Изображение выглядит как текст, керамическая посуда

Автоматически созданное описание

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**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

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

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Кликушин В.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

ОГЛАВЛЕНИЕ

[1 УСЛОВИЕ ЗАДАЧИ ВАРИАНТА 3](#_Toc153143523)

[2 ОПИСАНИЕ МЕТОДА 4](#_Toc153143524)

[3 ОЦЕНКА КОЛИЧЕСТВА ПЕРЕБОРОВ 5](#_Toc153143525)

[4 АНАЛИЗ СНИЖЕНИЯ ЧИСЛА ПЕРЕБОРОВ 6](#_Toc153143526)

[5 РАЗРАБОТКА ПРОГРАММЫ 7](#_Toc153143527)

[5.1 ОПИСАНИЕ ПОДХОДА К РЕШЕНИЮ 7](#_Toc153143528)

[5.2 ФУНКЦИОНАЛ РЕШЕНИЯ 7](#_Toc153143529)

[5.3 ТЕСТОВЫЙ ПРИМЕР 8](#_Toc153143530)

[5.4 КОД ПРОГРАММЫ 9](#_Toc153143531)

[5.5 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ 17](#_Toc153143532)

[6 ВЫВОДЫ 21](#_Toc153143533)

[7 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ 22](#_Toc153143534)

# 1 УСЛОВИЕ ЗАДАЧИ ВАРИАНТА

Искатель сокровищ входит в пещеру, которая состоит из одинаковых комнат с проходами налево и направо в другие комнаты. Глубина пещеры известна. В каждой комнате есть возможность найти сокровище (+ очки) или встретить опасность (-очки). Необходимо совершить поход в пещеру до конца вглубь с наибольшим количеством очков. Применить метод ветвей и границ.

# 2 ОПИСАНИЕ МЕТОДА

Метод ветвей и границ (Branch and Bound) — это алгоритмический метод решения задач оптимизации, который используется для эффективного поиска наилучшего решения в пространстве всех возможных вариантов. Он обычно применяется для задач комбинаторной оптимизации, таких как задачи нахождения минимального пути в графе, распределения ресурсов, задачи коммивояжера и другие. Принцип метода ветвей и границ:

1. Разбиение (Branching): на каждом шаге алгоритма выбирается один из доступных вариантов решения. Пространство поиска разбивается на более мелкие подпространства (ветви).
2. Оценка (Bounding): для каждой ветви вычисляется некоторая верхняя или нижняя граница решения (bound). Эта оценка позволяет ограничить пространство поиска и отсечь ветви, которые не могут содержать оптимальное решение.
3. Поиск (Search): Алгоритм исследует дерево возможных решений, отсекая ветви с невыгодными оценками и продолжая исследование более перспективных вариантов.
4. Поиск оптимального решения: Метод ветвей и границ продолжает работу до тех пор, пока не будет найдено оптимальное решение или пока не будут исследованы все перспективные ветви.

Будем считать, что пещера стремится к структуре полного идеально-сбалансированного бинарного дерева. Зная глубину пещеры и пользуясь фактом, согласно которому в комнате может быть только +1 или -1 очко, можно вычислить, сколько максимум очков можно заработать в наилучшем случае, спускаясь от комнаты на i-ом уровне пещеры. На каждом уровне пещеры выбираются две ветви - влево и вправо, соответствующие двум доступным путям. Для каждой ветви вычисляется оценка, представляющая максимально возможное количество очков, которое может быть получено при продолжении пути с учетом оставшихся комнат. Алгоритм просматривает различные ветви, отсекая те, где оценка показывает, что потенциальное количество очков не может превзойти уже найденное лучшее решение. Метод продолжает исследование перспективных ветвей до тех пор, пока не будет найдено оптимальное решение или пока не будут исследованы все перспективные ветви.

# 3 ОЦЕНКА КОЛИЧЕСТВА ПЕРЕБОРОВ

Приравняем решение методом грубой силы и наихудший случай, возникающий в методе ветвей и границ для моей задачи.

При использовании метода ветвей и границ в худшем случае время выполнения может быть экспоненциальным, так как мы рассматриваем все возможные ветви в дереве решений.

В данной задаче глубина дерева равна высоте пещеры, и она может быть выражена как log₂(N), где N - общее количество комнат. Общее количество переборов в худшем случае будет пропорциональным .

