ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»

Кафедра МОЭВМ

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

**на тему:**

**«Перебор с возвратом»**

**по дисциплине «Построение и анализ алгоритмов»**

Выполнила: студентка группы №1381

Лавринович Т. В.

Проверил: Казаков Б.Б.

Санкт-Петербург

2013 г.

**Цель работы**

Изучить перебор с возвратом, принцип его работы. Составить программу с использованием данного перебора.

**Задание**

Вариант 8.6.5 (перетягивание каната)

Перетягивание каната – это состязание в грубой силе, когда две группы людей тянут канат в противоположные стороны. Та команда, которая сумела утянуть в свою сторону канат, оъявляется победителем.

На корпоративном пикнике решили посостязаться в перетягивании каната. Участников пикника нужно было честно разбить на две команды. Каждый человек должен попасть в одну или другую команду. число человек в оной команде не должно превышать число человек в ругой более чем на одного, и суммарные веса людей должны быть близки, насколько это возможно.

**Входные данные**

Входные данные начинаются со строки, содержащей одно положительное число, которое означает количество тестовых блоков.

Первая строка каждого блока содержит n, число участников. Каждая из следующих n строк содержит вес одного из участников пикника, вес задается целым числом от 1 до 400.

**Выходные данные**

Для каждого тестового блока вывести строку, содержащую два числа: суммарные вес людей в одной команде и суммарный вес людей в другой.

**Пример**

Входные данные:

1

3

100

90

200

Выходные данные:

190 200

**Описание алгоритма**

Шаг 1. Выберем первого неиспользованного участника из массива весов.

Шаг 2. Записываем его вес к одной из команд. Отмечаем, что участник использован.

Шаг 3. Если все участники использованы и данное решение лучше предыдущего, фиксируем его. Если разница весов равна нулю, это и есть оптимальное решение, в таком случае, выходим из алгоритма.

Шаг 4. Если существует возможность перебора, то переходим к шагу 5.

Шаг 5. Если все участники использованы, то последнего записанного участника исключаем из данной команды. Переходим к шагу 1.

**Пример работы алгоритма**

3 человека участвуют в состязании.

Их веса: 120 60 70.

1) Вес первого участника 120 записываем в первую команду.

2) Вес второго участника 60 добавили к первой команде.

3) Вес третьего участника 70 добавили к первой команде

4) Разница между количеством участников в двух командах равна 3, т.к. это больше 1, не считаем, чему равна разница между суммарными весами участников двух команд.

5) Так как не использовали все переборы, то исключаем последнего участника с весом 70, записываем его во вторую команду. Все участники использованы, разница равна 110, фиксируем ее, записываем как наилучший вариант суммарные веса в обеих командах.

6) Исключаем участника с весом 70 из второй команды, так как варианты перебора с данным участником закончились, исключаем еще одного участника с весом 60, записываем его во вторую команду. Не все участники использованы (остался человек с весом 70), записываем его в первую команду. Разница равна 130, она является большей чем 110, не фиксируем ее.

7) Так как не использовали все переборы, то исключаем последнего участника 70 и записываем его во вторую команду. Разница равна 10, записали как наилучшую. Переборы закончились. Оптимальное решение: 120 130.

**Контрольные примеры**

Таблица 1. Контрольные примеры

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные данные | | | Результат |
| Количество тестовых блоков | Количество человек | Вес каждого человека | Суммарные веса в каждой команде |
| 1 | 3 | 40 20 20 | 40 40 |
| 1 | 3 | 23 87 90 | 110 90 |
| 1 | 4 | 20 30 80 80 | 100 110 |
| 1 | 5 | 25 57 80 12 100 | 137 137 |

**Исходная программа**

#include <iostream>

#include <math.h>

using namespace std;

bool see\_all(bool see[], int n) //Функция проверяющая все ли люди

//использовались в соревновании

{ //Если использовались все участникивернуть true, иначе false

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

{

if (see[i] != true)

return false;

}

return true;

}

int BackTracking(int &CommandA, int &CommandB, int Quant, int PartMass[], bool k, bool see[], int a, int b, int count, int &monte\_carlo)

{ //окончательный вес первой команды, второй команды,

//количество учавствующих людей, веса этих людей, параметр показывающий в //какую команду нужно добавить людей

//массив, определяющий использовался ли человек в соревновании или нет, вес одной команды, второй команды, разница между количеством людей в командах

int p = 0; //параметр определяющий какой человек используеться на

//данном шаге перебора

++monte\_carlo;

for (int i = 0; i < Quant; i++) //Пока не найдем неиспользованного

//участника

{

if (see[i] != true) //Если участник не использован, то

{ //запишем его к одной из команд(определяеться

//параметром k)

if (k == true)

{

count++;

a += PartMass[i];

}

else

{

count--;

b += PartMass[i];

