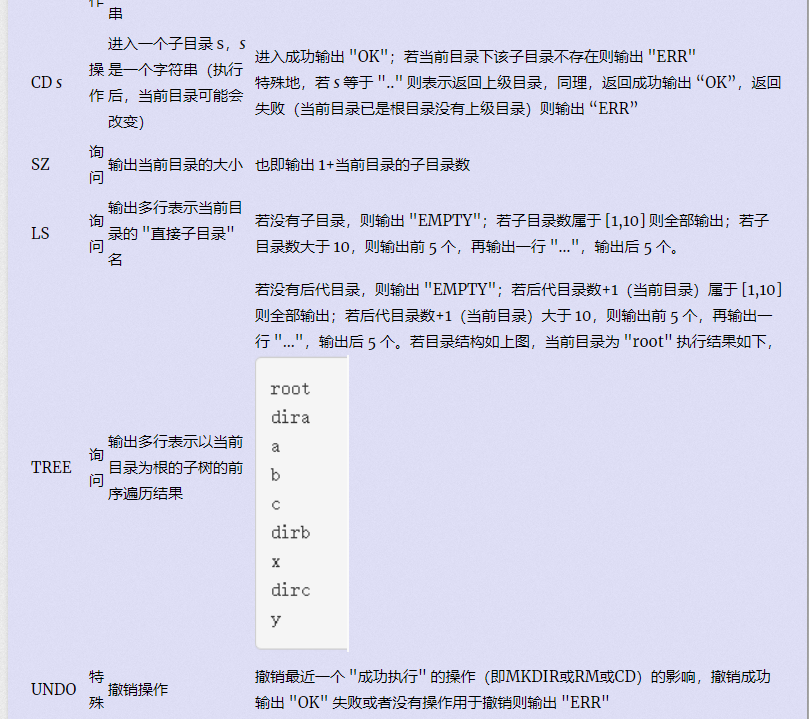
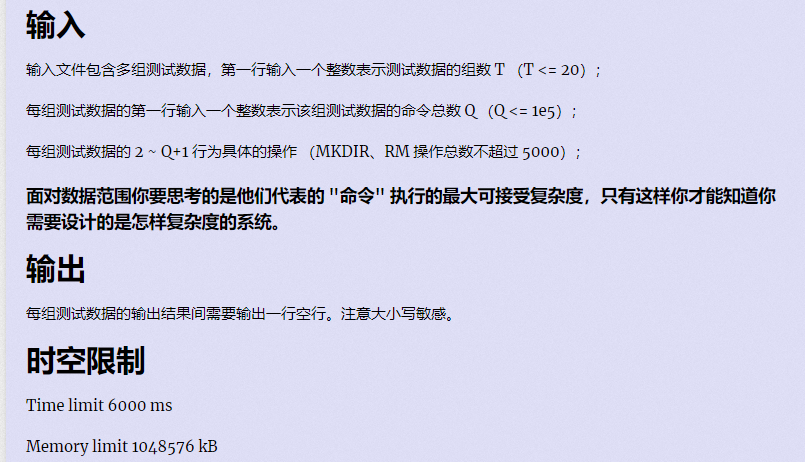
# week9

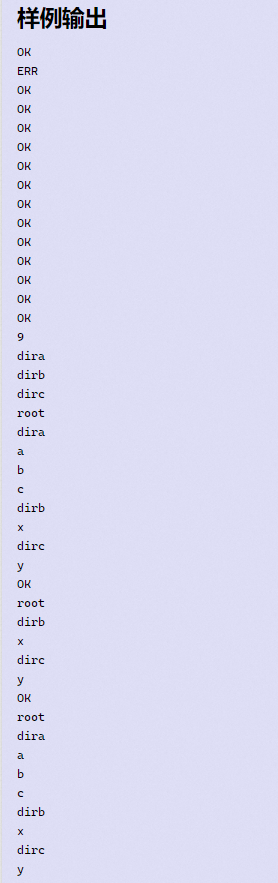
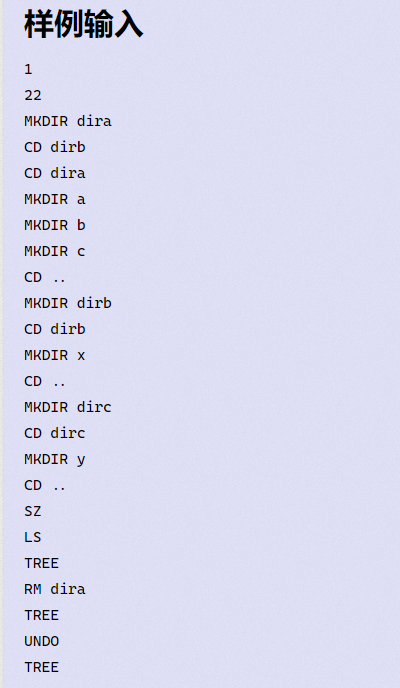
## [A - 咕咕东的目录管理器](https://vjudge.net/problem/Kattis-directorymanagement)

[Kattis - directorymanagement](https://vjudge.net/problem/Kattis-directorymanagement/origin)









**思路：**

1.每个节点就是一个文件夹，包括名字，子文件夹的信息，根文件夹名等信息

2.目录树类包括根文件夹，总文件夹数，文件树的状态及对文件树的各种操作

3.利用map结构来储存每个目录节点的对应子目录，节点的关键字用数组的下标来表示。

4.对于撤销功能，可以用vector数组储存每次操作的目录信息·，每次MKDIR，RM，CD命令成功后，都把其对应的命令、当前目录、操作的子目录信息存下来。

5.由于可能多次遇到同一目录的打印，可以先根据当前子目录数量把可能要打印到的都保存下来，如果打印时这些信息没有改动就可以直接打印，如果遇到其他操作改动就重新更新下之前保存的信息。

**代码：**

#include <algorithm>

#include <cstring>

#include <iostream>

#include <map>

#include <vector>

using namespace std;

int maxn = 100001;

int tot, n, t;

string \_cmd[] = {"MKDIR", "RM", "CD", "SZ", "LS", "TREE", "UNDO"};

struct CMD {

    string name, sstr;

    int type;

    void init(string *s*) {

        name = s;

        for (int i = 0; i < 7; ++i) {

            if (s == \_cmd[i]) {

                type = i;

                if (i < 3) cin >> sstr;

                break;

            }

        }

    }

} cmd;

struct Node {

    string name;

    map<string, int> nexts;

    vector<string> Front, End;

    bool updated;

    int root, sz;

    void init(string *s*, int *rt*) {

        updated = false;

        root = rt;

        name = s;

        sz = 1;

        Front.clear();

        End.clear();

        nexts.clear();

    }

} node[maxn];

class Tree {

   protected:

    int now;*//实时位置*

    int tot;

    Node root;

    vector<pair<string, pair<int, int> > > undo;

    CMD cmd;

   public:

    void init() {

        node[0].init("root", -1);

        root = node[0];

        undo.clear();

        tot = 0;

        now = 0;

    }

    void add(string *s*, int *rt*) {

        node[++tot].init(s, rt);

        node[rt].nexts[s] = tot;

    }

    void update(int *id*, int *num*) {

        while (id != -1) {

            node[id].updated = 0;

            node[id].sz += num;

            id = node[id].root;

        }

    }

    void viewFront(int *id*) {

        node[id].Front.push\_back(node[id].name);

        if (node[id].sz == 1) return;

        if (node[id].sz < 11) {

            for (auto i : node[id].nexts) {

                if (!node[i.second].updated) push(i.second);

                node[id].Front.insert(node[id].Front.end(),

                                      node[i.second].Front.begin(),

                                      node[i.second].Front.end());

            }

            return;

        }

        int ct = 1;

        for (auto i : node[id].nexts) {

            if (!node[i.second].updated) push(i.second);

            for (auto j : node[i.second].Front) {

                node[id].Front.push\_back(j);

                ++ct;

                if (ct >= 5) break;

            }

            if (ct >= 5) break;

        }

    }

    void viewEnd(int *id*) {

        int ct = 0;

        auto it = node[id].nexts.end();

        --it;

        for (;; --it) {

            if (!node[it->second].updated) push(it->second);

            int u = it->second;

            for (int i = node[u].End.size() - 1; i >= 0; --i) {

                node[id].End.push\_back(node[u].End[i]);

                ++ct;

                if (ct >= 5) {

                    reverse(node[id].End.begin(), node[id].End.end());

                    break;

                }

            }

            if (ct >= 5) break;

            if (it == node[id].nexts.begin()) break;

        }

    }

    void push(int *id*) {

        node[id].Front.clear();

        node[id].End.clear();

        viewFront(id);

        if (node[id].sz > 10)

            viewEnd(id);

        else

            node[id].End = node[id].Front;

        node[id].updated = true;

    }

    void MKDIR();

    void RM();

    void CD();

    void SZ();

    void LS();

    void TREE();

    void UNDO();

    void solve();

} tree;

void Tree::MKDIR() {

    if (node[now].nexts.count(cmd.sstr)) {

        cout << "ERR" << endl;

        return;

    }

    add(cmd.sstr, now);

    update(now, 1);

    undo.push\_back(make\_pair("MKDIR", make\_pair(now, tot)));

    cout << "OK" << endl;

}

void Tree::RM() {

    if (!node[now].nexts.count(cmd.sstr)) {

        cout << "ERR" << endl;

        return;

    }

    int u = node[now].nexts[cmd.sstr];

    update(now, (-1) \* node[u].sz);

    node[now].nexts.erase(node[u].name);

    undo.push\_back(make\_pair("RM", make\_pair(now, u)));

    cout << "OK" << endl;

}

void Tree::CD() {

    if (cmd.sstr == "..") {

        if (node[now].root == -1) {

            cout << "ERR" << endl;

            return;

        }

        undo.push\_back(make\_pair("CD", make\_pair(now, node[now].root)));

        now = node[now].root;

        cout << "OK" << endl;

        return;

    }

    if (!node[now].nexts.count(cmd.sstr)) {

        cout << "ERR" << endl;

        return;

    }

    int u = node[now].nexts[cmd.sstr];

    undo.push\_back(make\_pair("CD", make\_pair(now, u)));

    now = u;

    cout << "OK" << endl;

}

void Tree::SZ() { cout << node[now].sz << endl; }

void Tree::LS() {

    int t = node[now].nexts.size();

    if (t == 0) {

        cout << "EMPTY" << endl;

        return;

    }

    auto pos = node[now].nexts.begin();

    if (t > 0 && t < 11) {

        while (pos != node[now].nexts.end()) {

            cout << pos->first << endl;

            pos++;

        }

        return;

    }

    for (int i = 0; i < 5; ++i) {

        cout << pos->first << endl;

        pos++;

    }

    cout << "..." << endl;

    pos = node[now].nexts.end();

    for (int i = 0; i < 5; ++i) pos--;

    for (int i = 0; i < 5; ++i) {

        cout << pos->first << endl;

        pos++;

