2016级数据结构第一次上机解题报告（助教版）

A:

A题只要按照题目要求做就行，因为按值传递是不会改变原来变量的值的，而传入引用和指针都会改变原来的数据，所以第一行输出是a，b，其余两行是b，a，

参考代码：

#include <iostream>

void swap1(int a, int b) {

int hold = a;

a = b;

b = hold;

}

void swap2(int& a, int& b) {

int hold = a;

a = b;

b = hold;

}

void swap3(int\* a, int\* b) {

int hold = \*a;

\*a = \*b;

\*b = hold;

}

int main() {

int a, b;

while(std::cin >> a >> b) {

int a1 = a, b1 = b;

swap1(a1, b1);

std::cout << a1 << " " << b1 << "\n";

int a2 = a, b2 = b;

swap2(a2, b2);

std::cout << a2 << " " << b2 << "\n";

int a3 = a, b3 = b;

swap3(&a3, &b3);

std::cout << a3 << " " << b3 << "\n";

}

}

B

一道大水题。

只需要找到“中间巧克力”，即从此块巧克力开始，吃掉此块巧克力（包括此块）之前所有巧克力的时间和\*2>吃掉所有巧克力的总时间。然后只需比较这块巧克力被谁吃，如果吃掉这块巧克力左面的巧克力所需要的时间<吃掉这块巧克力右面的巧克力所需要的时间，则这块巧克力被Syw吃，反之被Gzh吃，详情请看代码。

#include <iostream>

using namespace std;

long long sum[100001];

int main()

{

int n,pos;//pos记录中间巧克力的位置

while(cin>>n)

{

cin>>sum[1];

for(int i=2;i<=n;i++)

{

cin>>sum[i];

sum[i]+=sum[i-1];

}//sum记录吃掉前n块巧克力所需的时间

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

if(sum[i]\*2>=sum[n])

{

pos=i;

break;

}

if(sum[n]-sum[pos]<sum[pos-1])

pos--;

cout<<pos<<' '<<n-pos<<endl;

}

return 0;

}

C：

有些同学可能没有读懂，在这里把样例解释清楚一些，一开始给他们标号，分别为1-3号男生，1-3号女生，排完队后男生队列是2 1 3，女生队列是1 3 2，所以先输出2号男生对应舞伴即2号女生的位置：3，然后输出排在第二位的男生即1号男生对应的女生1号，而女生1号排在女生队伍里的第一个，所以输出1，其实这道题暴力找是可以做的，还可以按照hint的思路，先给每个人标一个序号，然后排序，然后输出每个人对应的舞伴的位置。

参考代码：

#include<cstdio>

#include<algorithm>

const int maxt = 10001;

struct student {

double h;

int i;

bool operator <(student const &o) const {

return h < o.h;

}

};

int position[maxt];

student girl[maxt], boy[maxt];

int main() {

int n;

while(scanf("%d", &n) == 1) {

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

scanf("%lf %lf", &girl[i].h, &boy[i].h);

girl[i].i = boy[i].i = i;

}

std::sort(girl + 1, girl + 1 + n);

std::sort(boy + 1, boy + 1 + n);

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

position[boy[i].i] = i;

}

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

printf("%d ",position[girl[i].i]);

}

printf("\n");

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

position[girl[i].i] = i;

}

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

printf("%d ",position[boy[i].i]);

}

printf("\n");

}

return 0;

}

D：

D题框架和练习赛的差不多，只不过多了一个查询，而这个查询只需要遍历数组就可以，并不需要二分法，不过注意找到的时候记得break,不然会TLE，而且大家要熟悉scanf，printf的输入输出方式，不然会TLE。

参考代码：

#include <cstdio>

#include <cmath>

#include <algorithm>

#define epsilon 1e-6

struct student {

int id;

char name[20];

double grade;

bool operator < (student const & other) const {

if (fabs(grade - other.grade) < epsilon) {

return id < other.id;

}

return grade > other.grade;

}

};

student students[10050];

int main() {

int n, m;

while(scanf("%d", &n) == 1) {

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

student student1;

scanf("%s %d %lf", student1.name, &student1.id, &student1.grade);

students[i] = student1;

}

std::sort(students, students + n);

scanf("%d", &m);

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

int id;

scanf("%d", &id);

bool isFound = false;

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

if (id == students[i].id) {

isFound = true;

student student1 = students[i];

printf("%s %d %.2f %d\n", student1.name, student1.id, student1.grade, i + 1);

break;

}

}

if (!isFound) {

printf("Only god knows where he is.\n");

