# 分治：众数

#include<iostream>

#include<algorithm>

using namespace std;

// 众数

int mode;

// 重数

int modeCount = 0;

int arr[10010];

void solve(int n,int left, int right)

{

// 函数递归出口

if(left > right) {

return;

}

// 划分找到左侧第一个不等于a[mid]的位置

int mid = (left + right) >> 1;

int i = mid, j = mid;

// 找到左侧第一个不等于a[mid]的位置

while(i >= 0 && arr[i] == arr[mid]) {

i--;

}

// 找到右侧第一个不等于a[mid]的位置

while(j <= n-1 && arr[j] == arr[mid]) {

j++;

}

// 众数在中间的情况

// j-i-1为中位数的重数

if(j - i - 1 > modeCount) {

mode = arr[mid];

modeCount = j - i - 1;

}

// 分治：众数可能在左侧

if(i - left + 1 > modeCount) {

solve(n,left,i);

}

// 分治：众数可能在右侧

if(right - j + 1 > modeCount) {

solve(n,j,right);

}

}

main()

{

int n;

cout << "请输入元素个数：";

cin >> n;

cout << "请输入元素值：";

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

cin >> arr[i];

}

sort(arr,arr+n);

solve(n,0,n);

cout << "众数：" << mode;

cout << "重数：" << modeCount;

return 0;

}

// 1 2 2 7 2 7 5

// 1 2 3 3 3 4 4 5

# 动态规划：最大子段和

#include <iostream>

#include <math.h>

using namespace std;

int arr[10010];

// 存储子段和数组

int dp[10010];

int index[3] = {0,0,0};

int maxValue = -99999;

int maxSum(int n)

{

dp[0] = arr[0];

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

//dp[i] = dp[i-1] > 0 ? dp[i-1] + arr[i] : arr[i];

// 如果前i-1项和大于0

if(dp[i-1] > 0) {

// 就计算前i项最大子段和

dp[i] = dp[i-1] + arr[i];

}else {

// 如果前i-1项和小于0

dp[i] = arr[i];

// 更新起始下标

index[0] = i;

}

//maxValue = max(maxValue,dp[i]);

// 新的前i项和 与 maxValue比较

if(dp[i] > maxValue) {

// 更新maxValue值

maxValue = dp[i];

// 再次记录起始下标（有可能上面的起始下标更新了，但是它不是最大子段和，所以保证起始下标不变）

index[2] = index[0];

// 记录终止下标

index[1] = i;

}

}

return maxValue;

}

int main()

{

cout << "请输入元素个数：";

int n;

cin >> n;

cout<< "请输入元素值：";

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

cin >> arr[i];

}

int res = maxSum(n);

cout << "起始下标：" << index[2] << endl;

cout << "终止下标：" << index[1] << endl;

cout << "最大子段和：" << res << endl;

return 0;

}

/\*

7

2 -4 3 -1 2 -4 3

6

-2 11 -4 13 -5 -2

\*/

# 最长公共子序列

#include <iostream>

#include <string>

using namespace std;

int dp[10010][10010];

int flag[10010][10010];

string x;

string y;

void lcs\_length(int m, int n)

{

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

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

// 如果x和y的字符相等

if(x[i-1] == y[j-1]) {

// 取对角线上的值+1

dp[i][j] = dp[i-1][j-1] + 1;

// 记录标志

flag[i][j] = -1;

}else if(dp[i-1][j] >= dp[i][j-1]) {

// 如果x和y不相等，那么就取相邻（左上）的最大值

dp[i][j] = dp[i-1][j];

flag[i][j] = -2;

}else {

dp[i][j] = dp[i][j-1];

flag[i][j] = -3;

}

}

}

}

void print\_lcs(int i, int j) {

if( i == 0 || j == 0) {

return;

}

// 根据条件依次入栈

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

print\_lcs(i-1,j-1);

// 只有符合相等条件-1时，才能够输出，其它的弹栈后无操作。

cout << x[i-1];

}else if(flag[i][j] == -2) {

// i-1大，走的-2

print\_lcs(i-1,j);

}else {

// j-1大，走的-3

print\_lcs(i,j-1);

}

/\*

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

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

if(flag[n][m] == -1) {

// 用x，n代表x的长度，所以是n-1，而不是m-1，

// 因为n表示x字符串的位置

// 如果用y，m则是y[m-1]，同理。

cout << x[n-1];

}

}

} \*/

}

int main()

{

cout << "请输入字符串x：";

cin >> x;

cout << "请输入字符串y：";

cin >> y;

// 表格初始化

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

dp[i][0] = 0;

}

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

dp[0][j] = 0;

}

lcs\_length(x.length(), y.length());

cout << "公共子序列：";

print\_lcs(x.length(), y.length());

cout <<"" <<endl;

int j = 0;

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

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

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

}

if(j == y.length()+1) {

cout <<"" <<endl;

}

}

return 0;

}

/\*

ALCHEMIST

ALGORITHMS

A L C H E M I S T

0 0 0 0 0 0 0 0 0 0

A 0 1 1 1 1 1 1 1 1 1

L 0 1 2 2 2 2 2 2 2 2

G 0 1 2 2 2 2 2 2 2 2

O 0 1 2 2 2 2 2 2 2 2

R 0 1 2 2 2 2 2 2 2 2

I 0 1 2 2 2 2 2 3 3 3

T 0 1 2 2 2 2 2 3 3 4

H 0 1 2 2 3 3 3 3 3 4

M 0 1 2 2 3 3 4 4 4 4

S 0 1 2 2 3 3 4 4 5 5

\*/

# 贪心：背包

#include <iostream>

using namespace std;

// 存放物品重量

double w[10010];

// 存放物品价值

double v[10010];

