Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2-3 по курсу «Дискретный анализ»

Студент: А. О. Ларченко
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №4

Задача: Необходимо создать программную библиотеку, реализующюс труктуру данных , на основе которой разработать программу-словарь.

Вариант структуры данных: PATRICIA.

Вариант ключа: `регистронезависимую последовательность букв английского алфавита длиной не более 256 символов`.

Вариант значения: `Числа в диапазоне 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, отсутствие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Также необходимо провести профилирование программы с помощью утилиты **gprof** и провести поиск утечек памяти с помощью утилиты **valgrind**.

1 Описание

Требуется написать реализацию структуры данных PATRICIA (Practical Algorithm To Retrieve Information Coded In Alphanumeric), в которой смешаны структуры данных BST и Trie. Соответсвеноо, также как и указанные структуры используется для хранения и эффективного поиска слов из набора строк или словаря.

Основная идея заключается в том, чтобы проходить дерево как в бинарном дереве поиска, но сравнивать ключи не полностью, а только соответствующие биты, это позволит сократить время время поиска образцов, уже имеющихся в структуре и избежать лишних сравнений при заранее известном несовпадении.
Каждый узел имеет, помимо исходных данных, индекс бита и указатели на левый и правый узлы. Указатели могут быть прямые (указывающие на узел ниже) и обратные (указывающие на собственный узел или узел выше).
Поиск осуществляется следующим образом. Когда идёт сравнение с очередным узлом, из исходного ключа берётся бит, чей номер соответствует индексу бита в данном узле. Если этот бит равен 0, то осуществляется переход по левому указателю, иначе – по правому. Поиск заканчивается тогда, когда происходит переход по обратному ука- зателю. Затем ключ последнего узла сравнивается с исходным ключом, после чего делается вывод о результате поиска.
Перед вставкой ключа осуществляется поиск. Если ключ не найден, то происходит непосредственно вставка нового узла. Берётся найденный при поиске узел, затем побитого сравниваются ключ в этом узле и исходный ключ. Индекс бита нового узла будет номером первого несовпадающего бита. Затем при помощи поиска по дереву ищется такое место для вставки нового узла, что после вставки сохранится следующий принцип: индексы узлов от корня до листов должны возрастать.

2 Исходный код

PATRICIA является сложной структурой данных, состоящей из узлов, в каждом из которых хранится ключ + значение, бит, необходимый для проверки, а также 2 указателя на следующие узлы. Поэтому создадим структуру узла $TNode$, а также саму нашу структуру данных и функции для работы с ней, которые содержатся в классе $TTrie$:

```
namespace Patricia_trie{

    int make_idx(int letter_pos, int bit_pos);
    TMy_pair<int, int> get_idx(int idx);

    struct TNode{
        TNode(TStr &new_str_key, uint64_t new_val){
            str_key = new_str_key;
            val = new_val;
        }

        TStr str_key;
        uint64_t val;
        TNode* refs[2];
        int idx; // letter_position * BIT_COUNT + bit_position

    };


    class TTrie{
        public:
            TTrie();
            TNode* get_root();
            TMy_pair<bool, uint64_t> search(TStr needed_key);
            bool insert(TStr insert_key, uint64_t insrt_val);
            bool erase(TStr erase_key);
            void save_trie(string & path);
            void load_trie(string & path);
            void print_trie();
            int size();
            ~TTrie();

        private:
            TNode* root;
            int elem_count;
            void destroy_nods(TNode* start_node);
            void destroy_tree();
            bool add(TNode* new_node);
            TNode* get_node(TStr find_key, TNode* start_node );
            TMy_pair <TNode*, TNode*> find_node(TNode *needed_node);
            TMy_pair<TNode*, TNode*> find_pair_for_insert(TNode *needed_node);
```

```
42            void print_nodes(TNode* node, int prev_idx);
43            TMy_pair<TNode*, TMy_pair<TNode*, TNode*>> find_PRC(TStr &needed_str);
44            void save_nodes(TNode* cur_node, ofstream &file_to);
45            void write_node_to_file(TNode* cur_node, ofstream &file_to);
46            void load_nodes(TNode* cur_node, ifstream &stream);
47            void read_node_from_file(ifstream &stream);
48
49            void del_node(TNode* node);
50            void check_root();
51            void root_create();
52
53    };
54
55 };
```

Функции класса можно разделить на 2 категории:
- пользовательские функции, которые отвечают только за вызов других, скрытых от пользователя функций:

Функция	Описание
TMy_pair<bool, uint64_t> search(TStr needed_key)	Отвечает за поиск необходимого узла
bool insert(TStr insert_key, uint64_t insrt_val	Добавлет в элемент insert_key со значением insrt_val в структуру, при отсутствии такого узла в структуре
void save_trie(string & path)	Схраняет текущую структуру в файл path
void load_trie(string & path);	Строит структуру из данных файла path

- основные функции:

Функция	Описание
void destroy_tree()	Фцнкция, которая запускает реверсивное удаление дерева с помощью функции void destroy_nods(TNode* start_node)
bool add(TNode* new_node)	Добавляет новый узел в структуру. Путем вызова функции $find\_node$ проверяет наличие узла в дереве.
TMy_pair<TNode*, TNode*> find_pair_for_insert(TNode *needed_node)	Используется в функции *add*, возвращает 2 узла, между которыми нужно создать новый узел
TMy_pair<TNode*, TMy_pair<TNode*, TNode*» find_PRC(TStr &needed_str)	Используется при удалении узлла, возвращает указатели на 3 узла: узел, который нужно удалить, его предка и его прапредка
void save_nodes(TNode* cur_node, ofstream &file_to)	Вызывает рекурсивное созранение узлов в файл, путем выхова функции $write\_node\_to\_file$

Необычными могут показаться 3 очень похожих, но в тоже время разных функций (get_node, $find_node$, $find_PRC$). Все они возвращают указатели на узлы. Наличие всех этих функций обусловленно листо-подобному представлению, которое чтобы сэкономить место не хранит обратную ссылку, а для вставки и удаления нам нужно знать родительские ссылки.

Для удобного парсинга строк и дальнейшей работы с ними нешей структуры была реализована структура $TStr$:

```
1
2  struct TStr{
3      friend istream& operator >>(istream& is, TStr &el);
4      friend ostream& operator <<(ostream& os, TStr &el);
5
6      friend ifstream& operator >>(ifstream& is, TStr &el);
7      friend ofstream& operator <<(ofstream& os, TStr &el);
8
9      TStr(){
10         val_size=0;
11         first_idx = STRING_LENGTH-1;
12         memset(val, '0', sizeof(char)* STRING_LENGTH);
13     };
14
15     TStr(string str){
16         memset(val, '0', sizeof(char)* STRING_LENGTH);
17         if (str.size() > STRING_LENGTH){
18             val_size= STRING_LENGTH;
19         } else{
```

```
20              val_size = str.size();
21          }
22          str_to_char(str, val, STRING_LENGTH);
23          first_idx = STRING_LENGTH - val_size;
24      };
25
26      TStr(const TStr &other){
27          memset(val, '0', sizeof(char)* STRING_LENGTH);
28          val_size = other.val_size;
29          first_idx = other.first_idx;
30          for (int i=first_idx; i< STRING_LENGTH; ++i){
31              val[i] = other.val[i];
32          }
33      }
34
35      TStr(TStr &&other) noexcept{
36          // val = other.val;
37          memcpy(val, other.val, STRING_LENGTH);
38          val_size = other.val_size;
39          first_idx = other.first_idx;
40          // for (int i=first_idx; i< STRING_LENGTH; ++i){
41          // val[i] = other.val[i];
42          // }
43      }
44
45
46      TStr& operator=(const TStr& other){
47          memset(val, '0', sizeof(char)* STRING_LENGTH);
48          val_size = other.val_size;
49          first_idx = other.first_idx;
50          for (int i=first_idx; i< STRING_LENGTH; ++i){
51              val[i] = other.val[i];
52          }
53          return *this;
54      }
55
56      operator string() const {
57          return string(&val[first_idx], val_size );
58      }
59
60      char operator[](int needed_idx){
61          return this->val[needed_idx];
62      }
63
64      int operator ==(const TStr& other) const{
65          if (this->val_size == 0 and this->val_size == other.val_size){
66              return -1;
67          }else if(first_idx != other.first_idx){
68              return min(first_idx, other.first_idx);
```

```
69          }
70
71          for(int i = first_idx; i< STRING_LENGTH ; ++i){
72              if (this->val[i]!= other.val[i]){
73                  return i;
74              }
75              // cout<<"Cur val: "<<this->val[i]<<" El val = "<<el.val[i]<<"\n";
76          }
77          return -1;
78      }
79
80      int size(){
81          return val_size;
82      }
83
84      int get_first_idx(){
85          return first_idx;
86      }
87
88      void print_bit_str(){
89          for(int i =0; i<STRING_LENGTH;++i){
90              print_bit_mask(val[i]-'a'+1, BIT_COUNT);
91              cout<<".";
92          }
93      }
94
95      private:
96          char val[STRING_LENGTH];
97          int val_size;
98          int first_idx;
99  } ;
```

