, m = 0,ans = 0;
        for (int i = 1; i <= N; i++) {
            int count = 0;
            while (set[i].a <= position + 1 ) {
                if (set[i].b > m) {
                    count++;
                    m = set[i].b;
                }
                i++;
            }
            i--;
            position = m;
            if (count > 0)
                ans++;
            else
                break;
        }
        if (position < T)
            cout << "-1\n";
        else
            cout << ans << endl;
    }

    return 0;
}
```

Week4

A - DDL 的恐惧 HDU - 1789

ZJM 有 n 个作业，每个作业都有自己的 DDL，如果 ZJM 没有在 DDL 前做完这个作业，那么老师会扣掉这个作业的全部平时分。

所以 ZJM 想知道如何安排做作业的顺序，才能尽可能少扣一点分。

请你帮帮他吧！

Input

输入包含 T 个测试用例。输入的第一行是单个整数 T，为测试用例的数量。

每个测试用例以一个正整数 N 开头(1<=N<=1000)，表示作业的数量。

然后两行。第一行包含 N 个整数，表示 DDL，下一行包含 N 个整数，表示扣的分。

Output

对于每个测试用例，您应该输出最小的总降低分数，每个测试用例一行。

Sample Input

```
3
3
3 3 3
10 5 1
3
1 3 1
6 2 3
7
1 4 6 4 2 4 3
3 2 1 7 6 5 4
```

Sample Output

```
0
3
5
```

Hint

上方有三组样例。

对于第一组样例，有三个作业它们的 DDL 均为第三天，ZJM 每天做一个正好在 DDL 前全部做完，所以没有扣分，输出 0。

对于第二组样例，有三个作业，它们的 DDL 分别为第一天，第三天、第一天。ZJM 在第一天做了第一个作业，第二天做了第二个作业，共扣了 3 分，输出 3。

思路解释

将作业存储为结构体，并按照扣得分数降序排序。假设所有作业都没做，产生一个最多能扣的分数 ans，从第一个作业（分值最大的作业）开始向后，在这个作业 ddl 之前的某空闲天（尽量靠后）做了它，把分从 ans 扣掉 。最后所有作业遍历过后的 ans 就是要扣的分数。

代码

```
#include "algorithm"
#include "cstring"
```

```cpp
#include "iostream"
using namespace std;
struct homeWork {
    int ddl;
    int score;
    bool operator<(homeWork z) {
        if (score > z.score)
            return true;
        else
            return false;
    }
} work[1001];
bool done[1001];
int n, num;
int DDL[1001], mark[1001];
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) {
        memset(done, false, sizeof(done));
        cin >> num;
        int ans = 0;
        for (int j = 0; j < num; j++) {
            cin >> work[j].ddl;
        }
        for (int k = 0; k < num; k++) {
            cin >> work[k].score;
            ans += work[k].score;
        }
        sort(work, work + num);
        for (int j = 0; j < num; j++) {
            for (int p = work[j].ddl; p >= 1; p--) {
                if (!done[p]) {
                    done[p] = true;
                    //可以做，要扣的分减掉
                    ans -= work[j].score;
                    break;
                }
            }
        }
        cout << ans << endl;
    }
    //system("pause");
    return 0;
}
```

B - 四个数列 POJ - 2785

ZJM 有四个数列 A,B,C,D，每个数列都有 n 个数字。ZJM 从每个数列中各取出一个数，他想知道有多少种方案使得 4 个数的和为 0。

当一个数列中有多个相同的数字的时候，把它们当做不同的数对待。

请你帮帮他吧！

Input

第一行：n（代表数列中数字的个数） **（1≤n≤4000）**

接下来的 n 行中，第 i 行有四个数字，分别表示数列 A,B,C,D 中的第 i 个数字（数字不超过 2 的 28 次方）

Output

输出不同组合的个数。

Sample Input

6

-45 22 42 -16

-41 -27 56 30

-36 53 -37 77

-36 30 -75 -46

26 -38 -10 62

-32 -54 -6 45

Sample Output

5

Hint

样例解释：(-45, -27, 42, 30), (26, 30, -10, -46), (-32, 22, 56, -46),(-32, 30, -75, 77), (-32, -54, 56, 30).

思路解释

四个数列两两一组，产生两组所有可能的两数和，然后在这两个数组中寻找互为相反数（和为零）的组合个数。大体思路很明显，一开始打算暴力算，果然超时，要排序其中一组数据，遍历另一个无序数组，从有序数组中二分找出合适的数并统计个数。

代码

