Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Е. А. Суханов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №2

Задача: Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до $2^{64}-1$. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).

! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Вариант структуры данных: Красно-чёрное дерево.

1 Описание

Задачу можно разделить на две части:

- Реализация красно-чёрного дерева;
- Интерактивный ввод-вывод.

Красно-чёрное дерево является сбалансированным деревом поиска. Идея этой структуры данных заключается в покраске всех вершин в два цвета: красный и чёрный. При этом дерево до и после выполнения операций над ним должно сохранять следующие свойства [1]:

1. Каждая вершина имеет красный или чёрный цвет;
2. Корень всегда чёрный;
3. Если узел красный, то оба его дочерних узла чёрные;
4. Для каждого узла все простые пути от него до листьев имеют одинакове количество черных вершин;
5. Каждый лист дерева является черным.

Поиск происходит обычным образом для бинарных деревьев поиска.

Вставка в красно-чёрное дерево происходит в два этапа:

1. Поиск места для вершины. Вставка вершины. Покраска её в красный цвет;
2. Восстановление нарушенных свойств дерева. Очевидно, если родитель вставленной вершины является черным, то никакие свойства не нарушаются. Если родитель красный, то нам нужно рассмотреть два случая:
   - Если дядя красный, нужно покрасить дядю и отца в черный цвет, а дедушку в красный. Если дети дедушки красные, то дед был черным. Следовательно баланс не изменился, но теперь нам нужно восстновить свойства дерева относительно дедушки, так как его родитель мог быть красным;
   - Если дядя черный. Во-первых нужно проверить, что бы вставляемая вершина была левым ребенком, если дядя является правым и наоборот. В противном случае нужно совершить соответствующий поворот. Во-вторых отца надо покрасить в черный, а дедушку в красный. И совершить соответствующий поворот. Таким образом черная высота не изменяется, а свойства не нарушены.

Удаление вершины из красно-чёрного дерево является немного более сложной задачей:

1. Удаление вершины. Если у вершины не было детей, то просто удаляем ее. Если у вершины был один ребенок, скрепляем ребенка с родителем. Если у вершины два ребенка, нужно заменить ключ у удаляемой вершины на ключ следующего или предыдущего узела. Затем производить удаление относительно следующего или предыдущего узла.

2. Если удаленная вершина была черной, нужно восстановить кол-во черных вершин:

   - Если брат ребенка удаленной вершины красный, сводим ко второму случаю. Нам нужно совершить поворот ребра между отцом и братом. Брата красим в черный, а отца в красный. Тогда черная высота не нарушается, но у текущей вершины брат становится черным;
   - Если брат черный и оба его ребёнка тоже черные. Красим брата в красный, а отца в черный. Таким образом мы не изменяем кол-во черных элементов, но восстанавливаем высоту отца через ребенка. Однако теперь нужно рассмотреть отца, так как изначально он мог быть черным;
   - Если брат является правым ребенком и он чёрный, левый сын брата красный, а правый – черный: Красим брата в красный, а красного сына в черный. Затем выполняем правый поворот относительно брата. Таким образом сводим задачу к следующему случаю. Если брат левый ребёнок, то меняем лево и право местами.
   - Если брат ялвяется правым ребенком, а правый сын брата красный. Красим правого сына брата в черный, брата в цвет отца, отца в черный. Совершаем левый поворот относительно отца. Заканчиваем работу. Если брат левый ребёнок, то меняем лево и право местами.

В [1] доказывается лемма, согласно которой высота данного дерева меньше или равна $2\lg(n+1)$, где n – количество узлов в дереве. Таким образом сложность вставки, удаления и поиска равна $O(\lg n)$.

