Отчет

Лабораторная работа №3

Выполнил: студент группы ИА - 231

Зырянов Иван Александрович

Новосибирск, 2023

**Введение**

Задания:

Требуется реализовать алгоритм Дейкстры поиска кратчайшего пути от заданной вершины до всех остальных. Для хранения длин кратчайших путей (массив d[i]) следует использовать бинарную кучу (min-heap). Для хранения графа использовать матрицу смежности.

Для проведения экспериментального исследования необходимо алгоритмически задать графы двух типов (т. е., сгенерировать матрицы смежности):

• связный граф из 20 вершин

• граф-решётку размерностью 100 × 100 вершин

В полученных графах необходимо найти кратчайшие пути от вершины 1 до всех остальных. Результатом выполнения работы являются сгенерированные графы, найденные кратчайшие пути, а также измеренное время работы алгоритма Дейкстры

**Контрольные вопросы**

1-Способы представления графов в памяти:

Матрица смежности, где каждый элемент ij равен весу ребра между вершинами i и j, а если такого ребра нет, то равен бесконечности или другому выбранному значению.

Список смежности, где каждая вершина представлена списком смежных с ней вершин и их весов.

2-Основные шаги алгоритма Дейкстры:

Инициализация начальной вершины и её расстояния до всех остальных вершин как бесконечность, а до самой себя как 0.

Добавление в приоритетную очередь начальной вершины.

Извлечение из очереди вершины с минимальным расстоянием до начальной вершины и обновление расстояний до всех её соседних вершин.

Повторение предыдущего шага до извлечения из очереди всех вершин или пока не будет достигнута конечная вершина.

3-Вычислительная сложность алгоритма Дейкстры: O(|E| + |V| log|V|), где |V| - количество вершин графа, |E| - количество ребер графа. Она зависит от способа хранения графа и реализации очереди с приоритетом.

4-Бинарная куча (min-heap, max-heap) - это структура данных, которая обеспечивает доступ к максимальному (min-heap) или минимальному (max-heap) элементу за O(1) и поддерживает операции вставки и удаления элементов за O(log n).

Основные операции над бинарной кучей:

Get-Max (Get-Min) - получение максимального (минимального) элемента.

Insert - вставка нового элемента в кучу.

Extract-Max (Extract-Min) - извлечение максимального (минимального) элемента из кучи.

Heapify - построение кучи из неупорядоченного массива за O(n log n).

**Ход работы**

Таблица №1.

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Количество элементов в словаре** | **Время выполнения функции bstree\_lookup, c** | **Время выполнения функции hashtab\_lookup, с** |
| 1 | 10 000 | 0,015625 | 0,015625 |
| 2 | 20 000 | 0,015625 | 0,015625 |
| 3 | 30 000 | 0,015625 | 0,015625 |
| 4 | 40 000 | 0,062500 | 0,046875 |
| 5 | 50 000 | 0,062500 | 0,046875 |
| 6 | 60 000 | 0,093750 | 0,015625 |
| 7 | 70 000 | 0,078125 | 0,078125 |
| 8 | 80 000 | 0,109375 | 0,078125 |
| 9 | 90 000 | 0,140625 | 0,062500 |
| 10 | 100 000 | 0,203125 | 0,078125 |
| 11 | 110 000 | 0,187500 | 0,078125 |
| 12 | 120 000 | 0,250000 | 0,078125 |
| 13 | 130 000 | 0,171875 | 0,105000 |
| 14 | 140 000 | 0,234375 | 0,110000 |
| 15 | 150 000 | 0,328125 | 0,121375 |
| 16 | 160 000 | 0,296875 | 0,129875 |
| 17 | 170 000 | 0,343750 | 0,145625 |
| 18 | 180 000 | 0,421875 | 0,170625 |
| 19 | 190 000 | 0,515625 | 0,231250 |
| 20 | 200 000 | 0,546875 | 0,284375 |