```
#include "algorithm"
#include "iostream"
using namespace std;
int n;
int ans = 0;
int a1[4001], a2[4001], a3[4001], a4[4001];
int sum1[4001 * 4001], sum2[4001 * 4001];
int main() {
    while (cin >> n) {
        for (int i = 0; i < n; i++) {
            cin >> a1[i] >> a2[i] >> a3[i] >> a4[i];
        }
        /*for (int i = 0; i < 4; i++) {
```

```
            for (int j = 0; j < n; j++) {
                cout << a[i][j] << " ";
            }
            system("pause");
            cout << endl;
        }*/
        int count = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                sum1[count] = a1[i] + a2[j];
                sum2[count] = a3[i] + a4[j];
                // cout<<sum1[count]<<" "<<sum2[count]<<endl;
                count++;
            }
        }
        sort(sum1, sum1 + n * n);
        int l, r;
        for (int i = 0; i < count; i++) {
            l = 0;
            r = count - 1;
            while (l <= r) {
                int mid = (l + r) >> 1;
                if (sum1[mid] + sum2[i] >= 0)
                    r = mid - 1;
                else
                    l = mid + 1;
                while (sum1[l] + sum2[i] == 0 &&
                       l < count) {  //相同的数字出现多次，加起来
                    ans++;
                    l++;
                }
            }
        }
        cout << ans << endl;
    }

    return 0;
}
```

C - TT 的神秘礼物 POJ - 3579

TT 是一位重度爱猫人士，每日沉溺于 B 站上的猫咪频道。

有一天，TT 的好友 ZJM 决定交给 TT 一个难题，如果 TT 能够解决这个难题，ZJM 就会买一只可爱猫咪送给 TT。

任务内容是，给定一个 N 个数的数组 cat[i]，并用这个数组生成一个新数组 ans[i]。新数组定义为对于任意的 i，j 且 i != j，均有 ans[] = abs(cat[i] - cat[j])，1 <= i < j <= N。试求出这个新数组的中位数，中位数即为排序之后 (len+1)/2 位置对应的数字，’/’ 为下取整。

TT 非常想得到那只可爱的猫咪，你能帮帮他吗？

Input

多组输入，每次输入一个 N，表示有 N 个数，之后输入一个长度为 N 的序列 cat， cat[i] <= 1e9 ， 3 <= n <= 1e5

Output

输出新数组 ans 的中位数

Sample Input

4

1 3 2 4

3

1 10 2

Sample Output

1

8

思路解释

这题数据量很大，必须要用 scanf 读入而不是 cin，否则一定会超时。

开始打算暴力枚举算出 ans 数组，复杂度 O(N^2)，果然最后 TLE

所以要开始考虑省时间:

1.绝对值，把数排序，然后依次用大数减小数，可以去掉绝对值

2.要求数列 ans 的中位数，也就是计算每一个 x=cat[j] -cat[i]的名次，然后根据名次进行二分，如果算出的名次比中位数实际名次小，说明 x 小于中位数，区间右移（`left = mid + 1;`）；反之左移（`right = mid - 1;`）。X 的名次实际上就是在 i 确定的情况下满足条件的 j 的个数，在每次遍历 i 时借助一个 fun 函数二分找到数组 a（cat）中 a[j]<=[i]+x 的最后一个位置，并返回，那么 x 值的名次即为 j-i 。最后遍历完成也就找到了中位数。

代码