    }

}

void Tree::TREE() {

    if (!node[now].updated) push(now);

    if (node[now].sz == 1)

        cout << "EMPTY" << endl;

    else if (node[now].sz > 1 && node[now].sz < 11) {

        for (int i = 0; i < node[now].Front.size(); ++i)

            cout << node[now].Front[i] << endl;

    } else {

        for (int i = 0; i < 5; ++i) cout << node[now].Front[i] << endl;

        cout << "..." << endl;

        for (int i = 5; i > 0; --i)

            cout << node[now].End[node[now].End.size() - i] << endl;

    }

}

void Tree::UNDO() {

    if (!undo.size()) {

        cout << "ERR" << endl;

        return;

    }

    auto e = undo[undo.size() - 1];

    undo.pop\_back();

    cout << "OK" << endl;

    int tmp = now;

    if (e.first == "MKDIR") {

        cmd.name = "RM";

        now = e.second.first;

        cmd.sstr = node[e.second.second].name;

        int u = node[now].nexts[cmd.sstr];

        update(now, (-1) \* node[u].sz);

        node[now].nexts.erase(node[u].name);

        now = tmp;

    } else if (e.first == "RM") {

        now = e.second.first;

        int u = e.second.second;

        update(now, node[u].sz);

        node[now].nexts[node[u].name] = u;

        now = tmp;

    } else {

        now = e.second.first;

    }

}

void Tree::solve() {

    init();

    for (int i = 0; i < n; ++i) {

        string s;

        cin >> s;

        cmd.init(s);

        switch (cmd.type) {

            case 0:

                MKDIR();

                break;

            case 1:

                RM();

                break;

            case 2:

                CD();

                break;

            case 3:

                SZ();

                break;

            case 4:

                LS();

                break;

            case 5:

                TREE();

                break;

            case 6:

                UNDO();

                break;

        }

    }

}

int main() {

    ios::sync\_with\_stdio(false);

    while (cin >> t) {

        for (int i = 0; i < t; ++i) {

            cin >> n;

            tree.solve();

        }

    }

    return 0;

}

## [B - 东东学打牌](https://vjudge.net/problem/%E8%AE%A1%E8%92%9C%E5%AE%A2-41408)

[计蒜客 - 41408](https://vjudge.net/problem/%E8%AE%A1%E8%92%9C%E5%AE%A2-41408/origin)

**题面**

最近，东东沉迷于打牌。所以他找到 HRZ、ZJM 等人和他一起打牌。由于人数众多，东东稍微修改了亿下游戏规则：

* 所有扑克牌只按数字来算大小，忽略花色。
* 每张扑克牌的大小由一个值表示。A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K 分别指代 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13。
* 每个玩家抽得 5 张扑克牌，组成一手牌！（每种扑克牌的张数是无限的，你不用担心，东东家里有无数副扑克牌）

理所当然地，一手牌是有不同类型，并且有大小之分的。

举个栗子，现在东东的 "一手牌"（记为 α），瑞神的 "一手牌"（记为 β），要么 α > β，要么 α < β，要么 α = β。

那么这两个 "一手牌"，如何进行比较大小呢？首先对于不同类型的一手牌，其值的大小即下面的标号；对于同类型的一手牌，根据组成这手牌的 5 张牌不同，其值不同。下面依次列举了这手牌的形成规则：

1. 大牌：这手牌不符合下面任一个形成规则。如果 α 和 β 都是大牌，那么定义它们的大小为组成这手牌的 5 张牌的大小总和。
2. 对子：5 张牌中有 2 张牌的值相等。如果 α 和 β 都是对子，比较这个 "对子" 的大小，如果 α 和 β 的 "对子" 大小相等，那么比较剩下 3 张牌的总和。
3. 两对：5 张牌中有两个不同的对子。如果 α 和 β 都是两对，先比较双方较大的那个对子，如果相等，再比较双方较小的那个对子，如果还相等，只能比较 5 张牌中的最后那张牌组不成对子的牌。
4. 三个：5 张牌中有 3 张牌的值相等。如果 α 和 β 都是 "三个"，比较这个 "三个" 的大小，如果 α 和 β 的 "三个" 大小相等，那么比较剩下 2 张牌的总和。
5. 三带二：5 张牌中有 3 张牌的值相等，另外 2 张牌值也相等。如果 α 和 β 都是 "三带二"，先比较它们的 "三个" 的大小，如果相等，再比较 "对子" 的大小。
6. 炸弹：5 张牌中有 4 张牌的值相等。如果 α 和 β 都是 "炸弹"，比较 "炸弹" 的大小，如果相等，比较剩下那张牌的大小。
7. 顺子：5 张牌中形成 x, x+1, x+2, x+3, x+4。如果 α 和 β 都是 "顺子"，直接比较两个顺子的最大值。
8. 龙顺：5 张牌分别为 10、J、Q、K、A。

作为一个称职的魔法师，东东得知了全场人手里 5 张牌的情况。他现在要输出一个排行榜。排行榜按照选手们的 "一手牌" 大小进行排序，如果两个选手的牌相等，那么人名字典序小的排在前面。

不料，此时一束宇宙射线扫过，为了躲避宇宙射线，东东慌乱中清空了他脑中的 Cache。请你告诉东东，全场人的排名

**输入**

输入包含多组数据。每组输入开头一个整数 n (1 <= n <= 1e5)，表明全场共多少人。  
随后是 n 行，每行一个字符串 s1 和 s2 （1 <= |s1|,|s2| <= 10）， s1 是对应人的名字，s2 是他手里的牌情况。

**输出**

对于每组测试数据，输出 n 行，即这次全场人的排名。

**样例输入**

3  
DongDong AAA109  
ZJM 678910  
Hrz 678910

**样例输出**

Hrz  
ZJM  
DongDong

**思路：**

确定牌型->根据牌型确定分值->进行比较->对分值进行排序

**代码：**

#include "algorithm"

#include "cstring"

#include "iostream"

using namespace std;

int n;

int cnt[15];

int tot = 0;

struct card {

    string name;

    int sum, key;

    int key1, key2, key3, key4, key5;

    bool operator<(card *p*);

} cards[100001];

bool card::operator<(card *p*) {

    if (key != p.key)

        return key > p.key;*//ç‰Œåž‹æ¯”è¾ƒ*

    else {

        switch (key) {

            case 7:

                if (sum != p.sum)

                    return sum > p.sum;

                else

                    return name < p.name;

            case 2:

                if (key1 != p.key1)

                    return key1 > p.key1;

                else {

                    if (sum != p.sum)

                        return sum > p.sum;

                    else

                        return name < p.name;

                }

            case 3:

                if (key1 != p.key1)

                    return key1 > p.key1;

                else {

                    if (key2 != p.key2)

                        return key2 > p.key2;

                    else {

                        if (sum != p.sum)

                            return sum > p.sum;

                        else

                            return name < p.name;

                    }

                }

            case 4:

                if (key3 != p.key3)

                    return key3 > p.key3;

                else {

                    if (sum != p.sum)

                        return sum > p.sum;

                    else

                        return name < p.name;

                }

            case 5:

                if (key3 != p.key3)

                    return key3 > p.key3;

                else {

                    if (key4 != p.key4)

                        return key4 > p.key4;

                    else

                        return name < p.name;

                }

            case 6:

                if (key5 != p.key5)

                    return key5 > p.key5;

                else {

                    if (sum != p.sum)

                        return sum > p.sum;

                    else

                        return name < p.name;

                }

case 8:

             case 1:

        }

    }

}

void add(string *name*, string *str*) {

    card p;

    memset(cnt, 0, sizeof(cnt));

    int size = str.size();

    int t;

    int sum = 0, key = 1, key1 = 0, key2 = 0, key3 = 0, key4 = 0, key5 = 0;

    int max = -1, min = 15;

    for (int i = 0; i < size; ++i) {

        t = 0;

        switch (str[i]) {

            case '0':

                t = 0;

                break;

            case '1':

                t = 10;

                cnt[t]++;

                break;

            case 'A':

                t = 1;

                cnt[t]++;

                break;

            case 'J':

                t = 11;

                cnt[t]++;

                break;

            case 'Q':

                t = 12;

                cnt[t]++;

                break;

            case 'K':

                t = 13;

                cnt[t]++;

                break;

            default:

                t = str[i] - '0';

                cnt[t]++;

                break;*// 2~9*

        }

        sum += t;

        if (t) {

            switch (cnt[t]) {

                case 2:

                    if (key1)

                        key2 = t;

                    else

                        key1 = t;

                    break;

                case 3:

                    if (key1 == t) {

                        key1 = 0;

                        key3 = t;

                    } else {

                        key2 = 0;

                        key3 = t;

                    }

                    break;

                case 4:

                    key3 = 0;

                    key5 = t;

                    break;

            }

            if (t > max) max = t;

            if (t < min) min = t;

        }

    }

    if (key5)

        key = 6;

    else if (key3) {

        key4 = (key1 == 0) ? key2 : key1;

        if (key4) {

            key1 = key2 = 0;

            key = 5;

        } else

            key = 4;

    } else if (key1 || key2) {

        if (key1 && key2) {

            key = 3;

            if (key1 < key2) swap(key1, key2);

        } else {

            key1 = key1 > 0 ? key1 : key2;

            key = 2;

        }

    } else if ((max - min) == 4) {

        key = 7;

    } else if (cnt[1] && cnt[10] && cnt[11] && cnt[12] && cnt[13]) {

        key = 8;

    } else

        key = 1;

    p.key = key;

    p.key1 = key1;

    p.key2 = key2;

    p.key3 = key3;

    p.key4 = key4;

    p.key5 = key5;

    p.name = name;

    p.sum = sum;

    cards[tot] = p;

    ++tot;

}

int main() {

    string s1, s2;

    while (cin >> n) {

        tot = 0;

        while (n--) {

            cin >> s1 >> s2;

            add(s1, s2);

        }

        sort(cards, cards + tot);

        for (int i = 0; i < tot; ++i) cout << cards[i].name << endl;

    }

    return 0;

}

[C - 签到题](https://vjudge.net/problem/CodeForces-1042A)

[CodeForces - 1042A](https://vjudge.net/problem/CodeForces-1042A/origin)

SDUQD 旁边的滨海公园有 x 条长凳。第 i 个长凳上坐着 a\_i 个人。这时候又有 y 个人将来到公园，他们将选择坐在某些公园中的长凳上，那么当这 y 个人坐下后，记k = 所有椅子上的人数的最大值，那么k可能的最大值mx和最小值mn分别是多少。

**Input**

第一行包含一个整数 x (1 <= x <= 100) 表示公园中长椅的数目  
第二行包含一个整数 y (1 <= y <= 1000) 表示有 y 个人来到公园  
接下来 x 个整数 a\_i (1<=a\_i<=100)，表示初始时公园长椅上坐着的人数

