Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский институт ИТМО»

Факультет МР и П

Алгоритмы и структуры данных

Лабораторная работа №5.

«Деревья. Пирамида, пирамидальная сортировка. Очередь с приоритетами»

Вариант «9»

Выполнил студент:

Розметов Джалолиддин

Группа № D3210

Преподаватель: Артамонова Валерия Евгеньевна

г. Санкт-Петербург

2024

Оглавление

Задание 1

В этой задаче ваша цель - привыкнуть к деревьям. Вам нужно будет прочитать описание дерева из входных данных, реализовать структуру данных, сохранить дерево и вычислить его высоту.

• Вам дается корневое дерево. Ваша задача - вычислить и вывести его высоту. Напомним, что высота (корневого) дерева - это максимальная глубина узла или максимальное расстояние от листа до корня. Вам дано произвольное дерево, не обязательно бинарное дерево.

• Формат ввода или входного файла (input.txt). Первая строка содержит

число узлов n ($1 \le n \le 10^5$). Вторая строка содержит n целых чисел от −1 до n−1 – указание на родительский узел. Если i-ое значение равно −1, значит, что узел i - корневой, иначе это число является обозначением индекса родительского узла этого i-го узла ($0 \le i \le n - 1$). Индексы считать с 0. Гарантируется, что дан только один корневой узел, и что входные данные предстваляют дерево.

• Формат вывода или выходного файла (output.txt). Выведите целое число высоту данного дерева.

• Ограничение по времени. 3 сек.

• Ограничение по памяти. 512 мб.

• Пример 1:

input.txt	output.txt
5 4 -1 4 1 1	3

Код

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []
```

```
def build_tree(parents):
    nodes = [TreeNode(i) for i in range(len(parents))]
    root = None
    for i, parent in enumerate(parents):
        if parent == -1:
            root = nodes[i]
        else:
            nodes[parent].children.append(nodes[i])
    return root

def tree_height(root):
    if not root:
        return 0
    if not root.children:
        return 1
    max_height = 0
    for child in root.children:
        max_height = max(max_height, tree_height(child))
    return max_height + 1

def main():
    with open('input.txt', 'r') as f:
        n = int(f.readline().strip())
        parents = list(map(int, f.readline().strip().split()))

    root = build_tree(parents)
    height = tree_height(root)

    with open('output.txt', 'w') as f:
        f.write(str(height))

if __name__ == "__main__":
    main()
```

Вводимые данные:

5
4 -1 4 1 1

Вывод кода:
3

Описание кода:

1. TreeNode: Класс, представляющий узел дерева. Каждый узел имеет значение и список детей (поддеревьев).
2. build_tree: Функция, которая строит дерево на основе списка родителей. Для каждого элемента списка создается узел дерева, и затем связи между родителями и их детьми устанавливаются путем добавления узлов в список детей.
3. tree_height: Рекурсивная функция, которая вычисляет высоту дерева. Она проходит по всем детям текущего узла и находит максимальную высоту среди них, увеличивая ее на 1 при переходе к следующему уровню.

4. main: Функция чтения входных данных из файла, построения дерева, вычисления его высоты и записи результата в файл вывода.

Описание проведенных тестов:
Код строит дерево на основе списка родителей и определяет его высоту. Он создает узлы для каждого элемента списка родителей, устанавливает связи между родителями и их детьми, а затем рекурсивно вычисляет высоту дерева.

Вывод по работе кода:
Код успешно реализует построение дерева на основе списка родителей и определение его высоты. Он эффективно обрабатывает различные сценарии, такие как пустое дерево, дерево с одним узлом и дерево с разветвлениями. Общее время выполнения и использование ресурсов зависит от размера дерева, но алгоритм остается эффективным для большинства случаев.

Задание 2

В этой задаче вы создадите программу, которая параллельно обрабатывает список заданий. Во всех операционных системах, таких как Linux, MacOS или Windows, есть специальные программы, называемые планировщиками, которые делают именно это с программами на вашем компьютере. У вас есть программа, которая распараллеливается и использует n независимых потоков для обработки заданного списка m заданий. Потоки берут задания в том порядке, в котором они указаны во входных данных. Если есть свободный поток, он немедленно берет следующее задание из списка. Если поток начал обработку задания, он не прерывается и не останавливается, пока не завершит обработку задания. Если несколько потоков одновременно пытаются взять задания из списка, поток с меньшим индексом берет задание. Для каждого задания вы точно знаете, сколько времени потребуется любому потоку, чтобы обработать это задание, и это время одинаково для всех потоков. Вам необходимо определить для каждого задания, какой поток будет его обрабатывать и когда он начнет обработку.

• Формат ввода или входного файла (input.txt). Первая строка содержит целые числа n и m ($1 \le n \le 10^5$, $1 \le m \le 10^5$). Вторая строка содержит m целых чисел ti - время в секундах, которое требуется для выполнения i-ой задания любым потоком ($0 \le ti \le 10^9$). Все эти значения даны в том порядке, в котором они подаются на выполнение. Индексы потоков начинаются с 0.

• Формат выходного файла (output.txt). Выведите в точности m строк, причем i-ая строка (начиная с 0) должна содержать два целочисленных значения: индекс потока,

который выполняет i-ое задание, и время в секундах, когда этот поток начал выполнять задание.

• Ограничение по времени. 6 сек.

• Ограничение по памяти. 512 мб.

- Пример 1:

input.txt	output.txt
2 5 1 2 3 4 5	0 0 1 0 0 1 1 2 0 4

Код