// 问题的解

double x[10010];

// 根据单位价值排序

void Sort(int n)

{

int i,j;

double temp1,temp2;

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

for(j = 0; j < n-i; j++)//冒泡排序

{

temp1 = v[j] / w[j];

temp2 = v[j+1] / w[j+1];

if(temp1 < temp2)

{

swap(w[j], w[j+1]);

swap(v[j], v[j+1]);

}

}

}

int main()

{

double maxValue = 0;

int n;

int C;

cout << "请输入物品的数量和背包最大容量：";

cin >> n;

cin >> C;

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

cout << "请输入物品重量和价值：";

cin >> w[i] >> v[i];

}

Sort(n);

int i = 0;

while(w[i] < C) {

x[i] = 1; // 将第i个物品放入背包

C = C - w[i];

maxValue=maxValue+v[i];

i++;

}

if(i < n) {

x[i] = C / w[i];

maxValue=maxValue+x[i]\*v[i];

}

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

cout << "重量为：" << w[j] << " " << "价值为：" << v[j] << " " << "放入比例：" << x[j] << endl;

}

cout<<"放入背包物品的总价值为"<<maxValue<<endl;

return 0;

}

/\*

5 50

10 70

20 60

30 50

40 40

50 30

\*/

# 活动安排

#include <iostream>

using namespace std;

// 活动开始时间

int s[10010];

// 活动结束时间

int f[10010];

// 活动序列号

int num[10010];

// 是否选择该活动

bool flag[10010];

// 根据结束时间进行排序

void Sort(int n)

{ int i,j;

for(i=0;i<n-1;i++)

for(j=0;j<n-i-1;j++)

{

if(f[j]>f[j+1])

{

swap(s[j], s[j+1]);

swap(f[j], f[j+1]);

swap(num[j], num[j+1]);

}

}

}

void activity(int n)

{

int j = 0;

// 第一个肯定选

flag[num[0]] = true;

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

// 下一个活动的起始时间 > 当前活动的结束时间，则选择此活动

if(s[i] >= f[j]) {

flag[i] = true;

j = i;

}else {

flag[i] = false;

}

}

}

int main()

{

int n;

cout << "请输入活动个数：";

cin >> n;

cout << "请输入活动开始时间和结束时间：";

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

cin >> s[i] >> f[i];

num[i] = i;

}

Sort(n);

activity(n);

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

if(flag[i]) {

cout << "能举办的活动序号：" << num[i] << " " << "活动开始时间：" << s[i] << " " << "活动结束时间：" << f[i] << endl;

}

}

return 0;

}

/\*

11

1 4

3 5

0 6

5 7

3 8

5 9

6 10

8 11

8 12

2 13

12 14

\*/

# 回溯：N皇后

#include <iostream>

#include <vector>

#include <string>

#include <cmath>

using namespace std;

// board表示棋盘上每行皇后所在的列。第i行再第board[i]列

// 当前要放置皇后的行数row、列数col。

// 判断是否存在同列、同对角线的皇后。（行的差等于列的差 | 斜率绝对值相同）

bool is\_valid(vector<int>& board, int row, int col) {

for (int i = 0; i < row; ++i) { // 检查同列是否有皇后

if (board[i] == col) return false;

// 检查对角线是否有皇后

if (abs(row - i) == abs(col - board[i])) return false;

}

return true;

}

// res存储结果

void n\_queens\_helper(vector<vector<string>>& res, vector<int>& board, int row, int n) {

if (row == n) { // 找到一组解

vector<string> solution;

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

string row(n, '.');

row[board[i]] = 'Q';

solution.push\_back(row);

}

// 将一组解放入res中

res.push\_back(solution);

return;

}

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

if (is\_valid(board, row, col)) { // 找到一个可行的位置

board[row] = col;

n\_queens\_helper(res, board, row + 1, n); // 继续搜索下一行

board[row] = -1; // 回溯到上一层

}

}

}

// 表示棋盘大小

vector<vector<string>> n\_queens(int n) {

vector<vector<string>> res;

vector<int> board(n, -1);

n\_queens\_helper(res, board, 0, n);

return res;

}

int main()

{

int counts = 0;

vector<vector<string>> res = n\_queens(8);

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

for(int j = 0; j < res[i].size(); j++) {

cout << res[i][j] << endl;

}

counts++;

cout << "" << endl;

}

cout << counts;

return 0;

}

# 求子集

#include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> result;

vector<int> path;

void backTracking(vector<int>& nums, int startIndex) {

result.push\_back(path); // 收集子集，要放在终止添加的上面，否则会漏掉自己

if (startIndex >= nums.size()) { // 终止条件

return;

}

for (int i = startIndex; i < nums.size(); i++) {

path.push\_back(nums[i]); // 单个结果

backTracking(nums, i + 1);

path.pop\_back(); // 回溯

}

}

vector<vector<int>> subsets(vector<int>& nums) {

result.clear();

path.clear();

backTracking(nums, 0);

return result;

}

int main()

{

int sum = 0;

int M = 9;

vector<int> nums = { 7,5,1,2,10 };

vector<int> temp = {};

vector<vector<int>> res = subsets(nums);

cout << "全部子集：" << endl;

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

cout << "{";

for(int j = 0; j < res[i].size(); j++) {

cout << res[i][j] << ",";

}

cout<< "}";

cout << endl;

}

cout << "结果：" << endl;

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

for(int j = 0; j < res[i].size(); j++) {

sum += res[i][j];

temp.push\_back(res[i][j]);

}

if(sum == 9) {

cout << "{";

for(int k = 0; k < temp.size(); k++) {

cout << temp[k] << ",";

}

cout << "}" << endl;

}

sum = 0;

temp.clear();

}

return 0;

}