**Output**

输出 mn 和 mx

**Input Example**

3  
7  
1  
6  
1

**Output Example**

6 13

**样例解释**

最初三张椅子的人数分别为 1 6 1  
接下来来了7个人。  
可能出现的情况为{1 6 8},{1,7,7},…,{8,6,1}  
相对应的k分别为8,7,…,8  
其中，状态{1,13,1}的k = 13，为mx  
状态{4,6,5}和状态{5,6,4}的k = 6，为mn

**思路:**

1.最大值：来的所有人全都坐在初始人最多的椅子上，

2.最小值：升序排序所有椅子，用新来的人填补前n-1个椅子使人数平均分布，与原最大值比较，如果平均值更大则最终答案最小值就是原来所有人加上新来的人平均值向上取整，否则就是原最大值。

**代码：**

#include "algorithm"

#include "iostream"

using namespace std;

int a[101];

int x, y;

int main() {

    while (cin >> x) {

        cin >> y;

        int tot = 0;

        for (int i = 0; i < x; i++) {

            cin >> a[i];

            tot += a[i];

        }

        sort(a, a + x);

        tot -= a[x - 1];

        int mx = a[x - 1] + y, mn, leftCount, yuShu, tot1;

        tot1 = tot + y;

        if (tot1 <= a[x - 1] \* (x - 1)) {

            mn = a[x - 1];

        } else {

            leftCount = (tot + a[x - 1] + y) - a[x - 1] \* x;

            yuShu = leftCount % x;

            if (yuShu == 0) {

                mn = a[x - 1] + leftCount / x;

            } else {

                mn = a[x - 1] + leftCount / x + 1;

            }

        }

        cout << mn << " " << mx << endl;

    }

    return 0;

}

# week10

## [B - LIS & LCS](https://vjudge.net/problem/Gym-277140A)

[Gym - 277140A](https://vjudge.net/problem/Gym-277140A/origin)

东东有两个序列A和B。

他想要知道序列A的LIS和序列AB的LCS的长度。

注意，LIS为严格递增的，即a1<a2<…<ak(ai<=1,000,000,000)。

**Input**

第一行两个数n，m（1<=n<=5,000,1<=m<=5,000）  
第二行n个数，表示序列A  
第三行m个数，表示序列B

**Output**

输出一行数据ans1和ans2，分别代表序列A的LIS和序列AB的LCS的长度

**Simple Input**

5 5  
1 3 2 5 4  
2 4 3 1 5

**Simple Output**

3 2

**思路：**

LIS：最长递增子序列 从第一个数每当移动到位置i，遍历该位置之前的所有数，确定i位置的LIS长度并与前一次比较取较大值，最后得到答案。

LCS：最长公共子序列 状态转移方程：dp1[i + 1][j + 1] = max(dp1[i + 1][j], dp1[i][j + 1]);

**代码：**

#include "iostream"

using namespace std;

int a[5001], b[5001], dp[5001] ,dp1[5001][5001];

int n, m;

int main() {

    while (cin >> n >> m) {

        int ans = 0;

        for (int i = 0; i < n; i++) {

            cin >> a[i];

        }

        for (int i = 0; i < m; i++) {

            cin >> b[i];

        }

        for (int i = 0; i < n; i++) {

            dp[i] = 1;

            for (int j = 0; j < i; j++) {

                if (a[i] > a[j]) {

                    dp[i] = max(dp[i], dp[j] + 1);

                }

            }

            ans = max(ans, dp[i]);

        }

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < m; j++) {

                if (a[i] == b[j]) {

                    dp1[i + 1][j + 1] = dp1[i][j] + 1;

                } else {

dp1[i + 1][j + 1] = max(dp1[i + 1][j], dp1[i][j + 1]);

                }

            }

        }

        cout << ans << " " << dp1[n][m] << endl;

    }

}

## [C - 拿数问题 II](https://vjudge.net/problem/CodeForces-455A)

[CodeForces - 455A](https://vjudge.net/problem/CodeForces-455A/origin)

YJQ 上完第10周的程序设计思维与实践后，想到一个绝妙的主意，他对拿数问题做了一点小修改，使得这道题变成了 拿数问题 II。  
  
给一个序列，里边有 n 个数，每一步能拿走一个数，比如拿第 i 个数， Ai = x，得到相应的分数 x，但拿掉这个 Ai 后，x+1 和 x-1 (如果有 Aj = x+1 或 Aj = x-1 存在) 就会变得不可拿（但是有 Aj = x 的话可以继续拿这个 x）。求最大分数。  
  
**本题和课上讲的有些许不一样，但是核心是一样，需要你自己思考。**

**Input**

第一行包含一个整数 *n* (1 ≤ *n* ≤ 105)，表示数字里的元素的个数

第二行包含n个整数*a*1, *a*2, ..., *an* (1 ≤ *ai* ≤ 105)

**Output**

输出一个整数：n你能得到最大分值。

**Example**

Input

2  
1 2

Output

2

Input

3  
1 2 3

Output

4

Input

9  
1 2 1 3 2 2 2 2 3

Output

10

**Hint**

对于第三个样例：先选任何一个值为2的元素，最后数组内剩下4个2。然后4次选择2，最终得到10分。

**思路：**

状态转移方程为：dp[i]=max(dp[i-1],dp[i-2]+a[i]\*i)

a[i]为数字i出现的个数，dp为拿i可得到的分数

注意数据范围，要用long long数组

**代码：**

#include <iostream>

using namespace std;

long long a[100001];

long long dp[100001];

int maxx = 0;

int n;

int main() {

    while (cin >> n) {

        int temp;

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

            cin >> temp;

            a[temp]++;

            maxx = max(maxx, temp);

        }

        dp[1] = a[1];

        for (int i = 2; i <= maxx; i++) {

            dp[i] = max(dp[i - 1], dp[i - 2] + a[i] \* i);

        }

        cout << dp[maxx] << endl;

    }

}

# week11

## [E - 选做题11-1 东东与 ATM](https://vjudge.net/problem/POJ-1276)

[POJ - 1276](https://vjudge.net/problem/POJ-1276/origin)

一家银行计划安装一台用于提取现金的机器。  
机器能够按要求的现金量发送适当的账单。  
机器使用正好N种不同的面额钞票，例如D\_k，k = 1,2,…,N，并且对于每种面额D\_k，机器都有n\_k张钞票。  
例如，  
N = 3，  
n\_1 = 10，D\_1 = 100，  
n\_2 = 4，D\_2 = 50，  
n\_3 = 5，D\_3 = 10  
表示机器有10张面额为100的钞票、4张面额为50的钞票、5张面额为10的钞票。  
东东在写一个 ATM 的程序，可根据具体金额请求机器交付现金。  
注意，这个程序计算程序得出的最大现金少于或等于可以根据设备的可用票据供应有效交付的现金。

**Input**

程序输入来自标准输入。 输入中的每个数据集代表特定交易，其格式为：Cash N n1 D1 n2 D2 ... nN DN其中0 <= Cash <= 100000是所请求的现金量，0 <= N <= 10是 纸币面额的数量，0 <= nk <= 1000是Dk面额的可用纸币的数量，1 <= Dk <= 1000，k = 1，N。 输入中的数字之间可以自由出现空格。 输入数据正确。

**Output**

对于每组数据，程序将在下一行中将结果打印到单独一行上的标准输出中。

**Sample Input**

735 3 4 125 6 5 3 350

633 4 500 30 6 100 1 5 0 1

735 0

0 3 10 100 10 50 10 10

**Sample Output**

735

630

0

0

**Hint**

第一个数据集指定一笔交易，其中请求的现金金额为 735。 机器包含3种面额的纸币：4张钞票 125、6张钞票 5和3张钞票 350。 机器可以交付所需现金的确切金额。

在第二种情况下，机器的票据供应不能满足所要求的确切现金数量。 可以交付的最大现金为 630。 请注意，在机器中组合钞票以匹配交付的现金有多种可能性。

在第三种情况下，机器是空的，没有现金交付。 在第四种情况下，请求的现金金额为 0，因此机器不交付现金。

思路：

1.属于多重背包问题，且重量与价值是一个属性。

2.复杂度为1e9，大于1s，故采用二进制拆分；

3.二进制拆分指的就是将原来通过一个一个加，变成2的指数次幂来加，以及一个余数，能将n的复杂度降为logn.

4.套用0-1背包模板，取最大值输出

代码：

#include <cstring>

#include <iostream>

using namespace std;

int  c[20],f[100001], w[1001], ww[1001];

int ans, V, cnt, N;

int main() {

    while (cin >> V >> N) {

        memset(f, 0, sizeof(f));

        memset(w, 0, sizeof(w));

        memset(c, 0, sizeof(c));

        memset(ww, 0, sizeof(ww));

        cnt = 0;

        for (int i = 1; i <= N; i++) {

            cin >> c[i] >> w[i];

        }

        for (int i = 1; i <= N; i++) {

            int t = c[i], k;

            for (k = 1; k <= t; k <<= 1) {

                cnt++;

                ww[cnt] = k \* w[i];

                t -= k;

            }

            if (t > 0) {

                cnt++;

                ww[cnt] = t \* w[i];

            }

        }

        N = cnt;

        for (int i = 1; i <= N; i++) {

            for (int j = V; j >= ww[i]; j--) {

                f[j] = max(f[j], f[j - ww[i]] + ww[i]);

            }

        }

        ans = f[V];

        cout << ans << endl;

    }

}

## [F - 选做题11-2 东东开车了](https://vjudge.net/problem/UVA-624)

[UVA - 624](https://vjudge.net/problem/UVA-624/origin)

东东开车出去泡妞(在梦中)，车内提供了 n 张CD唱片，已知东东开车的时间是 n 分钟，他该如何去选择唱片去消磨这无聊的时间呢

假设:

* CD数量不超过20张
* 没有一张CD唱片超过 *N* 分钟
* 每张唱片只能听一次
* 唱片的播放长度为整数
* *N* 也是整数

我们需要找到最能消磨时间的唱片数量，并按使用顺序输出答案（必须是听完唱片，不能有唱片没听完却到了下车时间的情况发生）

本题是 Special Judge

**Input**

多组输入

每行输入第一个数字*N*, 代表总时间，第二个数字 *M* 代表有 *M* 张唱片，后面紧跟 *M* 个数字，代表每张唱片的时长 例如样例一: *N*=5, *M*=3, 第一张唱片为 1 分钟, 第二张唱片 3 分钟, 第三张 4 分钟

所有数据均满足以下条件:

N≤10000  
M≤20

**Output**

输出所有唱片的时长和总时长，具体输出格式见样例

**Sample Input**

5 3 1 3 4

10 4 9 8 4 2

20 4 10 5 7 4

90 8 10 23 1 2 3 4 5 7

45 8 4 10 44 43 12 9 8 2

**Sample Output**

1 4 sum:5

8 2 sum:10

10 5 4 sum:19

10 23 1 2 3 4 5 7 sum:55

4 10 12 9 8 2 sum:45

**思路：**

1.与上题的多重背包问题相似，唱片时间与开车时间代表价值与重量。

2.对于路径，用一个vis数组将能进行更新的两个数用数组存储起来，然后倒推将选择过得唱片记录下来，同样价值相应减少，最后顺序输出。

**代码：**

#include <cstring>

#include <iostream>

using namespace std;

int f[10010], vis[30][10010], r[30];

int N, M, w[30], x, t;

int main() {

    while (cin >> N >> M) {

            memset(f, 0, sizeof(f));

            memset(w, 0, sizeof(w));

            memset(vis, 0, sizeof(vis));

            memset(r, 0, sizeof(r));

            for (int i = 1; i <= M; i++) {

                cin >> w[i];

            }

            for (int i = 1; i <= M; i++) {

                for (int j = N; j >= w[i]; j--) {

                    if (f[j] < f[j - w[i]] + w[i]) {

                        f[j] = f[j - w[i]] + w[i];

                        vis[i][j] = 1;

                    }

                }

            }

            for (int i = M, j = N; j >= 0 && i >= 0; i--) {

                if (vis[i][j] == 1) {

                    r[i] = 1;

                    j -= w[i];

                }

            }

            for (int i = 1; i <= M; i++) {

                if (r[i]) cout << w[i] << " ";

            }

            cout << "sum:" << f[N] << endl;

        }

}

# week12

## [D - 选做题 - 1](https://vjudge.net/problem/POJ-2955)

[POJ - 2955](https://vjudge.net/problem/POJ-2955/origin)

We give the following inductive definition of a “regular brackets” sequence:

* the empty sequence is a regular brackets sequence,
* if *s* is a regular brackets sequence, then (*s*) and [*s*] are regular brackets sequences, and
* if *a* and *b* are regular brackets sequences, then *ab* is a regular brackets sequence.
* no other sequence is a regular brackets sequence

For instance, all of the following character sequences are regular brackets sequences:

(), [], (()), ()[], ()[()]

while the following character sequences are not:

(, ], )(, ([)], ([(]

Given a brackets sequence of characters *a*1*a*2 … *an*, your goal is to find the length of the longest regular brackets sequence that is a subsequence of *s*. That is, you wish to find the largest *m* such that for indices *i*1, *i*2, …, *im* where 1 ≤ *i*1 < *i*2 < … < *im* ≤ *n*, *ai*1*ai*2 … *aim* is a regular brackets sequence.

Given the initial sequence ([([]])], the longest regular brackets subsequence is [([])].

**Input**

The input test file will contain multiple test cases. Each input test case consists of a single line containing only the characters (, ), [, and ]; each input test will have length between 1 and 100, inclusive. The end-of-file is marked by a line containing the word “end” and should not be processed.

**Output**

For each input case, the program should print the length of the longest possible regular brackets subsequence on a single line.

**Sample Input**

((()))

()()()

([]])

)[)(

([][][)

end

**Sample Output**

6

6

4

0

6

思路：

题意为给定一串包含”(“，”)“，”[“，”]”的字符串，从中选取最长的能够实现括号合法匹配的序列，输出序列长度。包含多组数据，end表示结束。

1.采用区间dp的思路。f[i][j]表示从i到j符合题意的子序列的最大长度。

2.首先枚举区间长度。然后从头开始枚举起点i，因为长度已知了，所以j可以确定。如果i j位置处恰好可以匹配：s[i]=='(' s[j]==')'或者s[i]=='[' s[j]==']'，判断i，j是否位置相邻，相邻则f[i][j]=2,否则此时f[i][j]=f[i+1][j-1]+2.

代码：

#include <cstring>

#include <iostream>

using namespace std;

int n, f[1001][1001];

string s;

bool judge(int *i*, int *j*) {

    if (s[i] == '[' && s[j] == ']') return 1;

    if (s[i] == '(' && s[j] == ')') return 1;

    return 0;

}

int main() {

    while (cin >> s) {

        if (s == "end") {

            break;

        }

        n = s.length();

        memset(f, 0, sizeof(f));

        for (int ii = 1; ii < n; ii++)

            for (int i = 0; i + ii < n; i++) {

                int j = i + ii;

                if (judge(i, j)) {

                    if (j == i + 1)

                        f[i][j] = 2;

                    else

                        f[i][j] = f[i + 1][j - 1] + 2;

                }

                for (int k = i; k < j; k++)

                    f[i][j] = max(f[i][j], f[i][k] + f[k + 1][j]);

            }

        cout << f[0][n - 1] << endl;

    }

    return 0;

}

## [E - 选做题 - 2](https://vjudge.net/problem/HDU-1074)

[HDU - 1074](https://vjudge.net/problem/HDU-1074/origin)

**Description**

马上假期就要结束了，zjm还有 n 个作业，完成某个作业需要一定的时间，而且每个作业有一个截止时间，若超过截止时间，一天就要扣一分。  
zjm想知道如何安排做作业，使得扣的分数最少。  
Tips: 如果开始做某个作业，就必须把这个作业做完了，才能做下一个作业。

**Input**

有多组测试数据。第一行一个整数表示测试数据的组数  
第一行一个整数 n(1<=n<=15)  
接下来n行，每行一个字符串(长度不超过100) S 表示任务的名称和两个整数 D 和 C，分别表示任务的截止时间和完成任务需要的天数。  
这 n 个任务是按照字符串的字典序**从小到大**给出。

**Output**

每组测试数据，输出最少扣的分数，并输出完成作业的方案，如果有多个方案，输出字典序最小的一个。

**Sample Input**

2

3

Computer 3 3

English 20 1

Math 3 2

3

Computer 3 3

English 6 3

Math 6 3

**Sample Output**

2

Computer

Math

English

3

Computer

English

Math

**Hint**

在第二个样例中，按照 Computer->English->Math 和 Computer->Math->English 的顺序完成作业，所扣的分数都是 3，由于 English 的字典序比 Math 小，故输出前一种方案。

**思路：**

1.状态压缩dp，f数组表示当前状态最少需要扣掉的分数。

2.暴力枚举所有情况，然后枚举所有的作业，若某作业包含在当前情况中，则算出当前需要被扣除的分数，更新当前状态的最小值，同时记录前驱，最后递归输出结果。

3.逆序枚举作业避免了按字典序排序。

**代码：**

#include <cstring>

#include <iostream>

using namespace std;

const int maxX = 1e9;

struct node {

    string cla;

    int d;

    int c;

} a[16];

int n, f[1 << 16], pre[1 << 16], sum[1 << 16];

void output(int *x*) {

    if (x == 0) return;

    output(x - (1 << pre[x]));

    cout << a[pre[x]].cla << endl;

}

int main() {

    int T;

    while (cin >> T) {

        while (T--) {

            memset(f, 0, sizeof(f));

            memset(pre, 0, sizeof(pre));

            memset(sum, 0, sizeof(sum));

            cin >> n;

            for (int i = 0; i < n; i++) cin >> a[i].cla >> a[i].d >> a[i].c;

            int tot = (1 << n);

            for (int i = 1; i <= tot; i++) {

                f[i] = maxX;

                for (int j = n - 1; j >= 0; j--) {

                    int now = 1 << j;

                    if (!(i & now)) continue;

                    int tmp = sum[i - now] + a[j].c - a[j].d;

                    tmp = max(0, tmp);

                    if (f[i] > f[i - now] + tmp) {

                        f[i] = f[i - now] + tmp;

                        sum[i] = sum[i - now] + a[j].c;

                        pre[i] = j;

                    }

                }

            }

            cout << f[tot - 1] << endl;

            output(tot - 1);

        }

    }

    return 0;

}

# week13-15选做部分没做（假期会补上）

# week10

## [B - 团 队 聚 会 （不支持C++11）](https://vjudge.net/problem/POJ-1960)

[POJ - 1960](https://vjudge.net/problem/POJ-1960/origin)

**题目描述**

TA团队每周都会有很多任务，有的可以单独完成，有的则需要所有人聚到一起，开过会之后才能去做。但TA团队的每个成员都有各自的事情，找到所有人都有空的时间段并不是一件容易的事情。

给出每位助教的各项事情的时间表，你的任务是找出所有可以用来开会的时间段。

**输入格式**

第一行一个数T（T≤100），表示数据组数。

对于每组数据，第一行一个数m（2 ≤ m ≤ 20），表示TA的数量。

对于每位TA，首先是一个数n（0≤ n≤100），表示该TA的任务数。接下来n行，表示各个任务的信息，格式如下

YYYY MM DD hh mm ss YYYY MM DD hh mm ss "some string here"

每一行描述的信息为：开始时间的年、月、日、时、分、秒；结束时间的年、月、日、时、分、秒，以及一些字符串，描述任务的信息。

数据约定：

所有的数据信息均为固定位数，位数不足的在在前面补前导0，数据之间由空格隔开。

描述信息的字符串中间可能包含空格，且总长度不超过100。

所有的日期时间均在1800年1月1日00:00:00到2200年1月1日00:00:00之间。

为了简化问题，我们假定所有的月份（甚至2月）均是30天的，数据保证不含有不合法的日期。

注意每件事务的结束时间点也即是该成员可以开始参与开会的时间点。

**输出格式**

对于每一组数据，首先输出一行"Scenario #i:"，i即表明是第i组数据。

接下来对于所有可以用来开会的时间段，每一个时间段输出一行。

需要满足如下规则：

1. 在该时间段的任何时间点，都应该有至少两人在场。
2. 在该时间段的任何时间点，至多有一位成员缺席。
3. 该时间段的时间长度至少应该1h。

所有的成员都乐意一天24h进行工作。

举个例子，假如现在TA团队有3位成员，TT、zjm、hrz。

那么这样的时间段是合法的：会议开始之初只有TT和zjm，后来hrz加入了，hrz加入之后TT离开了，此后直到会议结束，hrz和zjm一直在场。

要求：

1. 输出满足条件的所有的时间段，尽管某一段可能有400年那么长。
2. 时间点的格式为MM/DD/YYYY hh:mm:ss。
3. 时间段的输出格式为"appointment possible from T0 to T1"，其中T0和T1均应满足时间点的格式。
4. 严格按照格式进行匹配，如果长度不够则在前面补前导0。
5. 按时间的先后顺序输出各个时间段。
6. 如果没有合适的时间段，输出一行"no appointment possible"。
7. 每组数据末尾须打印额外的一行空行。

**Simple Input**

2  
3  
3  
2020 06 28 15 00 00 2020 06 28 18 00 00 TT study  
2020 06 29 10 00 00 2020 06 29 15 00 00 TT solving problems  
2020 11 15 15 00 00 2020 11 17 23 00 00 TT play with his magic cat  
4  
2020 06 25 13 30 00 2020 06 25 15 30 00 hrz play  
2020 06 26 13 30 00 2020 06 26 15 30 00 hrz study  
2020 06 29 13 00 00 2020 06 29 15 00 00 hrz debug  
2020 06 30 13 00 00 2020 06 30 15 00 00 hrz play  
1  
2020 06 01 00 00 00 2020 06 29 18 00 00 zjm study  
2  
1  
1800 01 01 00 00 00 2200 01 01 00 00 00 sleep  
0

