Типы и структуры данных — Сиденко Анастасия
Лабораторная работа 6
Вариант 0 — ИУ7-33Б
Обработка очередей

Техническое задание:

Задача:

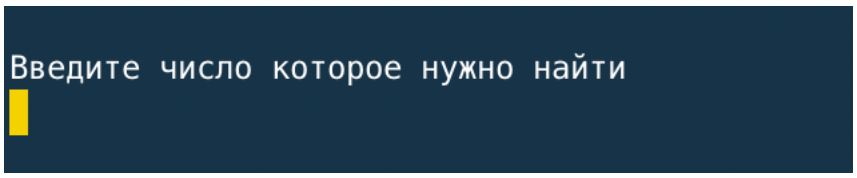
В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного целого числа в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного, то произвести реструктуризацию таблицы, выбрав другую функцию.

Входные данные: файл с числами.

Выходные данные: двоичное дерево поиска, сбалансированное дерево, хеш-таблица.

Взаимодействие с программой

Взаимодействие через консольное меню.



Аварийные ситуации:

1. Некорректные данные. Вывод сообщения.

Используемые структуры:

Структура дерева

```
struct tree {
    int data;
    struct tree *left;
    struct tree *right;
};
```

Структура сбалансированное дерево

```
struct avl_tree {
    int data;
    int height;
    struct avl_tree *left;
    struct avl_tree *right;
};
```

Структура хеш-таблица

```
struct table {
    int data;
    struct table *next;
};
```

Алгоритмы:

1. Для двоичного дерева поиска

- Печать
  void print_t_t(struct tree *tree)

- Добавление
  struct tree *tree_add(struct tree *root, struct tree *r, int data)
- Поиск
  struct tree *tree_find(struct tree *root, int data, int *finds)

2. Для сбалансированного дерева

- Печать
  void print_t_a(struct avl_tree *tree)
- Создание нового узла
  struct avl_tree *node_new(const int data)
- Разница между высотами правого и левого поддерева
  int tree_dif(struct avl_tree *tree)
- Высота дерева
  void tree_height(struct avl_tree *tree)
- Правый поворот
  struct avl_tree *rotate_right(struct avl_tree *tree)
- Левый поворот
  struct avl_tree *rotate_left(struct avl_tree *tree)
- Балансировка дереве пока разница между высотами не равна {-1,0,1}
  struct avl_tree *tree_balance(struct avl_tree *tree)
- Добавление
  struct avl_tree *tree_insert(struct avl_tree *tree, const int data)
- Чтение из файла
  struct avl_tree *tree_from_file()
- Поиск
  struct avl_tree *avl_tree_find(struct avl_tree *root, int data, int *finds)

3. Для списка

- Печать
  void list_print(struct table *head)
- Добавление
  struct table *list_push(struct table *st, int data)

4. Хеш-таблица

- Добавление
  void hash_table_push(struct table *table[], int data, int kol)

- Печать
  void hash_table_print(struct table *table[], int kol)

- Поиск
  struct table *hash_table_find(struct table *table[], int kol, int data, int *finds)

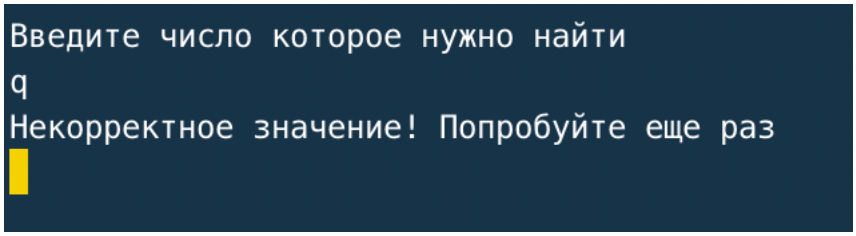
5. Файл

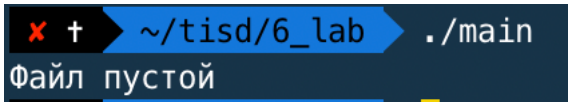
- Поиск
  int file_find(FILE *f, int data, int *finds)

Тесты:

1. Некорректные данные

Ввод некорректных данных





2. Правильные тесты

```
Дерево
(1   )
  .------\
      (2   )
        .------\
            (3   )
              .------\
                  (4   )
                    .------\
                        (5   )
                          .------\
                              (6   )

Сбалансированное дерево
                    (4   )
          /----------.------\
      (2   )                (5   )
   /----.------\              .------\
(1   )       (3   )               (6   )

Хеш-таблица
[0] = NULL
[1] = 1
[2] = 2
[3] = 3
[4] = 4
[5] = 5
[6] = 6
```

```
Введите число которое нужно найти
1

Двоичное дерево поиска
Время 4
Количество сравнений 1
Память 144

Сбалансированное дерево
Время 2
Количество сравнений 3
Память 192

Хеш таблица
Время 2
Количество сравнений 1
Память 96

Файл
Время 20
Количество сравнений 1
Память 4
```

Сравнение времени работы.

```
Среднее время поиска
Для двоичного дерева поиска 8.501000
Для сбалансированного дерева 1.353000
Для хеш-таблицы 1.265000
Для файла 306.000000
```

Вопросы:

1. Что такое дерево?
   Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

2. Как выделяется память под представление деревьев?
   В памяти деревья можно представить в виде связей с предками или связного списка потомков.

3. Какие стандартные операции возможны над деревьями?
   Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

4. Что такое дерево двоичного поиска?
   Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?
   Если при добавлении узлов в дерево мы будем их равномерно располагать слева и справа, то получится дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу. Такое дерево называется идеально сбалансированным.

6. Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?
   Пробегаемся от корня дерева, переходя на левую или правую ветвь пока не найдем нужное нам число. В сбалансированном дереве поиск короче за счет равномерное распределения вершин по ветвям.

7. Что такое хеш-таблица, каков принцип ее построения?
   Массив, заполненный в порядке, определенным хеш-функцией, называется хеш- таблицей.
   Строится по хеш-функции.

8. Что такое коллизии? Каковы методы их устранения.
   Может возникнуть ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда $h(K1)=h(K2)$, в то время как $K1 \neq K2$.
   Такая ситуация называется коллизией.
   Первый метод – внешнее (открытое) хеширование (метод цепочек). В случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список.
   Другой путь решения проблемы, связанной с коллизиями – внутреннее

(закрытое) хеширование (открытая адресация). Оно, состоит в том, чтобы полностью отказаться от ссылок.

9. В каком случае поиск в хеш-таблицах становится неэффективен?
   Если для поиска элемента необходимо более 3−4 сравнений, то эффективность использования такой хеш-таблицы пропадает и ее следует реструктуризировать (т.е. найти другую хеш-функцию), чтобы минимизировать количество сравнений для поиска элемента

10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш- таблицах
    Использование деревьев для поиска информации – $O(\log_2 n)$.
    Поиска в хеш-таблице равна O(1)

Вывод:

Использование хеш-таблиц эффективно и по времени, и по памяти. Сбалансированное дерево гораздо эффективнее по времени в отличие от дерева двоичного поиска, но памяти занимает больше. Поиск по файлу неэффективен совсем, так как проходимся по всем строкам пока не найдем нужную.