

**Министерство образования и науки Российской Федерации**

**Федеральное агентство по образованию**

**Государственное образовательное учреждение высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Робототехника и комплексная автоматизация» (РК)**

**Кафедра «Системы автоматизированного проектирования» (РК6)**

****

**Отчет по лабораторной работе №4 по курсу**

**«Методы комбинаторных вычислений»**

**Студент: Петраков Станислав**

**Группа:** РК6-56Б

**Преподаватель:** Волосатова Т.М.

Проверил:

Дата:

2021 год

**+10Е вариант**

**Задание:**

Перечислить все разбиения заданного целого числа на слагаемые, количество которых делится на заданную величину . Для генерации разбиений следует применить алгоритм Эрлиха с мульти записью слагаемых каждого разбиения в порядке убывания своих величин слева направо и знаком + между ними.

**Описание алгоритма:**

Разбиением натурального числа n называется его представление в виде суммы целых положительных слагаемых. Однако, исходя из алгоритмических соображений, более удобно располагать слагаемые разбиений в неубывающем или в невозрастающем порядке их величин:

Суммарное количество разбиений Р(n) для любого заданного числа n можно определить по следующему рекуррентному соотношению:

В этом соотношении предполагается, что все аргументы в правой части должны быть неотрицательны и считается, что Р(0)=1;

Алгоритм Эрлиха использует мульти множественную запись его слагаемых, порождая разбиения в словарном порядке. В мульти множественных обозначениях учитывается кратность слагаемых разбиения. Это даёт возможность записать разбиение любого натурального числа n на l мульти слагаемых в следующем формате:

– кратность слагаемого .

Мульти множественная запись позволяет построить алгоритм прохождения разбиений, который переходит от текущего разбиения к следующему в словарном порядке, рассматривая только самый первый элемент разбиения без анализа предыдущих слагаемых. Способ конструирования очередного разбиения зависит от кратности последнего слагаемого текущего разбиения. При этом рассматриваются 2 случая: и .

В 1-м случае можно исключить из разбиения , чтобы иметь возможность добавить ещё одно слагаемое со значением . Во 2-м случае за счёт 2-х последних мульти слагаемых и можно добавить ещё одно слагаемое со значением . В любом то, что остаётся от уменьшаемых мультислагаемых превращается в соответствующее число единиц последнего мультислагаемого и формируется очередное разбиение. Указанный процесс должен начинаться с разбиения , где слагаемые равны 1 и завершаться разбиением из одного мультислагаемого.

**Исходный код:**

#include <iostream>

#include <vector>

using namespace std;

void PrintVector(std::vector<int>);

std::vector<int> GenerateNext(std::vector<int>, int);

std::vector<int> InsertOrdered(std::vector<int>, int, int);

std::vector<int> CompleteOnes(std::vector<int>, int);

int SumMult(std::vector<int>);

int main(int argc, char\*\* argv)

{

if (argc < 3) {

std::cout << "Enter n and m" << endl;

exit(-1);

}

int n = atoi(argv[1]);

if (n <= 0) {

std::cout << "N must be more than 0" << endl;

exit(-2);

}

int m = atoi(argv[2]);

if (m <= 0) {

std::cout << "M must be more than 0" << endl;

}

std::vector<int> start = { n, 1 };

std::vector<int> finish = { 1, n };

std::cout << "n=" << n << std::endl << "m=" << m << std::endl << std::endl;

int count = 0;

while (start != finish) {

if ((start.size() / 2 % m) == 0) {

std::cout << ++count << ": ";

PrintVector(start);

}

start = GenerateNext(start, n);

}

if (start[1] / 2 % m == 0) {

std::cout << ++count << ": ";

PrintVector(start);

}

std::cout << "Total: " << count << std::endl;

return 0;

}

void PrintVector(std::vector<int> vec) {

for (int i = 0; i < vec.size() - 2; i += 2) {

std::cout << vec[i] << "\*" << vec[i + 1] << " + ";

}

std::cout << vec[vec.size() - 2] << "\*" << vec.back();

std::cout << endl;

}

std::vector<int> GenerateNext(std::vector<int> previous, int n) {

int previousSize = previous.size();

if (previous[previousSize - 2] > 1) {

previous[previousSize - 2] -= 2;

previous = InsertOrdered(previous, 1, previous.back() + 1);

previousSize = previous.size();

if (previous[previousSize - 2] == 0) {

previous.erase(previous.end() - 2, previous.end());

}

previous = CompleteOnes(previous, n);

}

else {

previous.erase(previous.end() - 2, previous.end());

int deletedValue = previous.back();

previous.erase(previous.end() - 2, previous.end());

previous = InsertOrdered(previous, 1, deletedValue + 1);

previous = CompleteOnes(previous, n);

}

return previous;

}

std::vector<int> InsertOrdered(std::vector<int> vec, int k, int p) {

for (int i = vec.size() - 1; i > 0; i -= 2) {

if (vec[i] > p) {

vec.insert(vec.begin() + i + 1, p);

vec.insert(vec.begin() + i + 1, k);

return vec;

}

else if (vec[i] == p) {

vec[i - 1] += k;

return vec;

}

}

vec.insert(vec.begin(), p);

vec.insert(vec.begin(), k);

return vec;

}

std::vector<int> CompleteOnes(std::vector<int> vec, int n) {

int sum = SumMult(vec);

if (sum != n) {

if (vec.back() == 1) {

vec[vec.size() - 2] += n - sum;

}

else {

vec.insert(vec.end(), n - sum);

vec.insert(vec.end(), 1);

}

}

return vec;

}

int SumMult(std::vector<int> vec) {

int sum = 0;

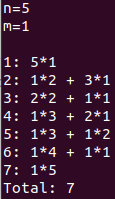
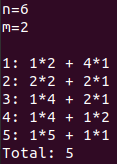
for (int i = 0; i < vec.size(); i += 2) {

sum += vec[i] \* vec[i + 1];

}

return sum;

}

**Результат работы программы:**

**Используемая литература:**

## Т.М. Волосатова, С.В. Родионов. Методы комбинаторных вычислений, учебное пособие. Издательство МГТУ им. Н.Э. Баумана. Москва. 2011