# 4 АНАЛИЗ СНИЖЕНИЯ ЧИСЛА ПЕРЕБОРОВ

Снижение числа переборов зависит от условия задачи (задания пещеры). Анализ снижения числа переборов при использовании метода ветвей и границ включает в себя следующие ключевые моменты:

Отсечение поддеревьев: Метод ветвей и границ может отсекать поддеревья, которые не могут содержать оптимального решения. Это существенно сокращает количество проверок и переборов.

Оценка вершин: Алгоритм использует верхние и нижние границы для оценки поддеревьев. Если верхняя граница поддерева меньше или равна текущему лучшему решению, то поддерево отсекается, так как оно не может привести к лучшему результату.

# 5 РАЗРАБОТКА ПРОГРАММЫ

## 5.1 ОПИСАНИЕ ПОДХОДА К РЕШЕНИЮ

Пещеру можно реализовать на двумерном векторе, общая длина вектора – количество комнат. Каждый i-ый элемент, представленный вектором содержит три целых числа: количество очков в данной комнате, комната, в которую можно перейти, идя налево от текущей и комната, в которую можно перейти, идя направо от текущей. Количество комнат задается пользователем. С помощью рекурсивной функции реализуем перебор с использованием метода ветвей и границ. В параметрах функции, реализующей задачу, будем хранить саму структуру данных, представляющую пещеру, текущее количество очков, максимальное количество очков, текущий путь, путь при максимальном количестве очков, текущую комнату, глубину пещеры. Необходимо совершить один полный обход, далее неоптимальные обходы будут отброшены.

В коде используется переменная bestscore для хранения лучшего счета очков. Эта переменная представляет собой верхнюю границу (или текущую верхнюю оценку) для оптимального решения. В переменной maxPossibleScore вычисляется максимальное количество очков, которое можно получить на текущем уровне. Это число используется для оценки перспективности текущей ветви и дополнительного отсечения невыгодных ветвей.

## 5.2 ФУНКЦИОНАЛ РЕШЕНИЯ

Функции

1. Основная функция для обхода пещеры с наибольшим количеством очков

Предусловие. Ссылка на двумерный вектор, представляющий пещеру, номер текущей комнаты, текущее количество очков, ссылка на переменную типа int, которая отвечает за лучшее количество очков в пещере, вектор, представляющий текущий путь, передаваемый по ссылке, вектор, представляющий лучший путь, передаваемый по ссылке, глубина пещеры.

Постусловие. Нет возвращаемого значения.

**void findtreasure(vector<vector<int>>& cave, int currentroom, int currentscore, int& bestscore, vector<int>& currentpath, vector<int>& bestpath, int depth);**

1. Вспомогательная функция для обхода пещеры

Предусловие. Ссылка на двумерный вектор, представляющий пещеру.

Постусловие. Нет возвращаемого значения. Вызывает основную функцию.

**void searchtreasure(vector<vector<int>>& cave);**

## 5.3 ТЕСТОВЫЙ ПРИМЕР

В качестве теста возьмем следующую пещеру. (Зеленый кружок +1 очко, красный кружок -1 очко). Номера комнат указаны в кружках.

Изображение выглядит как круг, диаграмма

Автоматически созданное описание

Рисунок 1 – Тест №1

Изображение выглядит как диаграмма, линия, Красочность, круг

Автоматически созданное описание

Рисунок 2 – Тест №2

Изображение выглядит как линия, диаграмма, Красочность, круг

Автоматически созданное описание

Рисунок 3 – Тест №3

## 5.4 КОД ПРОГРАММЫ

#include <iostream>

#include <vector>

using namespace std;

void findtreasure(vector<vector<int>>& cave, int currentroom, int currentscore, int& bestscore, vector<int>& currentpath, vector<int>& bestpath, int depth)

{

// проверка достижения конца пещеры

if (cave[currentroom][1] == -1 && cave[currentroom][2] == -1) {

// сохраняем лучший счет и путь до него

if (currentscore >= bestscore) {

bestscore = currentscore;

bestpath = currentpath;

}

return;

}

int leftroom = cave[currentroom][1];

int rightroom = cave[currentroom][2];

