|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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 № 8** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Алгоритмические стратегии или методы разработки алгоритмов. Перебор и методы его сокращения»** | |
|  | |
| Выполнил студент группы ИКБО-06-21 | Школьник Т.О. |
| Принял преподаватель | Филатов А.С. |

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

Москва 2022

# **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.
2. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы.
3. Привести анализ снижения числа переборов при применении метода.
4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант №28. Условие задания:

|  |  |  |
| --- | --- | --- |
| №  Варианта | Задача | Метод |
| 28 | Кузнечик прыгает вперед по столбикам, расположенным на одной линии на равных расстояниях друг от друга. Столбики имеют порядковые номера от 1 до N. Вначале Кузнечик сидит на столбике с номером 1. Он может прыгнуть вперед на расстояние от 1 до K столбиков, считая от текущего. На каждом столбике Кузнечик имеет определенный шанс встретиться с лягушкой. Определите, как нужно прыгать Кузнечику, чтобы встретить как можно меньше лягушек. | Динамическое программирование |

# **Решение**

Динамическое программирование — способ решения сложных задач путём разбиения их на более простые подзадачи. Динамическое программирование применимо не ко всем задачам, а лишь к тем, которые обладают «оптимальной структурой». Оптимальная структура означает, что задача может быть разбита на несколько аналогичных задач меньшего размера, при этом для решения конечной задачи могут быть использованы результаты решения меньших задач.

Оценим количество переборов w для каждого столбца при решении задачи стратегией «в лоб» - их будет w[n-1] + w[n-2] + … + w[n-k], где n - количество столбцов, k - максимально возможный шаг кузнечика, w[i] – количество способов попасть в данный столбец.

При использовании метода динамического программирования асимптотика решения станет равной O(n2).

Описание работы программы:

Пользователь вводит N столбцов и максимально возможный шаг – целочисленное значение K. После этого он вводит N-2 шанса для каждого из столбцов, кроме первого и последнего. Шансы были сделаны целочисленными для удобства в силу того, что условия задачи вне зависимости от типа данных для шанса встретить лягушку будут одинаковыми. Пользователь не вводит значения для 1 и последнего столбца, т.к. по условию кузнечик гарантированно начинает с первого и гарантированно должен в последнем закончить – соответственно для данных столбцов шанс встретить лягушку нулевой. Затем программа анализирует предыдущие столбцы и смотрит, из какого столбца конкретно можно эффективнее всего переместиться на данный столбец. После этого выводится минимально возможный шанс, а также путь, который ведет к этому шансу.

После ввода данных в консоль выводится суммарный минимальный шанс на встречу лягушки и путь от первого столбца к последнему.

# **Тестирование**

Проверка работоспособности программы.

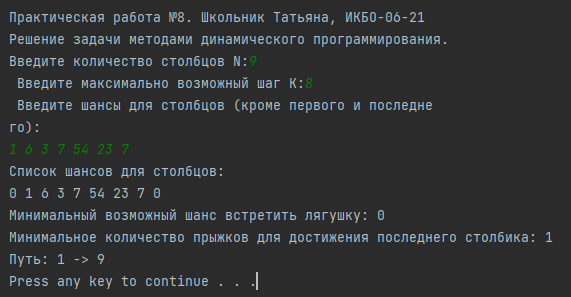


Рисунок 1. Тестирование программы

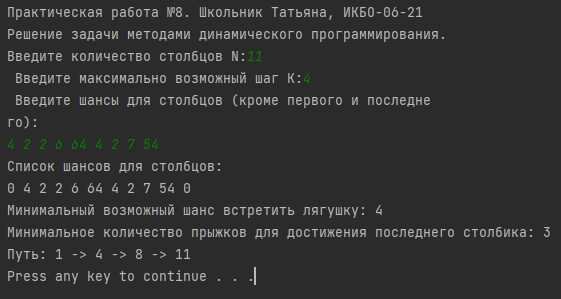


Рисунок 2. Тестирование программы

Из результатов выполнения программы видно корректную отработку программы согласно условиям задачи.

# **Вывод**

В результате выполнения работы были получены навыки сокращения времени работы программы посредством применения методов динамического программирования для тех задач, которые могут быть решены изначально только при помощи метода полного перебора.

**Исходный код программы**

main.cpp

|  |
| --- |
| //  // Created by t.shkolnik  //  #include <windows.h>  #include <iostream>  #include <vector>  using namespace std;  // Вводная функция  void init() {  cout << endl;  cout << "Практическая работа №8. Школьник Татьяна, ИКБО-06-21" << endl;  cout << "Решение задачи методами динамического программирования." << endl;  }  // Функция валидации ввода количества столбцов  int enterN() {  int n;  cout << "Введите количество столбцов N: ";  cin >> n;  if(n < 1) {  cout << "Некорректное значение!";  enterN();  }  return n;  }  // Функция валидации ввода максимального возможного шага  int enterK(int n) {  int k;  cout << "Введите максимально возможный шаг K: ";  cin >> k;  if(k < 1 || k > n) {  cout << "Некорректное значение!";  enterK(n);  }  return k;  }  // Функция ввода шансов каждого из столбцов (кроме первого и последнего)  int\* enterChances(int\* a, int n) {  cout << "Введите шансы для столбцов (кроме первого и последнего):" << endl;  for (int i = 1; i < n - 1; i++) {  cin >> a[i];  }  return a;  }  // Функция вывода шансов каждого из столбцов  void printChances(int\* a, int n) {  a[0] = 0;  a[n - 1] = 0;  cout << "Список шансов для столбцов: " << endl;  for (int i = 0; i < n; i++) {  cout << a[i] << " ";  }  cout << endl;  }  // Функция вычисления минимального шанса встретить лягушку  int calculateFrogChance(int\* a, int\* ch, int n, int k) {  int\* dp = new int[n];  dp[0] = 0;  for (int i = 1; i < n; i++) {  dp[i] = dp[i - 1];  int min\_pos = i - 1;  for (int j = 1; j <= min(i, k); j++) {  if (dp[i] > dp[i - j]) {  dp[i] = dp[i - j];  min\_pos = i - j;  }  }  ch[i] = min\_pos;  dp[i] += a[i];  }  return dp[n - 1];  }  // Функция вычисления минимального количества прыжков  int calculateMinJumpsCount(vector<int>& jumps, int\* ch, int n) {  int counter = 0;  int num = n - 1;  jumps.push\_back(num);  while (num) {  num = ch[num];  jumps.push\_back(num);  counter++;  }  return counter;  }  // Функция вывода результата программы  void printResult(int frogChance, int minJumpsCount) {  cout << "Минимальный возможный шанс встретить лягушку: " << frogChance << endl;  cout << "Минимальное количество прыжков для достижения последнего столбика: " << minJumpsCount << endl;  cout << "Путь: ";  }  // Функция вывода пути кузнечика  void printWay(vector<int>& jumps) {  for (int i = jumps.size() - 1; i >= 0; i--) {  cout << jumps[i] + 1;  if (i > 0)  cout << " -> ";  }  cout << endl;  }  int main() {  SetConsoleOutputCP(CP\_UTF8);  init();  int n, k;  n = enterN();  k = enterK(n);  int\* enteredChances = new int[n];  enteredChances = enterChances(enteredChances, n);  printChances(enteredChances, n);  int\* chances = new int[n];  int frogChance = calculateFrogChance(enteredChances, chances, n, k);  vector<int> jumps;  int minJumpsCount = calculateMinJumpsCount(jumps, chances, n);  printResult(frogChance, minJumpsCount);  printWay(jumps);  system("pause");  return 0;  } |