}

}

}

}

E

递归。

找到递归关系，我们不妨令f(m,n)表示m个人进入n个教室有多少种进法，下面根据不同情况分类：

(1):当教室数为1的时候，只有一种情况，即所有人在一个教室。

(2):当学生数为1的时候，也只有一种情况，注意题目中说明，教室之间并无顺序,所以不管这个学生去哪个教室，结果都算一个。

(3):当m<n时，因为此时最多只能去m个教室（一个教室去一个），实际上就相当于m个学生去到m个教室里一样，也就是f(m,m);

(4):当m==n时,此时分两种情况讨论，一种是一个教室里去一个，只是一种，第二种是，至少有一个教室里没有学生这就相当于是f(m,m-1);

(5):当m>n时，也分两种情况讨论，一种是至少有一个教室里没学生，这样子就相当于f(m,n-1),第二种是，每个教室先去一个学生，剩下的m-n个学生再进入到n个教室里去，即f(m-n,n);

综上所述:

得到递归表达式：

f(m,n)=1 当 m=1或n=1；

f(m,n)=f(m,m) 当m<n;

f(m,n)=1+f(m,m-1) 当m=n;

f(m,n)=f(m-n,n)+f(m,n-1);

在递归的过程中采用记忆化搜索可以减少不必要的时间，算过的东西就不要再算了。

详情请看代码：

#include <cstdio>

#include <cstring>

int f[123][123];

int GzhIsSoHandsome(int m,int n)

{

if(f[m][n])//避免重复搜索

return f[m][n];

else if(m==1||n==1)

{

f[m][n]=1;

return f[m][n];

}

else if(m<n)

{

f[m][n]=GzhIsSoHandsome(m,m);

return f[m][n];

}

else if(m==n)

{

f[m][n]=1+GzhIsSoHandsome(m,m-1);

return f[m][n];

}

else

{

f[m][n]=GzhIsSoHandsome(m-n,n)+GzhIsSoHandsome(m,n-1);

return f[m][n];

}

}

int main()

{

int t,m,n;

scanf("%d",&t);

while(t--)

{

memset(f,0,sizeof(f));

scanf("%d%d",&m,&n);

printf("%d\n",GzhIsSoHandsome(m,n));

}

return 0;

}

F

<问题描述>

所谓虫食算，就是原先的算式中有一部分被虫子啃掉了，需要我们根据剩下的数字来判定被啃掉的字母。来看一个简单的例子：

43#9865#045

+ 8468#6633

其中#号代表被虫子啃掉的数字。根据算式，我们很容易判断：第一行的两个数字分别是5和3，第二行的数字是5。

现在，我们对问题做两个限制：

首先，我们只考虑加法的虫食算。这里的加法是N进制加法，算式中三个数都有N位，允许有前导的0。

其次，虫子把所有的数都啃光了，我们只知道哪些数字是相同的，我们将相同的数字用相同的字母表示，不同的数字用不同的字母表示。如果这个算式是N进制的，我们就取英文字母表午的前N个大写字母来表示这个算式中的0到N-1这N个不同的数字：但是这N个字母并不一定顺序地代表0到N-1)。输入数据保证N个字母分别至少出现一次。

BADC

+ CRDA

DCCC

上面的算式是一个4进制的算式。很显然，我们只要让ABCD分别代表0123，便可以让这个式子成立了。你的任务是，对于给定的N进制加法算式，求出N个不同的字母分别代表的数字，使得该加法算式成立。输入数据保证有且仅有一组解，

- 输入文件

输入文件alpha.in包含4行。第一行有一个正整数N(N<=26)，后面的3行每行有一个由大写字母组成的字符串，分别代表两个加数以及和。这3个字符串左右两端都没有空格，从高位到低位，并且恰好有N位。

- 输出文件

输出文件alpha.out包含一行。在这一行中，应当包含唯一的那组解。解是这样表示的：输出N个数字，分别表示A，B，C……所代表的数字，相邻的两个数字用一个空格隔开，不能有多余的空格。

- 样例输入

5

ABCED

BDACE

EBBAA

- 样例输出

1 0 3 4 2

- 数据规模

对于30％的数据，保证有N<＝10；

对于50％的数据，保证有N<＝15；

对于全部的数据，保证有N<=26。

<算法分析>

经典的搜索题。最单纯的搜索的时间复杂度为O(n!)，是会非常严重的超时的。计算机是很“笨”的，它不会思考，在盲目搜索的过程中，很容易出现这种情况：

计算机在某一位搜索出了一个算式1 + 1 = 3，并且继续搜索。

明显，人眼很容易就看出这是不合法的，但计算机不会。于是，我们想到了第一个剪枝：每次搜索的时候，从最后向前判断是否有不合法的式子。

这一个剪枝非常简单，但是效果却非常的好。因为它剪去了很多不必要的搜索。为了配合这一种剪枝更好的实行，搜索顺序的改变也成为大大提高程序效率的关键：从右往左，按照字母出现顺序搜索，有很大程度上提高了先剪掉废枝的情况，使程序的效率得到大大的提高。