// максимальное количество очков, которое можно получить на текущем уровне

int maxPossibleScore = pow(2, depth - currentpath.size());

// продолжаем поиск только если текущий счет + максимально возможный остаток очков превышает лучший счет

if (currentscore + maxPossibleScore <= bestscore) {

return;

}

// продолжаем поиск влево

if (leftroom != -1) {

currentpath.push\_back(leftroom);

findtreasure(cave, leftroom, currentscore + cave[leftroom][0], bestscore, currentpath, bestpath, depth);

currentpath.pop\_back(); // возврат к предыдущему состоянию пути

}

// продолжаем поиск вправо

if (rightroom != -1) {

currentpath.push\_back(rightroom);

findtreasure(cave, rightroom, currentscore + cave[rightroom][0], bestscore, currentpath, bestpath, depth);

currentpath.pop\_back(); // возврат к предыдущему состоянию пути

}

}

// функция для поиска сокровища в пещере

void searchtreasure(vector<vector<int>>& cave) {

int currentscore = cave[0][0]; // текущий счет очков

int bestscore = INT\_MIN; // лучший счет очков

vector<int> currentpath; // текущий путь до лучшего результата

vector<int> bestpath; // лучший путь до лучшего результата

// вычисляем глубину дерева

int depth = log2(cave.size());

// начинаем поиск с комнаты 0 (вход в пещеру)

currentpath.push\_back(0);

findtreasure(cave, 0, currentscore, bestscore, currentpath, bestpath, depth);

// выводим лучший результат и путь до него

cout << " Лучший счет очков: " << bestscore << endl;

cout << " Лучший путь: ";

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

if (i > 0) {

cout << " -> ";

}

cout << bestpath[i];

}

cout << endl;

}

int main()

{

system("chcp 1251");

int n, num, res, room1, room2,points;

vector<vector<int>> cave;

while (true)

{

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

cout << " Меню\n";

cout << " 1. Заполнить пещеру с клавиатуры\n";

cout << " 2. Заполнить пещеру с клавиатуры со случайным количеством очков в комнатах\n";

cout << " 3. Собрать максимальное количество сокровищ в пещере\n";

cout << " 4. Протестировать функцию на готовом векторе\n";

cout << " 5. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cave.clear();

cout << " Введите количесто комнат: ";

cin >> n;

if (n == 0)

cout << " Пустая пещера\n";

else if (n==1)

cave.push\_back({ 0,-1,-1 });

else

{

cout << "!!!!! Введите -1 в переходе в левую и правую комнату, если текущая комната - конец пещеры !!!!!\n";

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

{

if (i == 0)

{

cout << " Введите в какую комнату налево можно перейти из начальной комнаты: ";

cin >> room1;

while (room1 > n-1 or room1 == i)

{

cout << " Введено неверное значение для левой комнаты, попробуйте еще раз: ";

cin >> room1;

}

cout << " Введите в какую комнату направо можно перейти из начальной комнаты: ";

cin >> room2;

while (room2 > n-1 or room2 == i)

{

cout << " Введено неверное значение для правой комнаты, попробуйте еще раз: ";

cin >> room2;

}

if (room2 == room1 and room2 != -1)

{

while (room2 == room1)

{

cout << " Нельзя перейти в одну и ту же комнату, идя налево и направо, попробуйте еще раз: ";

cin >> room2;

}

}

cave.push\_back({0,room1,room2});

}

else

{

cout << " Введите в какую комнату налево можно перейти из " << i << " комнаты: ";

cin >> room1;

while (room1 > n - 1 or room1 == i)

{

cout << " Введено неверное значение для левой комнаты, попробуйте еще раз:";

cin >> room1;

}

cout << " Введите в какую комнату направо можно перейти из " << i << " комнаты: ";

cin >> room2;

while (room2 > n - 1 or room2 == i)

{

cout << " Введено неверное значение для правой комнаты, попробуйте еще раз: ";

cin >> room2;

}

if (room2 == room1 and room2 != -1)

{

while (room2 == room1)

{

cout << " Нельзя перейти в одну и ту же комнату, идя налево и направо, попробуйте еще раз: ";

cin >> room2;

}

}

cout << " Введите количество очков в комнате " << i << " (-1 или 1): ";

cin >> points;

while (points != 1 && points != -1)