**Simple Output**

Scenario #1:  
appointment possible from 01/01/1800 00:00:00 to 06/25/2020 13:30:00  
appointment possible from 06/25/2020 15:30:00 to 06/26/2020 13:30:00  
appointment possible from 06/26/2020 15:30:00 to 06/28/2020 15:00:00  
appointment possible from 06/28/2020 18:00:00 to 06/29/2020 10:00:00  
appointment possible from 06/29/2020 15:00:00 to 01/01/2200 00:00:00

Scenario #2:  
no appointment possible

思路：

1.定义一个time结构体，存储时间信息

2.将每一件事的开始时间和结束时间点都存储到time数组中，用数组tbegin记录开始时间点，tend记录结束时间点。

3.给两个time数组排序，按照顺序来判断每个时间点是否为合适的开始时间点和结束时间点。

4.每确定一个时间段，就输出，若没有合适的时间段，就输出“no appointment possible”

代码：

#include <algorithm>

#include <cstring>

#include <iostream>

using namespace std;

struct time {

    int yae, mon, day, hou, min, sec;

    time() {}

    time(int *y*, int *mo*, int *d*, int *h*, int *mi*, int *s*) {

        yae = y, mon = mo, day = d, hou = h, min = mi, sec = s;

    }

    bool operator<(const time &*b*) const {

        if (yae != b.yae) return yae < b.yae;

        if (mon != b.mon) return mon < b.mon;

        if (day != b.day) return day < b.day;

        if (hou != b.hou) return hou < b.hou;

        if (min != b.min) return min < b.min;

        return sec < b.sec;

    }

    bool operator>(const time &*b*) const { return b < \*this; }

    bool operator<=(const time &*b*) const { return !(b < \*this); }

    bool operator==(const time &*b*) const { return !(b < \*this || \*this < b); }

} s[25][120], e[25][120], t[4020];

int T, num[25], cnt, n;

time tbegin(1800, 1, 1, 0, 0, 0), tend(2200, 1, 1, 0, 0, 0);

int flag = 0;

void output(int *i*) {

    if (t[i].mon < 10) cout << "0";

    cout << t[i].mon << "/";

    if (t[i].day < 10) cout << "0";

    cout << t[i].day << "/";

    cout << t[i].yae << " ";

    if (t[i].hou < 10) cout << "0";

    cout << t[i].hou << ":";

    if (t[i].min < 10) cout << "0";

    cout << t[i].min << ":";

    if (t[i].sec < 10) cout << "0";

    cout << t[i].sec << "";

}

*//确定空闲段的右边界*

bool isRight(int *index*) {

    if (index == 0) return 0;

    int tot = 0;

    for (int i = 1; i <= n; i++) {

        if (num[i] == 0) {

            tot++;

            continue;

        }

        if (t[index] <= s[i][1] && t[index] > tbegin) {

            tot++;

            continue;

        }

        if (t[index] > e[i][num[i]]) {

            tot++;

            continue;

        }

        if (t[index] == e[i][num[i]]) continue;

        for (int j = 1; j <= num[i]; j++) {

            if (t[index] > s[i][j] && t[index] <= e[i][j]) break;

            if (j + 1 <= num[i] && t[index] > e[i][j] &&

                t[index] <= s[i][j + 1]) {

                tot++;

                break;

            }

        }

    }

    if (tot >= 2 && tot >= n - 1)

        return 1;

    else

        return 0;

}

*//确定空闲段的左边界*

bool isLeft(int *index*) {

    if (index == cnt) return 0;

    int tot = 0;

    for (int i = 1; i <= n; i++) {

        if (num[i] == 0) {

            tot++;

            continue;

        }

        if (t[index] < s[i][1]) {

            tot++;

            continue;

        }

        if (e[i][num[i]] <= t[index] && tend > t[index]) {

            tot++;

            continue;

        }

        for (int j = 1; j <= num[i]; j++) {

            if (s[i][j] <= t[index] && e[i][j] > t[index]) break;

            if (j + 1 <= num[i] && e[i][j] <= t[index] &&

                s[i][j + 1] > t[index]) {

                tot++;

                break;

            }

        }

    }

    if (tot >= 2 && tot >= n - 1)

        return 1;

    else

        return 0;

}

*//满足1h长*

bool anhour(int *left*, int *right*) {

    time l = t[left], r = t[right];

    if (r.yae - l.yae >= 2) return 1;

    r.mon += (r.yae - l.yae) \* 12;

    if (r.mon - l.mon >= 2) return 1;

    r.day += (r.mon - l.mon) \* 30;

    if (r.day - l.day >= 2) return 1;

    r.hou += (r.day - l.day) \* 24;

    if (r.hou - l.hou >= 2) return 1;

    r.min += (r.hou - l.hou) \* 60;

    r.sec += (r.min - l.min) \* 60;

    if (r.sec - l.sec >= 3600) return 1;

    return 0;

}

void judge(int *left*, int *right*) {

    if (!anhour(left, right)) return;

    flag = 1;

    cout << "appointment possible from ";

    output(left);

    cout << " to ";

    output(right);

    cout << endl;

}

int main() {

    cin.sync\_with\_stdio(false);

    while (cin >> T) {

        for (int cc = 1; cc <= T; cc++) {

            memset(num, 0, sizeof(num));

            memset(s, 0, sizeof(s));

            memset(e, 0, sizeof(e));

            memset(t, 0, sizeof(t));

            cnt = 0;

            flag = 0;

            t[++cnt] = tbegin;

            t[++cnt] = tend;

            cin >> n;

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

                cin >> num[i];

                for (int j = 1; j <= num[i]; j++) {

                    cin >> s[i][j].yae >> s[i][j].mon >> s[i][j].day >>

                        s[i][j].hou >> s[i][j].min >> s[i][j].sec;

                    cin >> e[i][j].yae >> e[i][j].mon >> e[i][j].day >>

                        e[i][j].hou >> e[i][j].min >> e[i][j].sec;

                    t[++cnt] = s[i][j];

                    t[++cnt] = e[i][j];

                    string ttemp;

                    getline(cin, ttemp);

                }

            }

            sort(t + 1, t + 1 + cnt);

            cout << "Scenario #" << cc << ":\n";

            int left = 1, right = 1;

            while (left <= cnt && right <= cnt) {

                right++;

                if (right > cnt) break;

*//找到使时间段最长的右边界*

                while (right <= cnt && isRight(right)) right++;

                right--;

                judge(left, right);

                left = right + 1;

*//确定左边界*

                while (left <= cnt && !isLeft(left)) left++;

                right = left;

            }

            if (flag == 0) cout << "no appointment possible\n";

            cout << endl;

        }

    }

}

# week14

## [A - 猫睡觉问题](https://vjudge.net/problem/HDU-3700)

[HDU - 3700](https://vjudge.net/problem/HDU-3700/origin)

众所周知，TT家里有一只魔法喵。这只喵十分嗜睡。一睡就没有白天黑夜。喵喵一天可以睡多次！！每次想睡多久就睡多久╭(╯^╰)╮

喵睡觉的时段是连续的，即一旦喵喵开始睡觉了，就不能被打扰，不然喵会咬人哒[○･｀Д´･ ○]

可以假设喵喵必须要睡眠连续不少于 A 个小时，即一旦喵喵开始睡觉了，至少连续 A 个小时内（即A\*60分钟内）不能被打扰！

现在你知道喵喵很嗜睡了，它一天的时长都在吃、喝、拉、撒、睡，换句话说要么睡要么醒着滴！

众所周知，这只魔法喵很懒，和TT一样懒，它不能连续活动超过 B 个小时。

猫主子是不用工作不用写代码滴，十分舒适，所以，它是想睡就睡滴。

但是，现在猫主子有一件感兴趣的事，就是上BiliBili网站看的新番。

新番的播放时间它已经贴在床头啦（每天都用同一张时间表哦），这段时间它必须醒着！！

作为一只喵喵，它认为安排时间是很麻烦的事情，现在请你帮它安排睡觉的时间段。

**Input**

多组数据，多组数据，多组数据哦，每组数据的格式如下：

第1行输入三个整数，A 和 B 和 N (1 <= A <= 24, 1 <= B <= 24, 1 <= n <= 20)

第2到N+1行为每日的新番时间表，每行一个时间段，格式形如 hh:mm-hh:mm (闭区间)，这是一种时间格式，hh:mm 的范围为 00:00 到 23:59。注意一下，时间段是保证不重叠的，但是可能出现跨夜的新番，即新番的开始时间点大于结束时间点。  
保证每个时间段的开始时间点和结束时间点不一样，即不可能出现类似 08:00-08:00 这种的时间段。时长的计算由于是闭区间所以也是有点坑的，比如 12:00-13:59 的时长就是 120 分钟。  
不保证输入的新番时间表有序。

**Output**

我们知道，时间管理是一项很难的活，所以你可能没有办法安排的那么好，使得这个时间段满足喵喵的要求，即每次睡必须时间连续且不少于 A 小时，每次醒必须时间连续且不大于 B 小时，还要能看完所有的番，所以输出的第一行是 Yes 或者 No，代表是否存在满足猫猫要求的时间管理办法。

然后，对于时间管理，你只要告诉喵喵，它什么时候睡觉即可。  
即第2行输出一个整数 k，代表当天有多少个时间段要睡觉  
接下来 k 行是喵喵的睡觉时间段，每行一个时间段，格式形如 hh:mm-hh:mm (闭区间)，这个在前面也有定义。注意一下，如果喵喵的睡眠时段跨越当天到达了明天，比如从23点50分睡到0点40分，那就输出23:50-00:40，如果从今晚23:50睡到明天早上7:30，那就输出23:50-07:30。

输出要排序吗？（输出打乱是能过的，也就是说，题目对输出的那些时间段间的顺序是没有要求的）

哦对了，喵喵告诉你说，本题是 Special Judge，如果你的输出答案和 Sample 不太一样，也可能是对的，它有一个判题程序来判定你的答案（当然，你对你自己的答案肯定也能肉眼判断）

**Sample Input**

12 12 1

23:00-01:00

3 4 3

07:00-08:00

11:00-11:09

19:00-19:59

**Sample Output**

Yes

1

01:07-22:13

No

你尝试给喵喵喂小鱼干，它告诉了你一个秘密：“媌，吧唧吧唧小鱼…吧唧吧唧干香吧唧吧唧，媌，这题最麻烦吧唧吧唧…的吧唧吧唧是最后一个番到第二天第一个番期间时间段的处理哦，媌，但是吧唧吧唧可以有一种方法可以很容易处理吧唧吧唧吧唧。啊呀，我要去睡觉啦，媌”

