|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Алгоритмические стратегии или методы разработки алгоритмов. Перебор и методы его сокращения»** | |
|  | |
| Выполнил студент группы ИКБО-07-21 | Дамарад Д.В. |
| Принял преподаватель | Скворцова Л.А. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

**СОДЕРЖАНИЕ**

[**ЦЕЛЬ РАБОТЫ** 3](#_Toc122277364)

[**ПОСТАНОВКА ЗАДАЧИ** 4](#_Toc122277365)

[**РЕШЕНИЕ** 5](#_Toc122277366)

[**ТЕСТИРОВАНИЕ** 10](#_Toc122277367)

[**ВЫВОД** 11](#_Toc122277368)

[**ПОЛНЫЙ КОД ПРОГРАММЫ** 12](#_Toc122277369)

# **ЦЕЛЬ РАБОТЫ**

Получить навыки применения методов, позволяющих сократить число переборов в задачах, которые могут быть решены только методом перебора всех возможных вариантов решения.

# **ПОСТАНОВКА ЗАДАЧИ**

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы. Привести анализ снижения числа переборов при применении метода.
2. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

|  |  |  |
| --- | --- | --- |
| Вариант | Задача | Метод |
| 6 | Дано прямоугольное поле размером n\*m клеток. Можно совершать шаги длиной в одну клетку вправо, вниз или по диагонали вправо вниз. В каждой клетке записано некоторое натуральное число. Необходимо попасть из верхней левой клетки в правую нижнюю. Вес маршрута вычисляется как сумма чисел со всех посещенных клеток. Необходимо найти маршрут с минимальным весом. | Динамическое программирование |

# **РЕШЕНИЕ**

При подготовке задачи к решению на компьютере необходимо разработать алгоритм ее решения. Способ решения задачи может быть основан:

* на применении формул;
* на использовании рекурсий;
* на использовании алгоритма;
* на методах, которые позволяют сформировать эффективные алгоритмы:
* метод «грубой силы»;
* «разделяй и властвуй» или метод декомпозиции;
* эвристические алгоритмы;
* динамическое программирование;
* «жадные алгоритмы»;
* −поиск с возвратом;
* метод «Ветвей и границ» и др.

Если способом решения задачи является применение формулы или рекурсии, то это означает, что результаты будут получены по известным формулам. Например, вычислить линейного многочлена n-ой степени, или нахождение корней системы линейных уравнений.

Существует класс задач, например, NP - полные, для которых не существует единственного алгоритма решения и которые не могут быть решены ни одним из приведенных выше методов. Такие задачи могут иметь большое количество вариантов решения, из которых нужно выбрать одно или несколько, возможно самых оптимальных. Такие задачи относят к классу задач выбора (или классу переборных задач).

Задачи выбора обладают следующими общими свойствами:

1. Конечное число вариантов выбора (путей между городами, размещение деталей, распределение предметов между частями).
2. Каждому варианту соответствует количественная характеристика (длина пути, суммарное время обработки деталей, вес предметов в каждой части).
3. Требуется выбрать вариант, числовая характеристика которого удовлетворяет заданному условию, или определить существование такого варианта. Иногда нужен только один вариант или вариант с экстремальной характеристикой (оптимальный вариант), иногда – все варианты, а иногда не сам вариант, а только его характеристика.

На первый взгляд, кажется, что самым простым решением будет полный перебор всех вариантов и выбрать требуемый. Но такой подход может быть использован при небольшом количестве вариантов выбора. Для некоторых задач выбора разработаны точные алгоритмы, приводящие к получению результатов за приемлемое время. В задачах, в которых такие алгоритмы не применимы, или задачи достаточно сложны, применяют целенаправленные методы, сокращающие число переборов: жадные алгоритмы, методы перебора с возвратом, динамическое программирование и др. Для решения задач выбора при поиске оптимального варианта применяют эвристические алгоритмы. Эти алгоритмы обладают двумя свойствами:

* находит хорошее, возможно не оптимальное решение;
* его реализовать проще и быстрее, нежели точный алгоритм.

Метод «грубой силы» (от англ. brute force). именуется также методом решения «в лоб». Алгоритм разрабатывается на основе полного перебора всех возможных вариантов решения. Полный перебор (или метод «грубой силы», англ. brute force) — метод решения математических задач. Относится к классу методов поиска решения исчерпыванием всевозможных вариантов. 4 Этот метод представляет собой прямой подход к решению задачи, обычно основанный непосредственно на формулировке задачи и определениях используемых ею концепций Вычислительная сложность полного перебора зависит от размера задачи (размерности всех возможных решений задачи). Т.е. время выполнения будет линейно зависеть от размера задачи.

Динамическое программирование — способ решения сложных задач путём разбиения их на более простые подзадачи. Он применим к задачам с оптимальной подструктурой, выглядящим как набор перекрывающихся подзадач, сложность которых чуть меньше исходной. В этом случае время вычислений, по сравнению с «наивными» методами, можно значительно сократить.

Прямоугольник (матрицу) представим в виде двумерного динамического массива.