{

cout << " Неверное значение для количества очков в комнате, попробуйте еще раз: ";

cin >> points;

}

cave.push\_back({ points,room1,room2 });

}

}

}

break;

}

case 2:

{

cave.clear();

cout << " Введите количесто комнат: ";

cin >> n;

if (n == 0)

cout << " Пустая пещера\n";

else if (n == 1)

cave.push\_back({ 0,-1,-1 });

else

{

cout << "!!!!! Введите -1 в переходе в левую и правую комнату, если текущая комната - конец пещеры !!!!!\n";

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

{

if (i == 0)

{

cout << " Введите в какую комнату налево можно перейти из начальной комнаты: ";

cin >> room1;

while (room1 > n - 1 or room1 == i)

{

cout << " Введено неверное значение для левой комнаты, попробуйте еще раз:";

cin >> room1;

}

cout << " Введите в какую комнату направо можно перейти из начальной комнаты: ";

cin >> room2;

while (room2 > n - 1 or room2 == i)

{

cout << " Введено неверное значение для правой комнаты, попробуйте еще раз: ";

cin >> room2;

}

if (room2 == room1 and room2 != -1)

{

while (room2 == room1)

{

cout << " Нельзя перейти в одну и ту же комнату, идя налево и направо, попробуйте еще раз: ";

cin >> room2;

}

}

cave.push\_back({ 0,room1,room2 });

}

else

{

cout << " Введите в какую комнату налево можно перейти из " << i << " комнаты: ";

cin >> room1;

while (room1 > n - 1 or room1 == i)

{

cout << " Введено неверное значение для левой комнаты, попробуйте еще раз:";

cin >> room1;

}

cout << " Введите в какую комнату направо можно перейти из " << i << " комнаты: ";

cin >> room2;

while (room2 > n - 1 or room2 == i)

{

cout << " Введено неверное значение для правой комнаты, попробуйте еще раз: ";

cin >> room2;

}

if (room2 == room1 and room2 != -1)

{

while (room2 == room1)

{

cout << " Нельзя перейти в одну и ту же комнату, идя налево и направо, попробуйте еще раз: ";

cin >> room2;

}

}

cave.push\_back({ (rand()%2)\*2-1,room1,room2});

}

}

}

break;

}

case 3:

{

if (cave.size() == 0)

cout << " Пустая пещера\n";

else

searchtreasure(cave);

break;

}

case 4:

{

cave = {

{0, 1, 2}, //0

{1, 3, 4}, //1

{1, 5, 6}, //2

{-1, 7, 8}, //3

{1, 9, 10}, //4

{-1, 11, 12}, //5

{1, 13, 14}, //6

{1, 15, 16}, //7

{1, 17, 18}, //8

{-1, 19, 20}, //9

{-1, 21, 22}, //10

{1, 23, 24}, //11

{1, 25, 26}, //12

{1, 27, 28}, //13

{1, 29, 30}, //14

{-1, -1, -1}, //15

{-1, -1, -1}, //16

{-1, -1, -1}, //17

{-1, -1, -1}, //18

{-1, -1, -1}, //19

{-1, -1, -1}, //20

{-1, 31, -1}, //21

{-1, -1, -1}, //22

{1, -1, -1}, //23

{1, -1, -1}, //24

{-1, -1, -1}, //25

{1, -1, -1}, //26

{-1, -1, -1}, //27

{1, -1, -1}, //28

{-1, -1, -1}, //29

{-1, -1, -1}, //30

{-1, -1, -1}, //31

};

searchtreasure(cave);

break;

}

case 5:

return 0;

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

## 5.5 РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 4 – Вектор, задающий тестовый пример №1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 – Результат тестирования пещеры №1

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

Рисунок 6 - Вектор, задающий тестовый пример №2

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 - Результат тестирования пещеры №2

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описание

Рисунок 8 - Вектор, задающий тестовый пример №3

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 9 - Результат тестирования пещеры №3

# 6 ВЫВОДЫ

В ходе практической работы были получены навыки работы с алгоритмами сокращения числа переборов в комбинаторных задачах.

# 7 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Учебно-методическое пособие СиАОД (часть 2)

2. Приложение к практическим работам – СДО (online-edu.mirea.ru)