思路：

题目中读入A，B。不能连续醒着超过B小时，睡觉的时候至少睡A小时。

这里取睡觉的最大数，能睡就睡。然后题意处理，一张时间表是一组数据一直用的，为了时间计算方便，可将其全部转化为以分钟为单位的。构造一个结构体reg，其中记录每个番的开始时间和结束时间。注意跨夜的情况。

在读入数据时，若一个番的时间大于能醒着的最大时间，则直接判断不能。否则将所有读入的时间按照开始时间排序（本题不会出现交叉的情况）。然后从第一个番开始，记录下从什么时候醒着的，以用来判断是否能继续看番，记录上一个番的结束时间，用来判断是否能睡觉。

每次遍历到下一个番的时候，计算这个番开始时间，和上一个番的结束时间之间能不能睡觉，如果可以的话，记录这个休息的时间，并记录下来这个番的开始时间和结束时间。如果不可以睡觉，则判断是否能继续看，即从上一次醒来到这个番的结束，这段时间是否比能醒的最大时间大，如果是，则不能成功。否则只记录下结束时间，因为开始时间没有变。其中要对最后一个番进行单独处理，因为可能是跨夜的情况。最后将记录下来的休息时间全部输出即可。

代码：

#include <algorithm>

#include <cstring>

#include <iostream>

using namespace std;

struct reg {

    int startTime;*//开始时间，*

    int endTime;*//结束时间*

    bool operator<(const reg &*t*) const {

        if (startTime != t.startTime) return startTime < t.startTime;

    }

};

reg Time[100];

reg ans[100];

int tot;

int A, B, N;

void f(int *t*) {

    int t1 = t / 60;

    if (t1 < 10) cout << 0;

    cout << t1;

    cout << ":";

    int t2 = t % 60;

    if (t2 < 10) cout << 0;

    cout << t2;

}

int a, b;

char c;

int main() {

    while (cin >> A >> B >> N) {*// A至少休息的时间  B最多睁眼的时间*

        A \*= 60, B \*= 60;

        tot = 0;

        int ret = 1;*//可以*

        memset(Time, 0, sizeof(Time));

        memset(ans, 0, sizeof(ans));

        for (int i = 0; i < N; i++) {

            cin >> a >> c >> b;

            Time[i].startTime = a \* 60 + b;

            cin >> c;

            cin >> a >> c >> b;

            Time[i].endTime = a \* 60 + b;

            if (Time[i].startTime > Time[i].endTime) Time[i].endTime += 24 \* 60;

            if (Time[i].endTime - Time[i].startTime > B) {

                ret = 0;

                break;

            }

        }

        sort(Time, Time + N);

        reg last;

        last.startTime = Time[0].startTime, last.endTime = Time[0].endTime;

        for (int i = 1; i < N; i++) {

            int tmp = Time[i].startTime - last.endTime - 1;

            if (tmp >= A) {*//可以睡觉*

                ans[tot].startTime = last.endTime + 1;

                ans[tot].endTime = Time[i].startTime - 1;

                last = Time[i];

                tot++;

            } else {*//不可以睡觉*

                last.endTime = Time[i].endTime;

                if (last.endTime - last.startTime + 1 > B) {*//超时*

                    ret = 0;

                    break;

                }

            }

        }

        if (!ret) {

            cout << "No" << endl;

            continue;

        }

*//检查交界处*

        if (Time[0].startTime + 24 \* 60 - last.endTime - 1 >= A) {*//可以睡觉*

            ans[tot].startTime = (last.endTime + 1) % (60 \* 24);

            ans[tot].endTime =

                (Time[0].startTime + 60 \* 24 - 1) % (60 \* 24);*//防止是负数*

            tot++;

        } else if (tot == 0 || (ans[0].startTime - 1 + 60 \* 24) % (60 \* 24) -

                                       last.startTime + 1 >

                                   B) {*//不可以休息*

            cout << "No" << endl;

            continue;

        }

        cout << "Yes" << endl;

        cout << tot << endl;

        for (int i = 0; i < tot; i++) {

            f(ans[i].startTime);

            cout << "-";

            f(ans[i].endTime);

            cout << endl;

        }

    }

    return 0;

}

**问题描述**

|  |  |
| --- | --- |
| 试题编号： | 201609-3 |
| 试题名称： | 炉石传说 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |
| 问题描述： | **问题描述**  　　《炉石传说：魔兽英雄传》（Hearthstone: Heroes of Warcraft，简称炉石传说）是暴雪娱乐开发的一款集换式卡牌游戏（如下图所示）。游戏在一个战斗棋盘上进行，由两名玩家轮流进行操作，本题所使用的炉石传说游戏的简化规则如下：  　　\* 玩家会控制一些**角色**，每个角色有自己的**生命值**和**攻击力**。当生命值小于等于 0 时，该角色**死亡**。角色分为**英雄**和**随从**。 　　\* 玩家各控制一个英雄，游戏开始时，英雄的生命值为 30，攻击力为 0。当英雄死亡时，游戏结束，英雄未死亡的一方获胜。 　　\* 玩家可在游戏过程中召唤随从。棋盘上每方都有 7 个可用于放置随从的空位，从左到右一字排开，被称为**战场**。当随从死亡时，它将被从战场上移除。 　　\* 游戏开始后，两位玩家轮流进行操作，每个玩家的连续一组操作称为一个**回合**。 　　\* 每个回合中，当前玩家可进行零个或者多个以下操作： 　　1) **召唤随从**：玩家召唤一个随从进入战场，随从具有指定的生命值和攻击力。 　　2) **随从攻击**：玩家控制自己的某个随从攻击对手的英雄或者某个随从。 　　3) **结束回合**：玩家声明自己的当前回合结束，游戏将进入对手的回合。该操作一定是一个回合的最后一个操作。 　　\* 当随从攻击时，攻击方和被攻击方会同时对彼此造成等同于自己攻击力的**伤害**。受到伤害的角色的生命值将会减少，数值等同于受到的伤害。例如，随从 *X* 的生命值为 *HX*、攻击力为 *AX*，随从 *Y* 的生命值为 *HY*、攻击力为 *AY*，如果随从 *X* 攻击随从 *Y*，则攻击发生后随从 *X* 的生命值变为 *HX* - *AY*，随从 *Y* 的生命值变为 *HY* - *AX*。攻击发生后，角色的生命值可以为负数。 　　本题将给出一个游戏的过程，要求编写程序模拟该游戏过程并输出最后的局面。  **输入格式**  　　输入第一行是一个整数 *n*，表示操作的个数。接下来 *n* 行，每行描述一个操作，格式如下： 　　<action> <arg1> <arg2> ... 　　其中<action>表示操作类型，是一个字符串，共有 3 种：summon表示召唤随从，attack表示随从攻击，end表示结束回合。这 3 种操作的具体格式如下： 　　\* summon <position> <attack> <health>：当前玩家在位置<position>召唤一个生命值为<health>、攻击力为<attack>的随从。其中<position>是一个 1 到 7 的整数，表示召唤的随从出现在战场上的位置，原来该位置及右边的随从都将顺次向右移动一位。 　　\* attack <attacker> <defender>：当前玩家的角色<attacker>攻击对方的角色 <defender>。<attacker>是 1 到 7 的整数，表示发起攻击的本方随从编号，<defender>是 0 到 7 的整数，表示被攻击的对方角色，0 表示攻击对方英雄，1 到 7 表示攻击对方随从的编号。 　　\* end：当前玩家结束本回合。 　　注意：随从的编号会随着游戏的进程发生变化，当召唤一个随从时，玩家指定召唤该随从放入战场的位置，此时，原来该位置及右边的所有随从编号都会增加 1。而当一个随从死亡时，它右边的所有随从编号都会减少 1。任意时刻，战场上的随从总是从1开始连续编号。  **输出格式**  　　输出共 5 行。 　　第 1 行包含一个整数，表示这 *n* 次操作后（以下称为 *T* 时刻）游戏的胜负结果，1 表示先手玩家获胜，-1 表示后手玩家获胜，0 表示游戏尚未结束，还没有人获胜。 　　第 2 行包含一个整数，表示 *T* 时刻先手玩家的英雄的生命值。 　　第 3 行包含若干个整数，第一个整数 *p* 表示 *T* 时刻先手玩家在战场上存活的随从个数，之后 *p* 个整数，分别表示这些随从在 *T* 时刻的生命值（按照从左往右的顺序）。 　　第 4 行和第 5 行与第 2 行和第 3 行类似，只是将玩家从先手玩家换为后手玩家。  **样例输入**  8 summon 1 3 6 summon 2 4 2 end summon 1 4 5 summon 1 2 1 attack 1 2 end attack 1 1  **样例输出**  0 30 1 2 30 1 2  **样例说明**  　　按照样例输入从第 2 行开始逐行的解释如下： 　　1. 先手玩家在位置 1 召唤一个生命值为 6、攻击力为 3 的随从 A，是本方战场上唯一的随从。 　　2. 先手玩家在位置 2 召唤一个生命值为 2、攻击力为 4 的随从 B，出现在随从 A 的右边。 　　3. 先手玩家回合结束。 　　4. 后手玩家在位置 1 召唤一个生命值为 5、攻击力为 4 的随从 C，是本方战场上唯一的随从。 　　5. 后手玩家在位置 1 召唤一个生命值为 1、攻击力为 2 的随从 D，出现在随从 C 的左边。 　　6. 随从 D 攻击随从 B，双方均死亡。 　　7. 后手玩家回合结束。 　　8. 随从 A 攻击随从 C，双方的生命值都降低至 2。  **评测用例规模与约定**  　　\* 操作的个数0 ≤ *n* ≤ 1000。 　　\* 随从的初始生命值为 1 到 100 的整数，攻击力为 0 到 100 的整数。 　　\* 保证所有操作均合法，包括但不限于： 　　1) 召唤随从的位置一定是合法的，即如果当前本方战场上有 *m* 个随从，则召唤随从的位置一定在 1 到 *m* + 1 之间，其中 1 表示战场最左边的位置，*m* + 1 表示战场最右边的位置。 　　2) 当本方战场有 7 个随从时，不会再召唤新的随从。 　　3) 发起攻击和被攻击的角色一定存在，发起攻击的角色攻击力大于 0。 　　4) 一方英雄如果死亡，就不再会有后续操作。 　　\* 数据约定： 　　前 20% 的评测用例召唤随从的位置都是战场的最右边。 　　前 40% 的评测用例没有 attack 操作。 　　前 60% 的评测用例不会出现随从死亡的情况。 |

思路：

1.使用vector类来保存角色。

2.角色的插入和删除操作借助vector的insert和erase方法实现。

3.最后书橱两个vector容器中随从的生命值

代码：

#include <iostream>

#include <string>

#include <vector>

using namespace std;

struct follower {

    int hp;

    int attack;

    follower(int *h*, int *a*) : hp(h), attack(a) {}

};

vector<follower> player[2];

int N;

int main() {

    while (cin >> N) {

        int pid = 0;

        player[0].push\_back(follower(30, 0));

        player[1].push\_back(follower(30, 0));

        for (int n = 0; n < N; n++) {

            string type;

            cin >> type;

            if (type == "summon") {

             int pos, h, a;

             cin >> pos >> a >> h;

             player[pid].insert(player[pid].begin() + pos, follower(h, a));