3 Консоль

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ ./lab_2  >answ
a
+ A 1
a
+ a 2
+ ab 2
+ aaa 3
! Save ./task_1_data
-a
a
ab
aaa
! Load ./task_1_data
a
! Load ./task_2_data
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ cat answ
NoSuchWord
OK
OK: 1
Exist
OK
OK
OK
OK
NoSuchWord
OK: 2
OK: 3
OK
OK: 1
ERROR: Unable to open file
```

4 Тест производительности

Тест производительности представляет из себя следующее: скорость операции поиска, вставки и удаления в PATRICIA сравнивается со скоростью этих же операций в Красно-Черном дереве, которое реализовано в STL (std::map). Для поиска зависимостей было подготовлено 4 теста, в каждом их которых содержится одинаковое число для операций сравнения, поиска и вставки. (10^3, 10^4, 10^5, 10^6).
, , , ().,, , , .

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ python3 ./test.py tests/
Test # 1  having  1000  elements was created
Test # 2  having  10000  elements was created
Test # 3  having  100000  elements was created
Test # 4  having  1000000  elements was created

// Тестирование
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ g++ benchmark.cpp -o benchmark
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ ./benchmark <tests/01.t
Including. The count of values = 1000
PATRICIA: 0.005715924 s
RB-tree : 0.000767552 s

Deleting. The count of values = 1000
PATRICIA: 0.003349630 s
RB-tree : 0.000499705 s

Searching. The count of values = 1000
PATRICIA: 0.003571526 s
RB-tree : 0.002456841 s

arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ ./benchmark <tests/02.t
Including. The count of values = 10000
PATRICIA: 0.046698127 s
RB-tree : 0.005604166 s

Deleting. The count of values = 10000
PATRICIA: 0.090027202 s
RB-tree : 0.004150235 s

Searching. The count of values = 10000
PATRICIA: 0.076729053 s
```

```
RB-tree : 0.014839451 s

arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ ./benchmark <tests/03.t
Including. The count of values = 100000
PATRICIA: 0.060688123 s
RB-tree : 0.082958423 s

Deleting. The count of values = 100000
PATRICIA: 0.057786544 s
RB-tree : 0.071427491 s

Searching. The count of values = 100000
PATRICIA: 0.128882235 s
RB-tree : 0.166509925 s

arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ ./benchmark <tests/04.t
Including. The count of values = 1000000
PATRICIA: 1.322508116 s
RB-tree : 1.566153116 s

Deleting. The count of values = 1000000
PATRICIA: 1.253254608 s
RB-tree : 1.466430910 s

Searching. The count of values = 1000000
PATRICIA: 2.302821872 s
RB-tree : 2.438060811 s
```

Как видно из теста, патриция, начиная с количества элементов 10^5 везде выигрывает. Это по большей части связано с тем, что красно-чёрное дерево в процессе поиска сравнивает ключи, которые могут длиной 256 символов. Патриция же в процессе поиска сравнивает только один бит ключа, а полностью ключ проверяет только один раз в конце поиска.

5 Профилирование и поиск утечек памяти

Утилита *gprof* предназначена для профилирования программ, т.е. для сбора информации о работе программы, в частности временя работы каждой отдельно взятой функции и количестве ее вызовов.