2 Исходный код

Для хранение слов я использую следующую структуру:

```
struct TData {
    static const int MAX_STRING_LEN = 257;
    char buf[MAX_STRING_LEN];

    TData() {
        for(int i = 0; i < MAX_STRING_LEN; ++i)
            buf[i] = 0;
    }

    TData(const TData& data) {
        for(int i = 0; i < MAX_STRING_LEN; ++i)
            buf[i] = data.buf[i];
    }

    bool operator== (const TData& data) const {
        return operator==(data.buf);
    }

    bool operator== (const char* str) const {
        for(int i = 0; i < MAX_STRING_LEN; ++i) {
            char a = std::tolower(buf[i]);
            char b = std::tolower(str[i]);

            if(a == '\0' && b == '\0')
                return true;
            if(a != b)
                return false;
        }
        return true;
    }

    bool operator!= (const TData& data) const {
        return !(*this == data);
    }

    bool operator< (const TData& data) const {
        for(int i = 0; i < MAX_STRING_LEN; ++i) {
            char a = std::tolower(buf[i]);
            char b = std::tolower(data.buf[i]);

            if(b == '\0')
                return false;
            if(a == '\0')
                return true;
            if(a < b)
                return true;
```

```
47              else if (a > b)
48                  return false;
49          }
50          return false;
51      }
52
53      friend std::istream& operator>> (std::istream& in, TData& data) {
54          in >> std::setw(MAX_STRING_LEN) >> data.buf;
55          return in;
56      }
57
58      friend std::ostream& operator<< (std::ostream& out, const TData& data) {
59          out << data.buf;
60          return out;
61      }
62  };
```

Для простоты работы я перегрузил операторы ввода, вывода, а так же сравнения. Такой способ хранения строки может быть не самым эффективным по памяти, зато проще в написании.

Обработка команд происходит в функции main.

```
1   int main() {
2       std::ios_base::sync_with_stdio(false);
3
4       TRedBlackTree<TData, uint64_t> tr;
5       char cmd;
6       char path[PATH_MAX];
7       TData key;
8       TData mod;
9       while(std::cin >> cmd) {
10          switch (cmd)
11          {
12          case '+':
13              uint64_t value;
14              std::cin >> key >> value;
15              if(tr.Insert(key, value))
16                  std::cout << "OK\n";
17              else
18                  std::cout << "Exist\n";
19              break;
20          case '-':
21              std::cin >> key;
22              if (tr.Remove(key))
23                  std::cout << "OK\n";
24              else
25                  std::cout << "NoSuchWord\n";
26              break;
27          case '!':
```

```
28              std::cin >> mod >> path;
29              try {
30                  if (mod == "Save")
31                      TRedBlackTree<TData, uint64_t>::Save(path, tr);
32                  else if (mod == "Load")
33                      tr = std::move(TRedBlackTree<TData, uint64_t>::Load(path));
34
35                  std::cout << "OK\n";
36              }
37              catch (const std::runtime_error& e) {
38                  std::cout << "ERROR: " << e.what() << '\n';
39              }
40              break;
41
42          default:
43              std::cin.putback(cmd);
44              std::cin >> key;
45              if(uint64_t* element = tr.Search(key))
46                  std::cout << "OK: " << *element << '\n';
47              else
48                  std::cout << "NoSuchWord\n";
49              break;
50          }
51      }
52
53      return 0;
54  }
```

Каждая команда определяется первым считанным знаком строки. Я думаю, что для чистоты, логику команд можно было вынести в отдельные функции, но, с другой стороны, команд еще не очень много.

Само красно-черное дерево выполнено в виде шаблонного класса:

```
1   template <typename Key, typename Value>
2   class TRedBlackTree {
3   public:
4       TRedBlackTree();
5       ~TRedBlackTree();
6       TRedBlackTree(const TRedBlackTree&) = default;
7
8       TRedBlackTree& operator=(TRedBlackTree&&);
9
10      bool Insert(const Key& key, const Value& value);
11      bool Remove(const Key& key);
12      Value* Search(const Key& key);
13
14      static TRedBlackTree<Key, Value> Load(const char* path);
15      static void Save(const char* path, const TRedBlackTree<Key, Value>& tree);
16
```

```
17      void PrintTree();
18
19  private:
20      struct TNode;
21
22      void PrintTree(const TNode* node);
23
24      TNode** SearchTNode(const Key& key, TNode** parent);
25      void InsertFixup(TNode* node);
26      void RemoveFixup(TNode* node, TNode* father);
27      void LeftRotate(TNode* parent);
28      void RightRotate(TNode* parent);
29
30  private:
31      TNode *root;
32  };
33
34  template <typename Key, typename Value>
35  struct TRedBlackTree<Key, Value>::TNode {
36      TNode* parent;
37      TNode* left;
38      TNode* right;
39      bool isBlack;
40
41      Key key;
42      Value value;
43
44      TNode() = default;
45      TNode(const Key& key, const Value& value);
46      ~TNode();
47  };
```