            } else if (type == "attack") {

                int att, deff;

                cin >> att >> deff;

                player[pid][att].hp -= player[!pid][deff].attack;

                player[!pid][deff].hp -= player[pid][att].attack;

                if (player[pid][att].hp <= 0 && att != 0) {

                    player[pid].erase(player[pid].begin() + att);

                }

                if (player[!pid][deff].hp <= 0 && deff != 0) {

                    player[!pid].erase(player[!pid].begin() + deff);

                }

            } else if (type == "end") {

                pid = !pid;

            }

        }

        if (player[0][0].hp > 0 && player[1][0].hp > 0)

            cout << 0 << endl;

        else if (player[0][0].hp > 0)

            cout << 1 << endl;

        else

            cout << -1 << endl;

        for (int i = 0; i < 2; i++) {

            cout << player[i][0].hp << endl;

            cout << player[i].size() - 1 << " ";

            for (int j = 1; j < player[i].size(); j++) {

                cout << player[i][j].hp << " ";

            }

            cout << endl;

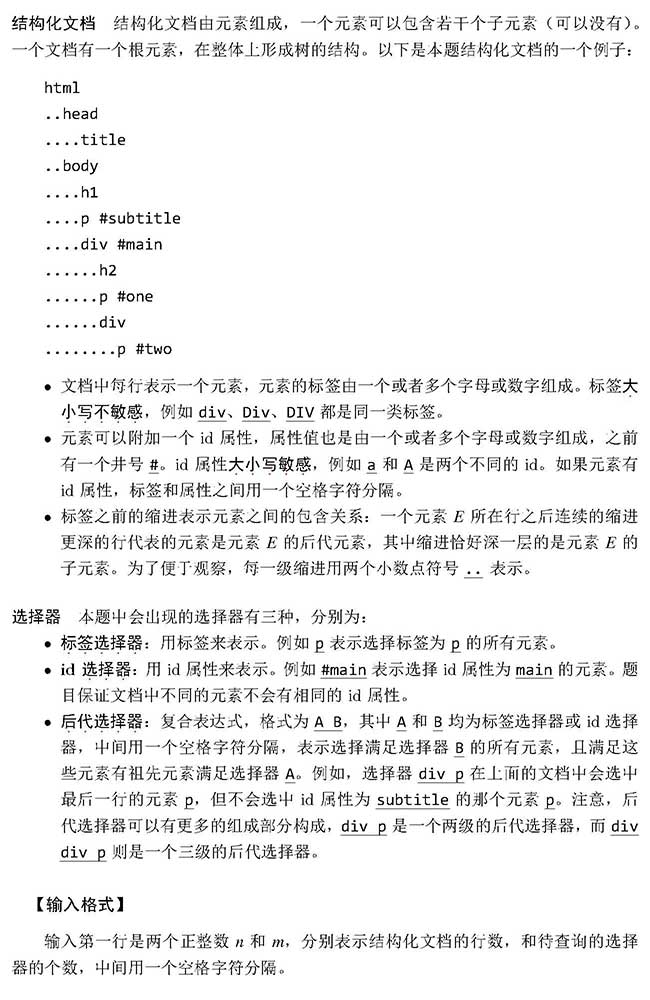
        }

    }

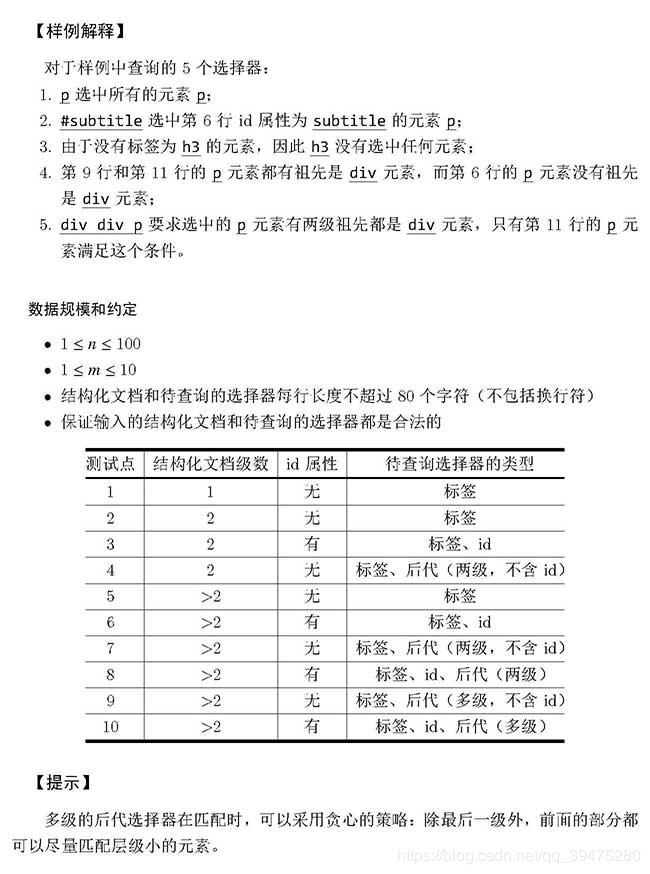
}

|  |  |
| --- | --- |
| 试题编号： | 201809-3 |
| 试题名称： | 元素选择器 |
| 时间限制： | 1.0s |
| 内存限制： | 256.0MB |









思路；

创建节点结构体，包含元素，属性和所有子节点。

元素和 id属性都可以通过字符串处理获得，也可能没有id属性。

用树型结构存储html文档，然后用dfs的方式进行搜索。

后代选择器进行匹配时两个相邻元素A,B，A只要是B的祖先就行了

一个匹配器匹配一个元素，不存在标签匹配器和id匹配器共同匹配一个标签的情况，所以可以使用dfs

代码：

#include "iostream"

#include "vector"

using namespace std;

const int maxn = 105;

struct node {

    string ele, id;

    vector<int> child;

};

int num[maxn];

node html[maxn], qry[maxn];

int len;

vector<int> ans;

void dfs(int *u*, int *d*) {

    bool flag = 0;

    if (qry[d].ele != "") {

        flag = (qry[d].ele == html[u].ele);

    } else if (qry[d].id != "") {

        flag = (qry[d].id == html[u].id);

    }

    if (flag) {

        if (d + 1 < len) {

            d++;

        } else

            ans.push\_back(u);

    }

    for (int i = 0; i < html[u].child.size(); i++) {

        int v = html[u].child[i];

        dfs(v, d);

    }

}

int main() {

    int n, m;

    ios::sync\_with\_stdio(0);

    while (cin >> n >> m) {

        cin.get();

        for (int i = 0; i < n; i++) {

            string s;

            getline(cin, s);

            int d = 0;

            node tmp;

            tmp.ele = "", tmp.id = "";

            for (int j = 0; j < s.length(); j++) {

                if (s[j] == '.') {

                    j++;

                    d++;

                } else if (s[j] == ' ')

                    continue;

                else if (s[j] != '#') {

                    num[d] = i;

                    while (j < s.length() && s[j] != ' ') {

                        tmp.ele += tolower(s[j]);

                        j++;

                    }

                } else {

                    while (j < s.length()) {

                        tmp.id += s[j];

                        j++;

                    }

                }

            }

            html[i] = tmp;

            if (d > 0) {

                html[num[d - 1]].child.push\_back(i);

            }

        }

        while (m--) {

            string s;

            getline(cin, s);

            len = 0;

            for (int i = 0; i < maxn; i++) qry[i].id = qry[i].ele = "";

            for (int i = 0; i < s.length(); i++) {

              if (s[i] == '#') {

                while (i<s.length() && s[i] != ' ') qry[len].id += s[i++];

                    len++;

                } else if (s[i] == ' ')

                    continue;

                else {

                    while (i < s.length() && s[i] != ' ') {

                        qry[len].ele += tolower(s[i]);

                        i++;

                    }

                    len++;

                }

            }

            ans.clear();

            dfs(0, 0);

            cout << ans.size();

            for (int i = 0; i < ans.size(); i++) {

                cout << " ";

                cout << ans[i] + 1;

            }

            cout << "\n";

        }

    }

    return 0;

}

# week12\_CSP\_M3



思路：

读入的同时判断与上一次读入的是否相同，不同则段数+1，最后输出段数

代码：

#include "iostream"

using namespace std;

int main() {

    int n, a[1010];

    while (cin >> n) {

        int cnt = 1;

        for (int i = 0; i < n; i++) {

            cin >> a[i];

            if (i > 0) {

                if (a[i]!= a[i - 1]) {

                    cnt++;

                }

            }

        }

        cout<<cnt<<endl;

    }

}

T2 消消乐大师——Q老师

题目描述

Q老师是个很老实的老师，最近在积极准备考研。Q老师平时只喜欢用Linux系统，所以Q老师的电 脑上没什么娱乐的游戏，所以Q老师平时除了玩Linux上的赛车游戏SuperTuxKart之外，就是喜欢 消消乐了。 游戏在一个包含有 n 行 m 列的棋盘上进行，棋盘的每个格子都有一种颜色的棋子。当一行或一列 上有连续三个或更多的相同颜色的棋子时，这些棋子都被消除。当有多处可以被消除时，这些地 方的棋子将同时被消除。 一个棋子可能在某一行和某一列同时被消除。 由于这个游戏是闯关制，而且有时间限制，当Q老师打开下一关时，Q老师的好哥们叫Q老师去爬 泰山去了，Q老师不想输在这一关，所以它来求助你了！！

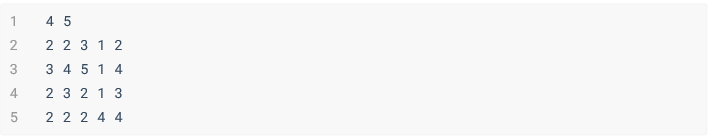
输入描述

输入第一行包含两个整数n,m，表示行数和列数 接下来n行m列，每行中数字用空格隔开，每个数字代表这个位置的棋子的颜色。数字都大于0.

