МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

# ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ 7_2
**по дисциплине**
**«Структуры и алгоритмы обработки данных»**

Тема. Алгоритмические стратегии или методы разработки алгоритмов. Перебор и методы его сокращения.

Выполнил студент группы ИКБО-60-23 Шеенко В.А

Принял старший преподаватель Скворцова Л.А.

Москва 2024

**СОДЕРЖАНИЕ**

# 1 ЦЕЛЬ РАБОТЫ

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

# 2 ЗАДАНИЕ №1

## 2.1 Постановка задачи

Задание. Разработка и программная реализация задач с применение метода сокращения числа переборов (табл. 1)

Таблица 1 – Задание индивидуального 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 |

## 2.2 Метод решения

### 2.2.1 Оценка сложности алгоритма для разных методов решения

При использовании стратегии грубой силы задача решается путем перебора всех возможных путей, которые Кузнечик может пройти от первого столбика до последнего.

**Количество всех путей**: для каждого столбика $\boldsymbol{i}$ Кузнечик может прыгнуть на расстояние от 1 до $\boldsymbol{K}$. На $\boldsymbol{i}$-м шаге есть максимум $\boldsymbol{K}$ вариантов прыжков. Таким образом, общее количество путей оценивается как:

$$T(N, K) = K^{(N-1)}$$

где $\boldsymbol{N}$ – количество столбиков, а $\boldsymbol{K}$ – максимальная длина прыжка.

**Динамическое программирование** – это метод оптимизации, который используется для решения задач, где сложная проблема разбивается на множество перекрывающихся подзадач. Решение каждой подзадачи хранится и переиспользуется, что значительно сокращает объем вычислений по сравнению с «грубой силой».

При применении динамического программирования каждый столбик $\boldsymbol{i}$ рассматривается только один раз и для каждого столбика выполняется не более $\boldsymbol{K}$ операций. Время выполнения алгоритма оценивается как:

$$O(N * K)$$

где N – количество столбиков, а K – максимальная длина прыжка.

**Сравнение с «грубой силой»:**

- При $\boldsymbol{N} = 10$ и $\boldsymbol{K} = 3$:
  - Стратегия грубой силы: $T(10, 3) = 19683$.
  - Динамическое программирование: 10×3=30 операций.
- При $\boldsymbol{N} = \boldsymbol{20}$ и $\boldsymbol{K} = 5$:
  - Стратегия грубой силы: $T(20, 5) = 1.9 \times 10^{13}$.
  - Динамическое программирование: $20 \times 5 = 100$.

Применение динамического программирования приводит к э**кспоненциальному снижению числа переборов** по сравнению с методом грубой силы.

### 2.2.2 Алгоритм решения задачи

Сначала инициализируются входные данные: максимальная длина прыжка K и массив рисков для каждого столбика. Затем создаем два массива:

- Первый (***dp***) для хранения минимального риска до каждого столбика.
- Второй (***prev***) для восстановления пути, указывающий, с какого столбика мы пришли на текущий.

Известно, что Кузнечик начинает с первого столбика, поэтому риск на старте равен риску первого столбика ($dp[1] = risks[0]$). Остальные значения

для риска пока устанавливаются как бесконечно большие, так как еще неизвестно, как их минимизировать.

Далее начинаем перебор столбиков. Для каждого столбика рассматриваются все возможные прыжки вперёд (от 1 до K столбиков). Если такой прыжок возможен (не выходит за пределы столбиков), вычисляем новый риск, который получится, если прыгнуть на этот столбик. Если он меньше уже записанного значения, обновляем минимальный риск и сохраняем, откуда мы пришли.

Когда мы рассчитали минимальный риск для последнего столбика, можем восстановить путь. Для этого мы идем обратно, начиная с последнего столбика, и смотрим, откуда мы туда пришли. Так шаг за шагом получаем оптимальный маршрут.