```
import heapq

class Task:
    def __init__(self, index, time):
        self.index = index
        self.time = time

    def __lt__(self, other):
        return (self.time, self.index) < (other.time, other.index)

def process_tasks(n, m, times):
    threads = [(i, 0) for i in range(n)]
    available_threads = [(0, i) for i in range(n)]
    heapq.heapify(available_threads)
    results = []

    for i in range(m):
        start_time, thread_index = heapq.heappop(available_threads)
        end_time = max(start_time, threads[thread_index][1]) + times[i]
        threads[thread_index] = (thread_index, end_time)
        heapq.heappush(available_threads, (end_time, thread_index))
        results.append((thread_index, start_time))

    return results

def main():
    with open('input.txt', 'r') as f:
        n, m = map(int, f.readline().strip().split())
        times = list(map(int, f.readline().strip().split()))

    results = process_tasks(n, m, times)

    with open('output.txt', 'w') as f:
        for result in results:
```

```
            f.write(f"{result[0]} {result[1]}\n")

if __name__ == "__main__":
    main()
```

Вводимые данные:
2 5
1 2 3 4 5
Вывод кода:
0 0
1 0
0 1
1 2
0 4

Описание кода

1. Task Class: Представляет задачу с индексом и временем выполнения.
2. process_tasks Function: Инициализирует кучу доступных потоков. Распределяет задачи, выбирая наименее занятые потоки, обновляет их время окончания и сохраняет результаты.
3. main Function: Читает входные данные из файла input.txt. Вызывает process_tasks для распределения задач. Записывает результаты в файл output.txt.

Описание проводимых тестов:
Тестирование включало проверку корректного распределения задач по потокам в различных сценариях, включая базовые случаи, один поток, максимальную загрузку, одинаковое и большое время выполнения задач. Тесты подтвердили равномерную загрузку потоков и корректность обработки задач.

Вывод по работе кода:
Код успешно распределяет задачи по потокам с минимальной задержкой, демонстрируя корректное и эффективное выполнение в различных сценариях тестирования.

Вывод

Деревья, пирамида, пирамидальная сортировка и очередь с приоритетами - ключевые компоненты в разработке алгоритмов и структурах данных. Они позволяют эффективно управлять данными, осуществлять сортировку и обработку задач в порядке их важности. Понимание их работы и применение помогают разработчикам создавать более эффективные и оптимизированные программы.