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ g++ -pg -Wall lab_2.cpp -o
lab_2_profiler
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ ./lab_2_profiler <tests/01.t
>/dev/null
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ gprof ./lab_2_profiler >report.txt
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ head -n 38 report.txt
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  us/call  us/call  name
50.00      0.01     0.01   431090     0.02     0.02  get_bit(char,int)
50.00      0.02     0.01   431090     0.02     0.02  Patricia_trie::get_idx(int)
0.00       0.02     0.00   433086     0.00     0.00  TStr::operator[](int)
0.00       0.02     0.00   431090     0.00     0.00  TMy_pair<int,int>::operator=(TMy_p
0.00       0.02     0.00     3000     0.00     0.00  str_to_char(std::__cxx11::basic_st
0.00       0.02     0.00     3000     0.00     0.00  TStr::TStr(TStr const&)
0.00       0.02     0.00     2999     0.00     0.00  TStr::operator==(TStr const&)
const
0.00       0.02     0.00     2083     0.00     0.00  TStr::operator=(TStr const&)
0.00       0.02     0.00     2081     0.00     4.89  Patricia_trie::TTrie::find_node(Pa
0.00       0.02     0.00     2001     0.00     0.00  TStr::TStr()
0.00       0.02     0.00     2000     0.00     0.00  operator>>(std::istream&,TStr&)
0.00       0.02     0.00     1998     0.00     0.00  Patricia_trie::TTrie::del_node(Pat
0.00       0.02     0.00     1482     0.00     0.00  int const& std::min<int>(int
const&,int const&)
0.00       0.02     0.00     1001     0.00     0.00  Patricia_trie::TTrie::check_root()
0.00       0.02     0.00     1000     0.00     5.26  Patricia_trie::TTrie::erase(TStr)
0.00       0.02     0.00     1000     0.00     9.85  Patricia_trie::TTrie::insert(TStr
long)
0.00       0.02     0.00     1000     0.00     4.89  Patricia_trie::TTrie::search(TStr)
0.00       0.02     0.00     1000     0.00     4.86  Patricia_trie::TTrie::find_PRC(TSt
0.00       0.02     0.00     1000     0.00     0.00  TStr::TStr(std::__cxx11::basic_str
0.00       0.02     0.00     1000     0.00     0.00  TMy_pair<Patricia_trie::TNode*,Pat
0.00       0.02     0.00     1000     0.00     0.00  TMy_pair<bool,unsigned
```