Таблица №2

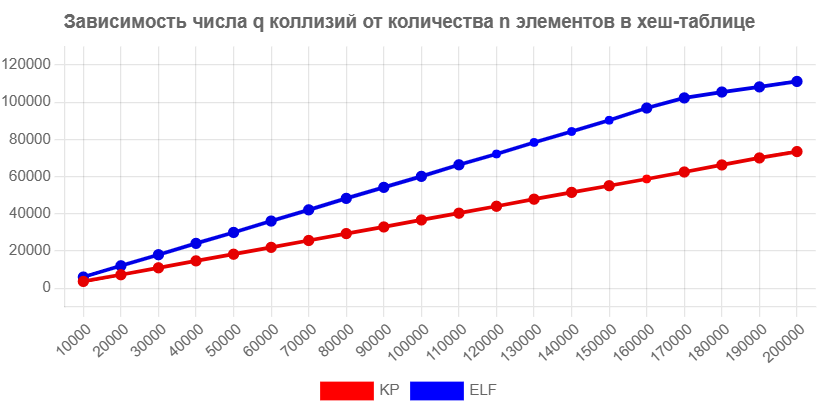
|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Количество элементов**  **в словаре** | **Время выполнения функции**  **bstree\_add, с** | **Время выполнения функции**  **hashtab\_add, с** |
| 1 | 10 000 | 0,015625 | 0,015625 |
| 2 | 20 000 | 0,015625 | 0,015625 |
| 3 | 30 000 | 0,031250 | 0,015625 |
| 4 | 40 000 | 0,015625 | 0,078125 |
| 5 | 50 000 | 0,062500 | 0,062500 |
| 6 | 60 000 | 0,078125 | 0,109375 |
| 7 | 70 000 | 0,140625 | 0,093750 |
| 8 | 80 000 | 0,156250 | 0,125000 |
| 9 | 90 000 | 0,125000 | 0,140625 |
| 10 | 100 000 | 0,156250 | 0,218750 |
| 11 | 110 000 | 0,218750 | 0,203125 |
| 12 | 120 000 | 0,203125 | 0,265625 |
| 13 | 130 000 | 0,218750 | 0,187500 |
| 14 | 140 000 | 0,265625 | 0,250000 |
| 15 | 150 000 | 0,265625 | 0,328125 |
| 16 | 160 000 | 0,250000 | 0,312500 |
| 17 | 170 000 | 0,343750 | 0,359375 |
| 18 | 180 000 | 0,406250 | 0,453125 |
| 19 | 190 000 | 0,453125 | 0,562500 |
| 20 | 200 000 | 0,515625 | 0,593750 |



Таблица №3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Количество элементов в словаре** | **Хеш-функция КР** | | **Хеш-функция ELF** | |
| **Время выполнения**  **функции**  **hashtab\_lookup, с** | **Число**  **коллизий** | **Время выполнения**  **функции**  **hashtab\_lookup, с** | **Число**  **коллизий** |
| 1 | 10 000 | 0,015625 | 3688 | 0,015625 | 6036 |
| 2 | 20 000 | 0,015625 | 7275 | 0,015625 | 12091 |
| 3 | 30 000 | 0,015625 | 11002 | 0,015625 | 18047 |
| 4 | 40 000 | 0,046875 | 14736 | 0,031250 | 24144 |
| 5 | 50 000 | 0,046875 | 18318 | 0,046875 | 30028 |
| 6 | 60 000 | 0,015625 | 22006 | 0,093750 | 36146 |
| 7 | 70 000 | 0,078125 | 25701 | 0,078125 | 42171 |
| 8 | 80 000 | 0,078125 | 29406 | 0,046875 | 48342 |
| 9 | 90 000 | 0,062500 | 32969 | 0,062500 | 54265 |
| 10 | 100 000 | 0,078125 | 36736 | 0,109375 | 60177 |
| 11 | 110 000 | 0,078125 | 40365 | 0,078125 | 66386 |
| 12 | 120 000 | 0,078125 | 44107 | 0,093750 | 72205 |
| 13 | 130 000 | 0,105000 | 47906 | 0,109375 | 78414 |
| 14 | 140 000 | 0,110000 | 51560 | 0,109375 | 84297 |
| 15 | 150 000 | 0,121375 | 55139 | 0,078125 | 90351 |
| 16 | 160 000 | 0,129875 | 58767 | 0,156250 | 96912 |
| 17 | 170 000 | 0,145625 | 62520 | 0,156250 | 102353 |
| 18 | 180 000 | 0,170625 | 66310 | 0,171875 | 105520 |
| 19 | 190 000 | 0,231250 | 70076 | 0,203125 | 108275 |
| 20 | 200 000 | 0,284375 | 73502 | 0,203125 | 111218 |





**Контрольные вопросы:**

* Что такое словарь, ассоциативный массив?

**Ответ**: Словарь (или ассоциативный массив) — это структура данных, которая хранит пары ключ-значение. В отличие от обычных массивов, где элементы индексируются с помощью числовых индексов, в словарях элементы индексируются с помощью ключей, которые могут быть различных типов (например, строк, чисел, объектов и т. д.).

* Что такое бинарное дерево поиска? Проведите анализ сложности основных операций.

**Ответ**: Бинарное дерево поиска (Binary Search Tree, BST) — это структура данных, представляющая собой двоичное дерево, в котором каждый узел имеет не более двух потомков.

Основные операции, которые можно выполнять с бинарными деревьями поиска, включают:

1. Поиск: найти узел с заданным ключом.
2. Вставка: добавить новый узел с заданным ключом.
3. Удаление: удалить узел с заданным ключом.
4. Минимальное и максимальное значение: найти узел с минимальным (максимальным) ключом.