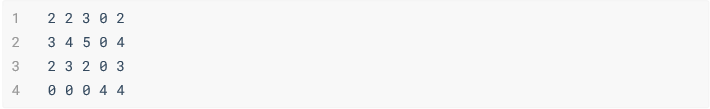
输出描述

输出n行m列，每行中数字用空格隔开，输出消除之后的棋盘。（如果一个方格中的棋子被消除， 则对应的方格输出0，否则输出棋子的颜色编号。）

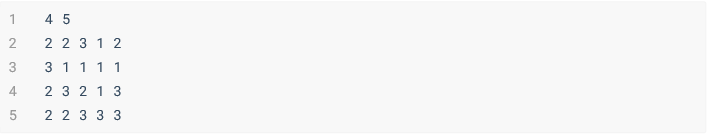
样例输入1



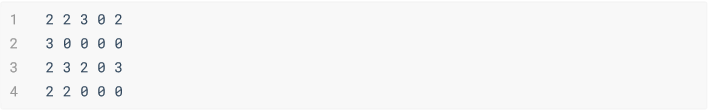
样例输出1



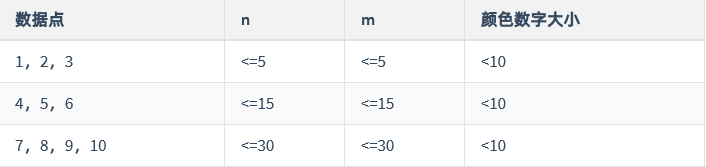
样例输入2



样例输出2



数据组成



**思路：**

两个二维矩阵，分别是初始游戏棋盘和消除后的游戏棋盘，一开始让结果棋盘与初始棋盘相同。

分别按照行方向和列方向遍历初始棋盘，在结果棋盘中完成消除之后的棋盘结果（确定哪些位置要被改成0）。最后输出结果棋盘矩阵。

代码：

#include "iostream"

using namespace std;

int a[50][50], b[50][50];

int n, m;

int main() {

    while (cin >> n >> m) {

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < m; j++) {

                cin >> a[i][j];

                b[i][j] = a[i][j];

            }

        }

        int ct = 1;

        for (int i = 0; i < n; i++) {

            for (int j = 1; j < m; j++) {

                if (a[i][j] == a[i][j - 1]) {

                    ct++;

                    if (j == m - 1 && ct > 2) {

                        for (int p = j; p > (j - ct); p--) {

                            b[i][p] = 0;

                        }

                    }

                    continue;

                }

                if (ct > 2) {

                    for (int p = j - 1; p > (j - 1 - ct); p--) {

                        b[i][p] = 0;

                    }

                    ct = 1;

                    continue;

                }

                ct = 1;

            }

            ct = 1;

        }

        ct = 1;

        for (int j = 0; j < m; j++) {

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

                if (a[i][j] == a[i-1][j ]) {

                    ct++;

                    if (i == n - 1 && ct > 2) {

                        for (int p = i; p > (i - ct); p--) {

                            b[p][j] = 0;

                        }

                    }

                    continue;

                }

                if (ct > 2) {

                    for (int p = i - 1; p > (i - 1 - ct); p--) {

                        b[p][j] = 0;

                    }

                    ct = 1;

                    continue;

                }

                ct = 1;

            }

            ct = 1;

        }

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < m - 1; j++) {

                cout << b[i][j] << " ";

            }

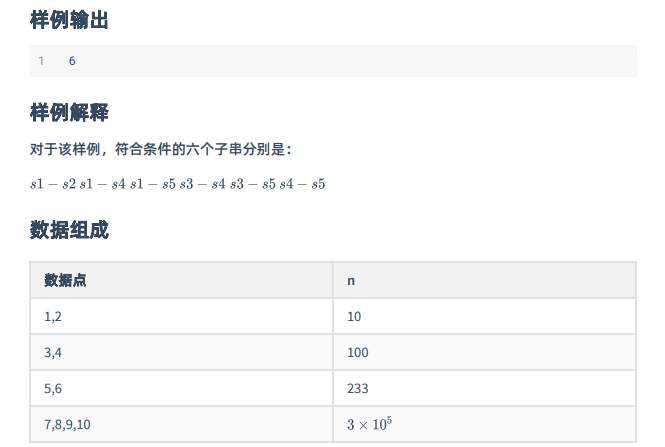
            cout << b[i][m - 1] << endl;

        }

    }

}





思路：

题目要求含有长度大于1的回文串即为符合要求的字符串，而且字符串中只有A，B两个字母，所以只有AAAAB或者BAAAA或者ABBBB或者BBBBA这种形式字符串是不符合要求的,其他的字串都是符合要求的。所以求出这些字符串的数量即可。

代码：

#include "cstring"

#include "iostream"

using namespace std;

string s;

int main() {

    long long n, ct = 0, last = 0, num = 0;

    while (cin >> n) {

        cin >> s;

        for (int i = 0; i < n; i++) {

            if (i && s[i] != s[i - 1])

            {

                num += ct;

                ct = 0;

                last = i;

            } else if (last)

            {

                num++;

            }

            ct++;

        }

        long long sum = n \* (n - 1) / 2;

        cout<<sum-num<<endl;

        ct = 0, last = 0, num = 0;

    }

    return 0;

}

# week16\_CSP\_M4



思路：

将数以字符串形式读入，直接计算每个字符串的不同字符的个数，再与k比较。

关了同步就不超时了，奇怪

代码：

#include <algorithm>

#include <iostream>

#include <string>

using namespace std;

int n, k, ans, t;

int main() {

    string str;

    ios::sync\_with\_stdio(false);

    while (cin >> n >> k) {

        if (k > 10) {

            for (int i = 1; i <= n; i++) cin >> t;

            cout << n;

        } else {

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

                cin >> str;

                sort(str.begin(), str.end());

                int cnt = 1;

                for (int j = 0; j < str.size() - 1; j++) {

                    if (str[j + 1] != str[j]) cnt++;

                }

                if (cnt < k) ans++;

            }

            cout << ans;

        }

        str.clear();

    }

    return 0;

}



思路：

遍历每个点到其他所有点距离中的最大值，然后再取最大值，得到点和半径（点对应的最大距离）

当时看错题了，没注意到点在给出的点里面，然后就做复杂了

还有，小心爆int

代码：

#include <algorithm>

#include <cmath>

#include <iostream>

using namespace std;

long long n;

struct \_point {

    long long x;

    long long y;

    bool operator<(\_point *t*) {

        if (x == t.x) return y < t.y;

        return x < t.x;

    }

} p[2000];

struct set {

    long long x;

    long long y;

    long long r;

    bool operator<(set *t*) {

        if (r != t.r) return r < t.r;

        if (x != t.x) return x < t.x;

        return y < t.y;

    }

} ans[2000];

int main() {

    ios::sync\_with\_stdio(false);

    while (cin >> n) {

        for (long long i = 1; i <= n; i++) {

            cin >> p[i].x >> p[i].y;

        }

        sort(p + 1, p + n + 1);

        for (long long i = 1; i <= n; i++) {

            long long rr = 0;

            for (long long j = 1; j <= n; j++) {

                long long t = (long long)(p[i].x - p[j].x) \* (p[i].x - p[j].x) +

                        (long long)(p[i].y - p[j].y) \* (p[i].y - p[j].y);

                if (rr < t) {

                    rr = t;

                }

            }

            ans[i].r = rr;

            ans[i].x = p[i].x;

            ans[i].y = p[i].y;

        }

        sort(ans + 1, ans + 1 + n);

        cout << ans[1].x << ".00 " << ans[1].y << ".00\n";

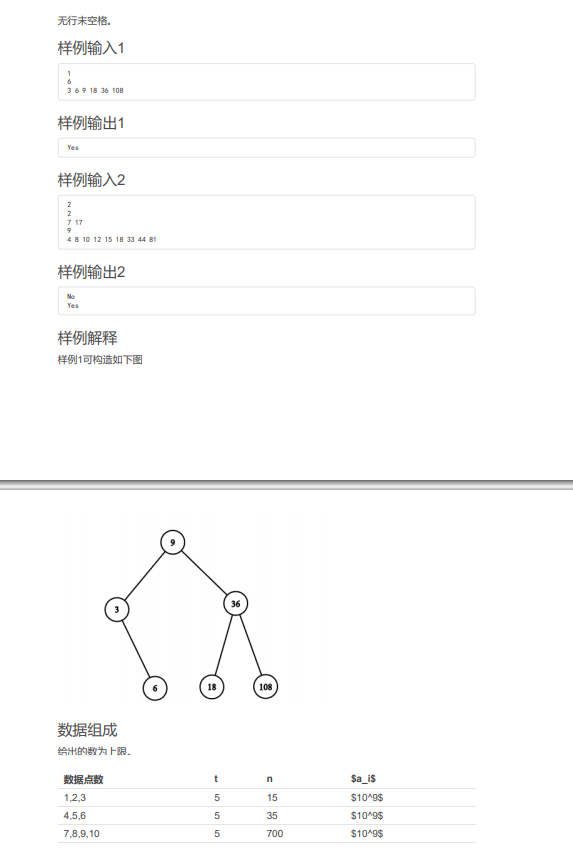
        cout << ans[1].r << ".00" << endl;

    }

    return 0;

}





思路：

用一个二维数组记录每两个数之间的最大公约数。对于升序序列a，子问题为a[i]-a[j]是否能够成一棵二叉平衡搜索树。然后可以将子问题再分为a[i]-a[k]的左子树和a[k]-a[j]右子树是否存在。枚举区间内所有点作为根节点，设a[k]为根节点，l[i][k]为1表示可以生成左子树，r[k][j]为1表示可以生成右子树，那么当两者皆为1时，f[i][j]等于1表示可以生成一棵二叉搜索树。

代码：

#include <cstring>

#include <iostream>

using namespace std;

int t, n;

int a[710];

int \_gcd[710][710], l[710][710], r[710][710], f[710][710];

int gcd(int *a*, int *b*) { return b == 0 ? a : gcd(b, a % b); }

void init() {

    memset(a, 0, sizeof(a));

    memset(\_gcd, 0, sizeof(\_gcd));

    memset(l, 0, sizeof(l));

    memset(r, 0, sizeof(r));

    memset(f, 0, sizeof(f));

}

void init1() {

    for (int i = 1; i <= n; i++) {

        for (int j = 1; j <= n; j++) {

            if (gcd(a[i], a[j]) > 1) {

                \_gcd[i][j] = 1;

            }

        }

    }

    for (int i = 1; i <= n; i++) {

        l[i][i] = 1;

        r[i][i] = 1;

        f[i][i] = 1;

    }

}

int main() {

    while (cin >> t) {

        for (int ii = 1; ii <= t; ii++) {

            cin >> n;

            init();

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

                cin >> a[i];

            }

            init1();

            for (int i = n; i >= 1; i--) {

                for (int rr = i; rr <= n; rr++) {

                    for (int k = i; k <= rr; k++) {

                        if (l[i][k] == 1 && r[k][rr] == 1) {

                            f[i][rr] = 1;

                            if (\_gcd[k][rr + 1]) {

                                l[i][rr + 1] = 1;

                                f[i][rr + 1] = 1;

                            }

                            if (\_gcd[i - 1][k]) {

                                r[i - 1][rr] = 1;

                                f[i - 1][rr] = 1;

                            }

                        }

                    }

                }

            }

            if (f[1][n])

                cout << "Yes\n";

            else

                cout << "No\n";

            //system("pause");

        }

    }

    return 0;

}