```
long>::TMy_pair(bool,unsigned long)
0.00      0.02     0.00      999     0.00     9.86  Patricia_trie::TTrie::add(Patricia
0.00      0.02     0.00      998     0.00     0.00  first_diff_bit(char,char)
0.00      0.02     0.00      998     0.00     4.93  Patricia_trie::TTrie::find_pair_fo
0.00      0.02     0.00      998     0.00     0.00  Patricia_trie::make_idx(int,int)
0.00      0.02     0.00       82     0.00     0.00  std::remove_reference<TStr&>::type
std::move<TStr&>(TStr&)
0.00      0.02     0.00        1     0.00     0.00  __static_initialization_and_destru
0.00      0.02     0.00        1     0.00     0.00  Patricia_trie::TTrie::root_create(
0.00      0.02     0.00        1     0.00     0.00  Patricia_trie::TTrie::destroy_nods(
0.00      0.02     0.00        1     0.00     0.00  Patricia_trie::TTrie::destroy_tree(
0.00      0.02     0.00        1     0.00     0.00  Patricia_trie::TTrie::TTrie()
0.00      0.02     0.00        1     0.00     0.00  Patricia_trie::TTrie::~TTrie()
```

Видно, что из функций патриции больше всего времени занимает вставка. А самое большое время выделяется на сравнение строк и взятие символа строки.

Утилита *valgrind* может показать, сколько байтов памяти выделено и освобождено, а также укажет на функции, в которых произошла утечка памяти, если таковая имеется.

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ valgrind --leak-check=full
-s ./lab_2 <tests/02.t >/dev/null
==23291== Memcheck,a memory error detector
==23291== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==23291== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==23291== Command: ./lab_2
==23291==
==23291==
==23291== HEAP SUMMARY:
==23291==     in use at exit: 125,544 bytes in 15 blocks
==23291==   total heap usage: 76,931 allocs,76,916 frees,13,922,112 bytes allocated
==23291==
==23291== 296 bytes in 1 blocks are definitely lost in loss record 1 of 8
==23291==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-l
==23291==    by 0x10AF85: Patricia_trie::TTrie::root_create() (in /home/arsenii/Docume
==23291==    by 0x10AF50: Patricia_trie::TTrie::TTrie() (in /home/arsenii/Documents/C_
==23291==    by 0x10CB5C: main (in /home/arsenii/Documents/C_pp_uk/DA/Lab_2/lab_2)
==23291==
==23291== 2,368 bytes in 8 blocks are definitely lost in loss record 2 of 8
```

```
==23291==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-l
==23291==    by 0x10BB8F: Patricia_trie::TTrie::insert(TStr,unsigned long)
(in /home/arsenii/Documents/C_pp_uk/DA/Lab_2/lab_2)
==23291==    by 0x10CC78: main (in /home/arsenii/Documents/C_pp_uk/DA/Lab_2/lab_2)
==23291==
==23291== LEAK SUMMARY:
==23291==    definitely lost: 2,664 bytes in 9 blocks
==23291==    indirectly lost: 0 bytes in 0 blocks
==23291==      possibly lost: 0 bytes in 0 blocks
==23291==    still reachable: 122,880 bytes in 6 blocks
==23291==         suppressed: 0 bytes in 0 blocks
==23291== Reachable blocks (those to which a pointer was found) are not shown.
==23291== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==23291==
==23291== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Как видно при вставке у нас есть утечка памяти. Это связано с тем, что для любой вставки у нас создается промежуточнй узел, а если вставка не удается(т.е. элемент уже существует, память не очищается). Данная ошибка была исправлена:

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ g++ lab_2.cpp -o lab_2
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ valgrind --leak-check=full
-s ./lab_2 <tests/02.t >/dev/null
==23534== Memcheck,a memory error detector
==23534== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==23534== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==23534== Command: ./lab_2
==23534==
==23534==
==23534== HEAP SUMMARY:
==23534==     in use at exit: 123,176 bytes in 7 blocks
==23534==   total heap usage: 76,931 allocs,76,924 frees,13,922,112 bytes allocated
==23534==
==23534== 296 bytes in 1 blocks are definitely lost in loss record 1 of 7
==23534==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-l
==23534==    by 0x10AF85: Patricia_trie::TTrie::root_create() (in /home/arsenii/Docume
==23534==    by 0x10AF50: Patricia_trie::TTrie::TTrie() (in /home/arsenii/Documents/C_
==23534==    by 0x10CB68: main (in /home/arsenii/Documents/C_pp_uk/DA/Lab_2/lab_2)
==23534==
==23534== LEAK SUMMARY:
==23534==    definitely lost: 296 bytes in 1 blocks
```

```
==23534==    indirectly lost: 0 bytes in 0 blocks
==23534==      possibly lost: 0 bytes in 0 blocks
==23534==    still reachable: 122,880 bytes in 6 blocks
==23534==         suppressed: 0 bytes in 0 blocks
==23534== Reachable blocks (those to which a pointer was found) are not shown.
==23534== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==23534==
==23534== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Также была 1 утечка, связанная с неудалением корня дерева. Она также была исправлена:

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ g++ lab_2.cpp -o lab_2
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/Lab_2$ valgrind --leak-check=full
-s ./lab_2 <tests/01.t >/dev/null
==24429== Memcheck,a memory error detector
==24429== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==24429== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==24429== Command: ./lab_2
==24429==
==24429==
==24429== HEAP SUMMARY:
==24429==     in use at exit: 122,880 bytes in 6 blocks
==24429==   total heap usage: 7,734 allocs,7,728 frees,1,578,402 bytes allocated
==24429==
==24429== LEAK SUMMARY:
==24429==    definitely lost: 0 bytes in 0 blocks
==24429==    indirectly lost: 0 bytes in 0 blocks
==24429==      possibly lost: 0 bytes in 0 blocks
==24429==    still reachable: 122,880 bytes in 6 blocks
==24429==         suppressed: 0 bytes in 0 blocks
==24429== Reachable blocks (those to which a pointer was found) are not shown.
==24429== To see them,rerun with: --leak-check=full --show-leak-kinds=all
==24429==
==24429== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

6 Выводы

Выполнив данную лабораторную работу, узнал о структуре данных «Патриция» и о красно-чёрном дереве. Патрицию следует использовать, когда в дереве нужно хранить достаточно большие ключи, ведь патриция при поиске позволяет не проходится по всему ключу, а брать только один бит из этого ключа.

Незадолго до лабораторной работы узнал о профилировании, а в ходе лабораторной работы попрактиковался с профилировщиком grpof. Профилирование помогает собрать информацию о скорости работы функций, что, в свою очередь, можно использовать для оптимизации медленных участков кода.

Узнал об утилите valgrind, одной из функций которой является поиск утечек памяти. Эту утилиту считаю очень полезной, ведь с её помощью можно быстрее обнаружить и «заделать дыры» и во многих случаях существенно сэкономить память.

Список литературы

[1] Гасфилд Дэн
Строки, деревья и последовательности в алгоритмах: Информатика и вычисвычислительная биология / Пер. с англ. И. В. Романовского. — СПб.: Невский Диа- Диалект; БХВ-Петербург, 2003. — 654 с: ил.