有了以上两个剪枝，程序就已经可以通过大部分测试点了。但是有没有更多的剪枝呢？答案是肯定的。

根据前面的剪枝，我们可以找到类似的几个剪枝：

对于a + b = c的形式，假如：

A\*\*\*?\*\*\*

+ B\*?\*\*?\*\*

C\*\*\*???\*

其中\*代表已知，?代表未知。那么，A + B与C的情况并不能直接确定。但是，假如(A + B) % N与(A + B + 1) % N都不等于C的话，那么这个等式一定是不合法的。因为它只有进位和不进位的两种情况。

同样，我们在一个数组里记录了Used[i]表示一个数字有没有用过，那么，对于某一位A + B = C的等式，如果已经得到了两个数，另一个数还待搜索的时候，我们还可以根据这个加入一个剪枝：

例如A + ? = C的形式，

考虑不进位的情况，则?处为P1 = (C - A + N) % N

假如考虑进位的情况，则?处为P2 = (C - A - 1 + N) % N

假如P1、P2均被使用过，那么这个搜索一定是无效的，可以剪去。

有了以上的剪枝，就可以很轻松地通过所有的测试数据了。当然，还有很多值得思考的剪枝以及其他的思路，例如枚举进位、解方程（但是可能需要枚举）等，在这里就不详细讨论了。

<代码清单>

#include <fstream>

#include <string>

using namespace std;

ifstream fin("alpha.in");

ofstream fout("alpha.out");

bool finish, hash[256], used[27];

int n, stk[27];

string a, b, c;

string word;

void init() {

fin >> n >> a >> b >> c;

finish = false;

}

void outsol() {

int i, ans[27];

for (i = 0; i < n; i ++)

ans[word[i] - 65] = stk[i];

fout << ans[0];

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

fout << " " << ans[i];

fout << endl;

finish = true;

}

void addup(char ch) {

if (!hash[ch]) {

hash[ch] = true;

word = word + ch;

}

}

string change(string str, char x, char y) {

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

if (str[i] == x)

str[i] = y;

return str;

}

void pre\_doing() {

word = "";

memset(hash, 0, sizeof(hash));

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

addup(a[i]); addup(b[i]); addup(c[i]);

}

memset(used, 0, sizeof(used));

}

bool bad() {

int p, g = 0;

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

if (a[i] >= n || b[i] >= n || c[i] >= n) return false;

p = a[i] + b[i] + g;

if (p % n != c[i]) return true;

g = p / n;

p %= n;

}

return false;

}

bool modcheck() {

int i, p, p1, p2, g = 0;

//a + b = c, all know

for (i = n - 1; i >= 0; i --) {

if (a[i] >= n || b[i] >= n || c[i] >= n) continue;

p = (a[i] + b[i]) % n;

if (!(p == c[i] || (p + 1) % n == c[i])) return true;

}

//a + ? = c

for (i = n - 1; i >= 0; i --) {

if (!(a[i] < n && c[i] < n && b[i] >= n)) continue;

p1 = (c[i] - a[i] + n) % n;

p2 = (p1 - 1) % n;

if (used[p1] && used[p2]) return true;

}

//? + b = c

for (i = n - 1; i >= 0; i --) {

if (!(a[i] >= n && c[i] < n && b[i] < n)) continue;

p1 = (c[i] - b[i] + n) % n;

p2 = (p1 - 1) % n;

if (used[p1] && used[p2]) return true;

}

//a + b = ?

for (i = n - 1; i >= 0; i --) {

if (!(a[i] < n && b[i] < n && c[i] >= n)) continue;

p1 = (a[i] + b[i]) % n;

p2 = (p1 + 1) % n;

if (used[p1] && used[p2]) return true;

}

return false;

}

void dfs(int l) {

int i;

string A, B, C;

if (finish) return;

if (bad()) return;

if (modcheck()) return;

if (l == n) {

outsol();

return;

}

for (i = n - 1; i >= 0; i --)

if (!used[i]) {

used[i] = true; A = a; B = b; C = c;

a = change(A, word[l], i);

b = change(B, word[l], i);

c = change(C, word[l], i);

stk[l] = i;

dfs(l + 1);

used[i] = false; a = A; b = B; c = C;

}

}

int main() {

init();

pre\_doing();

dfs(0);

return 0;

}

<小结>

　　搜索题的框架往往不难找到，关键就是在搜索的优化上，本文的主要篇幅也就是讨论了几种有效的优化。搜索问题的优化更多的需要选手的经验和思考、分析问题的能力，所以搜索剪枝也是竞赛中经久不衰的经典问题。