```
#include <stdio.h>
#include <algorithm>
#include <iostream>
using namespace std;

int fun(int x, int n, int *a) {
    int left = 0;
    int right = n - 1;
    int ans = -1;
    while (left <= right) {
```

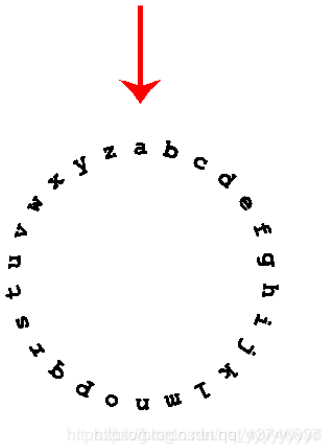
```
        int mid = (left + right) >> 1;
        if (a[mid] <= x) {
            ans = mid;
            left = mid + 1;
        } else
            right = mid - 1;
    }
    return ans;
}
int main() {
    int n;
    while (cin >> n) {
        int *cat = new int[n];
        for (int i = 0; i < n; i++) scanf("%d", &cat[i]);
        sort(cat, cat + n);

        int len = n * (n - 1) / 2;  //中位数应该在的位置
        int rMid = (len + 1) / 2;
        int left = 0;
        int right = cat[n - 1] - cat[0];
        int re;
        while (left <= right) {
            int num = 0;
            int mid = (left + right) / 2;
            for (int i = 0; i < n; i++) {
                int ans = fun(cat[i] + mid, n, cat);
                if (ans != -1) {
                    num += (ans - i);
                }
            }
            if (num >= rMid) {
                re = mid;
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        cout << re << endl;
    }
    return 0;
}
```

Week4_csp-M1

A 咕咕东的奇遇

咕咕东是个贪玩的孩子，有一天，他从上古遗迹中得到了一个神奇的圆环。这个圆环由字母表组成首尾相接的环，环上有一个指针，最初指向字母 a。咕咕东每次可以顺时针或者逆时针旋转一格。例如，a 顺时针旋转到 z，逆时针旋转到 b。咕咕东手里有一个字符串，但是他太笨了，所以他来请求你的帮助，问最少需要转多少次。



输入：
输入只有一行，是一个字符串。

输出：
输出最少要转的次数。

样例：
输入：zeus
输出：18

思路解释：

每两个字母之间的距离事实上也就是它们 ASCII 码之差，根据题意当它们之间距离 x 大于 13 时实际距离就是（26-x），所以计算距离时直接将字符相减再加上距离为 13 以内的判断条件即可，然后遍历字符串即可算得总距离（转动次数）。

代码：

```
#include "iostream"
#include "cmath"
#include "cstring"
using namespace std;
string s;
string a="a";
string str;
int main() {
    while (cin >> s)
    {
        str=a+s;
```

```
    int sum = 0;
    int length = str.size();
    for (int i = 1; i < length; i++) {
        int t1 = abs(str[i] - str[i - 1]);
        if (t1 < 13) {
            sum += t1;
        } else {
            sum += 26 - t1;
        }
    }
    cout<<sum<<endl;
    }
}
```

B 咕咕东想吃饭

1. 题目大意

咕咕东考试周开始了，考试周一共有 n 天。他决定每天都吃生煎，咕咕东每天需要买 ai 个生煎。但是生煎店为了刺激消费，只有两种购买方式：①在某一天一次性买两个生煎。②今天买一个生煎，同时为明天买一个生煎，店家会给一个券，第二天用券来拿。没有其余的购买方式，这两种购买方式可以用无数次，但是咕咕东是个节俭的好孩子，他训练结束就走了，不允许训练结束时手里有券。咕咕东非常有钱，你不需要担心咕咕东没钱，但是咕咕东太笨了，他想问你他能否在考试周每天都能恰好买 ai 个生煎。

输入：

输入两行，第一行输入一个正整数 n(1<=n<=100000)表示考试周的天数。

第二行有 n 个数，第 i 个数 ai（0<=ai<=10000）表示第 i 天咕咕东要买的生煎的数量。

输出：

如果可以满足咕咕东奇怪的要求，输出"YES"，如果不能满足，输出“NO”。（输出不带引号）

样例：

输入：4

1 2 1 2

输出：YES

思路解释：

一开始有点无从下手，后来仔细读题发现，题目里的人不止有钱，还有个四次元胃，也就是买生煎数量不受限制。

每天进行一次选择，首选用券购买，券不够用就把券用完，判断剩下的要用钱买的生煎个数的奇偶性，偶数就直接卖方案一买完，奇数就买一个方案二，券数+1，剩下用方案一，以此类推，判断 n 天过后剩余的券数是否为 0 即可。

代码：