rbtree.hpp	
bool Insert(const Key& key, const Value& value)	Создает новый узел с указаным ключом и значением и вставляет его в дерево. Возвращает true, если вставка произошла успешно. false – инчае.
bool Remove(const Key& key)	Удаляет вершину с указанным ключом из дерева. Если такой вершины не существует возвращает false. Если удаление завершилось успешно – true.
Value* Search(const Key& key)	Поиск значения по ключу. Возвращает указатель на найденное значение. Если значение не было найдено, возвращает nullptr.

static TRedBlackTree<Key, Value> Load(const char* path)	Загрузка дерева из файла path. Возвращает загруженное дерево. Если загрузка не удалась – вызывается исключение.
static void Save(const char* path, const TRedBlackTree<Key, Value>& tree)	Сохранение дерева tree в файл path. Если сохранение не удалось – вызывается исключение.
void PrintTree()	Функция для отладки. Выводит дерево в консоль.

3 Консоль

```
reterer@retcom:~/Desktop/da/lab2$ make -C solution/
make: Entering directory '/home/reterer/Desktop/da/lab2/solution'
g++ -c -Wall -pedantic -std=c++14 -O2 main.cpp -o main.o
g++ main.o -o solution -O2
make: Leaving directory '/home/reterer/Desktop/da/lab2/solution'
reterer@retcom:~/Desktop/da/lab2$ cat 01.t
+ a 2
+ A 1
+ b 3
+ c 4
! Save testSave
! Load testSave
a
A
b
c
reterer@retcom:~/Desktop/da/lab2$ solution/solution <01.t
OK
Exist
OK
OK
OK
OK
OK: 2
OK: 2
OK: 3
OK: 4
```

4 Тест производительности

Я буду сравнивать скорость работы моего КЧ дерева с контейнером map из стандартной библиотеки языка c++.

```
$ make clean && make banchmark
rm -f *.o solution banchmark
g++ -Wall -pedantic -std=c++14 -O0 banchmark.cpp -o banchmark
$ ./banchmark ../generated_tests/randomtest1000000.txt 1>/dev/null
My Tree : 4167033us
STL map : 6487974us
$ ./banchmark ../generated_tests/randomtest1000000.txt 1>/dev/null
My Tree : 4196927us
STL map : 6664112us
$ ./banchmark ../generated_tests/randomtest1000000.txt 1>/dev/null
My Tree : 4372497us
STL map : 6670335us
$ make clean && make banchmark
rm -f *.o solution banchmark
g++ -Wall -pedantic -std=c++14 -O2 banchmark.cpp -o banchmark
$ ./banchmark ../generated_tests/randomtest1000000.txt 1>/dev/null
My Tree : 2776364us
STL map : 2754123us
$ ./banchmark ../generated_tests/randomtest1000000.txt 1>/dev/null
My Tree : 2765016us
STL map : 2605321us
$ ./banchmark ../generated_tests/randomtest1000000.txt 1>/dev/null
My Tree : 2732643us
STL map : 2605839us
```

Видно, что без оптимизации моя реализация быстрее на 35%, однако с оптимизацией скорости выполнения примерно равны: моя реализация медленнее на 6%.

5 Выводы

Выполняя данную лабораторную работу я глубже познакомился со сбалансированными деревьями поиска. В первую очередь с красно-чёрным деревом. Которое часто используется в стандартной библиотеке c++ для отображений и множеств. Сбалансированные деревья позволяют улучшить асимптотику операций, по сравнению с обычным деревом поиска. Что дает $O(\log n)$ даже в худшем случае.

Кроме этого я попытался использовать стороннюю библиотеку для тестирования моей реализации дерева. Тесты помогли мне выробатать интерфейс работы с моим классом красно-чёрного дерева, а так же отслеживать и своевременно исправлять ошибки.

Возможно, мне не стоило использовать шаблоны для красно-чёрного дерева. Так как это усложняет разработку. А в рамках данной задачи мне не требовалось использовать дерево для разных типов.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Красно-чёрное дерево— Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Красно-черное_дерево` (дата обращения: 3.12.2020).