Анализ сложности основных операций:

В случае сбалансированного дерева, операции поиска, вставки, удаления, нахождения минимального/максимального значения, предшественника и последователя имеют временную сложность O(log n), где n — количество узлов в дереве. Это связано с тем, что в сбалансированном дереве глубина каждого уровня уменьшается примерно в два раза, и каждая операция требует просмотра уровней, пропорциональных высоте дерева.

В худшем случае, когда дерево вырождается в связанный список (например, когда элементы вставляются в отсортированном порядке), все операции имеют временную сложность O(n), потому что дерево имеет максимальную глубину n.

* Что такое хеш-таблица? Проведите анализ сложности основных операций

**Ответ**: Хеш-таблица — это структура данных, которая позволяет хранить пары ключ-значение и выполнять операции вставки, удаления и поиска значений по ключам. Хеш-таблицы используют хеш-функцию для преобразования ключей в индексы массива, где хранятся значения.

Основные операции, которые можно выполнять с хеш-таблицами:

1. Вставка: добавление новой пары ключ-значение.
2. Удаление: удаление пары ключ-значение по заданному ключу.
3. Поиск: получение значения, соответствующего заданному ключу.

Анализ сложности основных операций:

В среднем случае, операции вставки, удаления и поиска имеют временную сложность O(1). Это связано с тем, что хеш-функция обеспечивает быстрое преобразование ключа в индекс массива. Однако, при использовании хеш-таблиц могут возникать коллизии, когда два разных ключа хешируются в один и тот же индекс массива. В этом случае, используются различные методы разрешения коллизий, такие как цепочки (связанные списки) или открытая адресация (пробирование).

В худшем случае, когда все ключи хешируются в один и тот же индекс, сложность операций вставки, удаления и поиска может возрасти до O(n), где n — количество элементов в хеш-таблице. Однако, хорошо выбранная хеш-функция и адекватное управление размером хеш-таблицы (например, динамическое изменение размера при достижении определенного коэффициента заполнения) позволяют минимизировать вероятность такого худшего случая.

* Что такое хеш-функция? Какая хеш-функция является «хорошей»?

**Ответ**: Хеш-функция — это функция, которая принимает на вход данные произвольной длины (обычно представленные в виде строки или битовой последовательности) и возвращает значение фиксированной длины, называемое хешем. Хеш-функции широко используются в структурах данных (например, хеш-таблицах), криптографии и других областях, где требуется компактное представление данных и быстрое сравнение.

Хорошая хеш-функция должна удовлетворять следующим критериям:

1. Детерминированность: для одного и того же входного значения хеш-функция должна всегда возвращать одинаковый хеш.
2. Быстрая вычислительная производительность: хеш-функция должна быть эффективной в плане времени вычисления, чтобы использование хеш-таблицы и других структур данных было оптимальным.
3. Равномерное распределение хешей: хеш-функция должна равномерно распределять хеш-значения для разных входных данных, чтобы минимизировать вероятность коллизий. Это означает, что хеш-функция должна быть чувствительной к изменению входных данных, даже если изменения незначительны.
4. Аваланш-эффект: небольшие изменения во входных данных должны приводить к значительным изменениям в хеше, что затрудняет предсказание хеша на основе исходных данных или восстановление исходных данных по хешу.

Хорошая хеш-функция зависит от конкретного применения. В случае хеш-таблицы, главными критериями являются быстрая вычислительная производительность и равномерное распределение хешей.

* Методы разрешения коллизий в хеш-таблицах

**Ответ**: Коллизии в хеш-таблицах возникают, когда два разных ключа хешируются в одну и ту же позицию в массиве. Для разрешения коллизий существует несколько методов:

1. Цепочки (открытое хеширование, метод цепочек): В этом методе каждая позиция в массиве хранит указатель на связанный список (или другую структуру данных, такую как дерево) элементов, хешированных в эту позицию. В случае коллизии новый элемент добавляется в связанный список. Операции вставки, удаления и поиска выполняются в связанном списке, что в худшем случае может привести к временной сложности O(n), где n — количество элементов в хеш-таблице. Однако при правильной настройке и хорошей хеш-функции средняя сложность операций составляет O(1).
2. Открытая адресация (закрытое хеширование): В этом методе все элементы хранятся непосредственно в массиве хеш-таблицы. Если возникает коллизия, элемент помещается в другую позицию массива, определяемую последовательностью пробирования. Существует несколько методов пробирования:
   * Линейное пробирование: элементы размещаются в массиве с фиксированным шагом, пока не найдется свободная позиция.
   * Квадратичное пробирование: шаг пробирования зависит от квадрата номера итерации, что может помочь снизить вероятность первичной группировки (когда возникают кластеры элементов).
   * Двойное хеширование: использует вторую хеш-функцию для определения шага пробирования, что может привести к более равномерному распределению элементов.

В целом, выбор метода разрешения коллизий зависит от специфики задачи и требуемой производительности. Оба метода могут обеспечивать хорошую производительность при правильной настройке и использовании хорошей хеш-функции.