Функция заполнения массива fillmatrix() спрашивает пользователя каким способом заполнять массив: с помощью генератора случайных чисел или в ручную. В зависимости от выбранного ответа начинается заполнение массива значениями.

void fillmatrix(int\*\* matrix, int n, int m) {

int choice;

cout << "How would you like to fill in the matrix?" << endl << "1 - Random number generator" << endl << "2 - From the keyboard" << endl;

cout << ">> "; cin >> choice;

switch (choice) {

case 1: {

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

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

matrix[i][j] = rand() % 100;

}

}

break;

}

case 2: {

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

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

cout << "Enter " << i + 1 << j + 1 << " matrix element >> ";

cin >> matrix[i][j];

}

}

break;

}

}

}

Функция showmatrix() выводит двумерный массив на экран в виде матрицы:

void showmatrix(int\*\* matrix, int n, int m) {

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

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

cout << matrix[i][j] << ' ';

}

cout << '\n';

}

}

Идея решения задачи варианта состоит в том, чтобы использовать один и тот же заданный / входной массив для хранения решений подзадач.

Чтобы решить задачу выполним следующее:

1. Вычислим сумму весов ячеек матрицы для первой строки и столбца.
2. Запустим двумерный цикл for i [1, N] и j [1, M]. Присвоим [i][j] элементу значение равному минимуму ([i-1][j-1] элемент, [i-1][j] элемент, [i][j-1] элемент) + [i][j] элемент.
3. Возвращаем значение в последней ячейке.

Функция для поиска маршрута с минимальным весом:

int minCost(int\*\* matrix, int n, int m) {

// for 1st column

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

matrix[i][0] += matrix[i - 1][0];

}

// for 1st row

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

matrix[0][j] += matrix[0][j - 1];

}

// for rest of the matrix

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

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

matrix[i][j] += min(matrix[i - 1][j - 1], min(matrix[i - 1][j], matrix[i][j - 1]));

}

}

// returning the value in last cell

return matrix[n - 1][m - 1];

}

При решении задачи «в лоб» будет необходимо провести полный перебор всего двумерного массива и поиск маршрута с минимальной стоимостью, именно , где n и m – количество строк и столбцов соответственно. При использовании динамического программирования количество переборов будет значительно уменьшено и будет равно .

# **ТЕСТИРОВАНИЕ**

Протестируем поиск минимального маршрута на матрице размером 3x3, 3x4 и 4x3 (Рисунок 1, Рисунок 2, Рисунок 3).

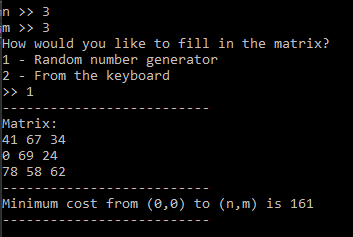


Рисунок 1 - Тестирование на матрице 3x3

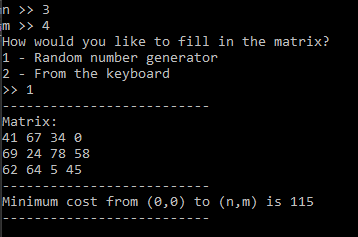


Рисунок 2 - Тестирование на матрице 3x4

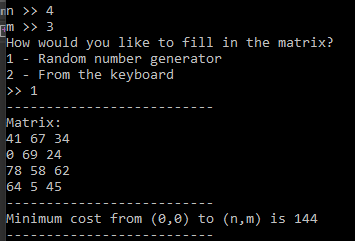


Рисунок 3 - Тестирование на матрице 4x3

Из результатов тестирования видно, что функция работает корректно.

# **ВЫВОД**

В результате выполнения практической работы были получены навыки применения методов, позволяющих сократить число переборов в задачах, которые могут быть решены только методом перебора всех возможных вариантов решения.

# **ПОЛНЫЙ КОД ПРОГРАММЫ**

#include <iostream>

using namespace std;

void fillmatrix(int\*\* matrix, int n, int m) {

int choice;

cout << "How would you like to fill in the matrix?" << endl << "1 - Random number generator" << endl << "2 - From the keyboard" << endl;

cout << ">> "; cin >> choice;

switch (choice) {

case 1: {

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

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

matrix[i][j] = rand() % 100;

}

}

break;

}

case 2: {

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

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

cout << "Enter " << i + 1 << j + 1 << " matrix element >> ";

cin >> matrix[i][j];

}

}

break;

}

}

}

void showmatrix(int\*\* matrix, int n, int m) {

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

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

cout << matrix[i][j] << ' ';

}

cout << '\n';

}

}

int minCost(int\*\* matrix, int n, int m) {

// for 1st column

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

matrix[i][0] += matrix[i - 1][0];

}

// for 1st row

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

matrix[0][j] += matrix[0][j - 1];

}

// for rest of the matrix

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

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

matrix[i][j] += min(matrix[i - 1][j - 1], min(matrix[i - 1][j], matrix[i][j - 1]));

}

}

// returning the value in last cell

return matrix[n - 1][m - 1];

}

int main() {

system("chcp 1251 > null");

int n, m;

cout << "n >> ";

cin >> n;

cout << "m >> ";

cin >> m;

int\*\* matrix = new int\* [n];

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

matrix[i] = new int[m];

}

fillmatrix(matrix, n, m);

cout << "--------------------------" << endl;

cout << "Matrix: " << endl;

showmatrix(matrix, n, m);

cout << "--------------------------" << endl;

cout << "Minimum cost from (0,0) to (n,m) is " << minCost(matrix, n, m) << endl;

cout << "--------------------------" << endl;

return 0;

}