### 2.3 Программная реализация

Далее будет представлено решение данной задачи, соответствующее алгоритму выше, на языке программирования c++.

```
std::pair<int, std::vector<int>> minimizeRisk(const std::vector<int>& risks,
int K) {
    int N = (int)risks.size();

    std::vector<int> dp(N + 1, std::numeric_limits<int>::max());
    std::vector<int> prev(N + 1, -1);

    dp[1] = risks[0];

    for (int i = 1; i <= N; ++i) {
        for (int jump = 1; jump <= K; ++jump) {
            if (i + jump <= N) {
                int newRisk = dp[i] + risks[i + jump - 1];
                if (newRisk < dp[i + jump]) {
                    dp[i + jump] = newRisk;
                    prev[i + jump] = i;
                }
            }
        }
    }

    // Восстановление пути
    std::vector<int> path;
    for (int i = N; i != -1; i = prev[i]) {
        path.push_back(i);
    }
    reverse(path.begin(), path.end());
```

```
    return {dp[N], path};
}
```

### 2.4 Тестирование

Тестовый пример:

$$risks = \{5, 2, 8, 6, 3, 7, 2, 4, 6, 1\}, \mathrm{K} = 3$$

Если решать поставленную задачу «руками», это будет выглядеть следующим образом (табл. 2):

Таблица 2 – Динамическое решение (зеленым цветом отмечены новые или измененные значения, красным – заменные значения для пути кузнечика)

| prev | 1 | 1 | 1 | 1 | 2 | 345 | 45 | 5 | 67 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| risks / i | 5 | 2 | 8 | 6 | 3 | 7 | 2 | 4 | 6 | 1 |
| 0 | 5 | * | * | * | * | * | * | * | * | * |
| 1 | 5 | 7 | 13 | 11 | * | * | * | * | * | * |
| 2 | | 7 | 13 | 11 | 10 | * | * | * | * | * |
| 3 | | | 13 | 11 | 10 | 20 | * | * | * | * |
| 4 | | | | 11 | 10 | 18 | 13 | * | * | * |
| 5 | | | | | 10 | 17 | 12 | 14 | * | * |
| 6 | | | | | | 17 | 12 | 14 | 23 | * |
| 7 | | | | | | | 12 | 14 | 18 | 13 |
| 8 | | | | | | | | 14 | 18 | 13 |
| 9 | | | | | | | | | 18 | 13 |
| 10 | | | | | | | | | | 13 |

Следовательно, минимальный риск равен 13, а путь $1 \to 2 \to 5 \to 7 \to 10$.

Проверим работоспособность программной реализации алгоритма (рис. 1)

```
Z:\MIREA\SIAOD2\SIAOD-3-sem\7_2\code\cmake-build-debug\code.exe
Minimal risk: 13
Optimal path: 1 2 5 7 10

Process finished with exit code 0
```

Рисунок 1 – Тестирование алгоритма

Экспериментальные результаты совпадают с теоретическими.

### 2.5 Код всей программы

```
#include <iostream>
#include <vector>
#include <limits>
#include <algorithm>


std::pair<int, std::vector<int>> minimizeRisk(const std::vector<int>& risks,
int K) {
    int N = (int)risks.size();

    std::vector<int> dp(N + 1, std::numeric_limits<int>::max());
    std::vector<int> prev(N + 1, -1);

    dp[1] = risks[0];

    for (int i = 1; i <= N; ++i) {
        for (int jump = 1; jump <= K; ++jump) {
            if (i + jump <= N) {
                int newRisk = dp[i] + risks[i + jump - 1];
                if (newRisk < dp[i + jump]) {
                    dp[i + jump] = newRisk;
                    prev[i + jump] = i;
                }
            }
        }
    }

    // Восстановление пути
    std::vector<int> path;
    for (int i = N; i != -1; i = prev[i]) {
        path.push_back(i);
    }
    reverse(path.begin(), path.end());

    return {dp[N], path};
}

int main() {
```

```
    int k = 3;
    std::vector<int> risks = {5,2,8,6,3,7,2,4,6,1};

    auto [minRisk, path] = minimizeRisk(risks, k);

    std::cout << "Minimal risk: " << minRisk << std::endl;
    std::cout << "Optimal path: ";
    for (int pos : path) {
        std::cout << pos << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

## ВЫВОД

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

## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных : учебное пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова. — Москва : РТУ МИРЭА, 2022 — Часть 2 : Поиск в тексте. Нелинейные структуры данных. Кодирование информации. Алгоритмические стратегии — 2022. — 111 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/310826 (дата обращения: 10.09.2024).
2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01- 01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с