}

see[i] = true; //отметим то, что i участник использовался

p = i;

break;

}

}

if (see\_all(see, Quant)) //Если использованы все участники

{

if ((abs(a-b) < abs(CommandA - CommandB)) && ((count >=-1) && (count <=1)))

{ //Проверяем на то, что разность весов в командах меньше

//существующей и то, что разница людей в команде не больше одного

CommandA = a; //Если да то фиксируем решение

CommandB = b;

if (CommandA - CommandB == 0) //Если разница весов 0,

//то решение оптимально и выходим

return 1;

}

}

else //Если не использованы все участники, то запускаем перебор заново

{

int ret = BackTracking(CommandA, CommandB, Quant, PartMass, true, see, a, b, count, monte\_carlo); //запускаем перебор рекурсивно (добавляем

// в первую команду)

see[++p] = false; //участник использовавшийся в последнем переборе

// используеться заново, а его вес не учитываться

if (ret == 1) //Если разница между весами команд 0, то выход

return 1;

ret = BackTracking(CommandA, CommandB, Quant, PartMass, false, see, a, b, count, monte\_carlo); //запускаем перебор рекурсивно

//(добавляем во вторую команду)

see[p] = false; //участник использовавшийся в последнем

//переборе используеться заново, а его вес не учитываться

if (ret == 1) //Если разница между весами команд 0, то выход

return 1;

}

return 0;

}

int Monte\_Carlo(int n) //Подсчет количества узлов для заданного колличества

//участников

{

int quant = 0;

int k = n;

while (n)

{

int add = 1;

for (int i = k; i >= n; i--)

{

add \*=i;

}

quant += add;

--n;

}

return quant;

}

int main()

{

//Ввод данных

int TestQuan = 0;

cin>>TestQuan; //Количество тестовых задач

int \*PartQuan = new int[TestQuan]; //Количество учасников в тестовой

//задаче

int \*\*PartMass = new int\*[TestQuan];//Массы участников в тестовой задаче

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

{

cin>>PartQuan[i]; //Ввод количества участников в i задаче

PartMass[i] = new int[PartQuan[i]];

for (int k = 0; k < PartQuan[i]; k++) //Ввод масс участников в

//i задаче

{

cin>>PartMass[i][k];

if ((PartMass[i][k] > 400) || (PartMass[i][k] < 1)) //Если

//вес больше 400 или меньше 1,

{ //то вес вводиться заново

cout<<"Input weight again\n";

--k;

}

}

}

//Решение задачи

int \*CommandA = new int[TestQuan]; //Вес первой стороны в каждой

//тестовой задаче

int \*CommandB = new int[TestQuan]; //Вес второй стороны в каждой

//тестовой задаче

for (int i = 0; i < TestQuan; i++) //Применим backtracking

// столько раз,

{ //сколько у нас тестоых задач

CommandA[i] = INT\_MAX; //Сделаем так что бы разность

//между командами

CommandB[i] = 0; //в начале была максимальной

int count = 0;

bool \*see = new bool[PartQuan[i]]; //Отображает использовался ли

//человек в соревновании или нет

for (int k = 0; k < PartQuan[i]; k++) //Вначале никто из людей

// не используеться в соревновании

{

see[k] = false;

}

BackTracking(CommandA[i], CommandB[i], PartQuan[i], PartMass[i], true, see, 0, 0, 0, count); //Запустим перебор

cout<<CommandA[i]<<" "<<CommandB[i]<<endl; //Выведем

// результат веса с одной стороны и со второй.

cout<<count<<endl; //Сложность выполнения. Количество исследуемых

// узлов.

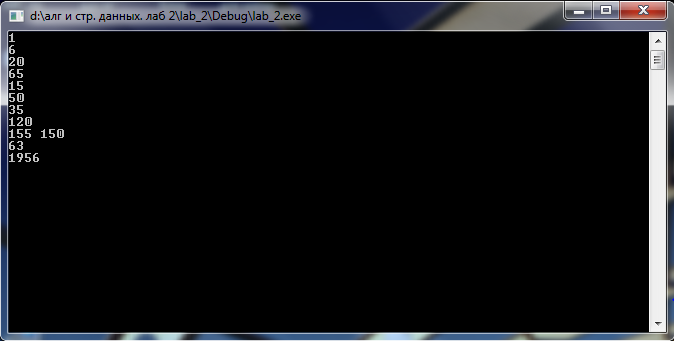
cout<<Monte\_Carlo(PartQuan[i])<<endl; //Всего узлов.

}

return 0;

}

**Результаты выполнения программы**

****

При выполнении программы полученные результаты совпадают с данными Таблицы 1. Ошибки не обнаружены.

**Вывод**

При выполнении лабораторной работы №2 «Перебор с возвратом» был составлен и реализован алгоритм перебора с возвратом.