```
#include "iostream"
using namespace std;
int dayNum;
```

```
int n ;
int a[100001];
int main() {
    while (cin >> dayNum) {
        n=0;
        for (int i = 0; i < dayNum; i++) {
            cin >> a[i];
        }
        for (int i = 0; i < dayNum; i++) {
            if (a[i] <= n) {
                n -= a[i];
            } else {
                a[i] -= n;
                n = 0;
                if (a[i] % 2)
                {
                    n++;
                }
            }
        }
        if (n > 0) {
            cout << "NO" << endl;
        } else {
            cout << "YES" << endl;
        }
    }
}
```

C 可怕的宇宙射线

众所周知，瑞神已经达到了 CS 本科生的天花板，但殊不知天外有天，人外有苟。在浩瀚的宇宙中，存在着一种叫做苟狗的生物，这种生物天生就能达到人类研究生的知识水平，并且天生擅长 CSP，甚至有全国第一的水平！但最可怕的是，它可以发出宇宙射线！宇宙射线可以摧毁人的智商，进行降智打击！

宇宙射线会在无限的二维平面上传播（可以看做一个二维网格图），初始方向默认向上。宇宙射线会在发射出一段距离后分裂，向该方向的左右 45° 方向分裂出两条宇宙射线，同时威力不变！宇宙射线会分裂 n 次，每次分裂后会在分裂方向前进 ai 个单位长度。

现在瑞神要带着他的小弟们挑战苟狗，但是瑞神不想让自己的智商降到普通本科生那么菜的水平，所以瑞神来请求你帮他计算出共有多少个位置会被"降智打击"。

输入：

输入第一行包含一个正整数 n(n<=30) ，表示宇宙射线会分裂 n 次。
第二行包含 n 个正整数 a1,a2…an，第 i 个数 ai 表示第 i 次分裂的宇宙射线会在它原方向上继续走多少个单位长度。

输出：
输出一个数 ans，表示有多少个位置会被降智打击。

样例：

输入：4
4 2 2 3
输出：39

思路解释：

主要思想是 dfs，根据数据范围，设定了一个二维平面 map，一个四维数组 visited 储存平面各点状态，dx、dy 为八个方向，数组 a 为每次分裂长度，从平面中心点开始，向指定方向移动，最后一次分裂完成或者当前点已被辐射，则直接 return；否则当前坐标对应四维数组进行标记，然后从 a 中取出当前次数之前每次辐射长度，标记 map 各个点并计数，已标记则跳过，然后进入下一次分裂，分裂次数+1，方向分别为原方向相邻的两个方向 dx[d+1],dy[d+7]，直到最后一次分裂完成，输出最终 map 中的标记点数。

代码：

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

int n, a[101], ans;  //不能放在 fun 函数后面
int dx[8] = {0, 1, 1, 1, 0, -1, -1, -1};
int dy[8] = {1, 1, 0, -1, -1, -1, 0, 1};
bool map[500][500], visited[500][500][101][8];
/*x,y 表示当前位置，curTime 是当前分裂次数，direction 是分裂的方向。*/
void fun(int x, int y, int curTime, int dirction) {
    if (curTime > n || visited[x][y][curTime][dirction]) return;
    visited[x][y][curTime][dirction] = ture;
    for (int i = 1; i <= a[curTime]; i++) {
        x += dx[dirction];
        y += dy[dirction];
        if (!map[x][y]) {
            map[x][y] = ture;
            ans++;
        }
    }
    fun(x, y, curTime + 1, (dirction + 7) % 8);
    fun(x, y, curTime + 1, (dirction + 1) % 8);
}

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

```
        fun(250, 250, 1, 0);
        cout << ans << endl;
    }

    return 0;
}
```