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: М. М. Парфенов
Преподаватель: С. А. Михайлова
Группа: М8О-201Б-22

Дата:
Оценка:
Подпись:

Москва, 2024

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 $2^{64} - 1$..

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.

Различия вариантов заключаются только в используемых структурах данных:

Красно-чёрное дерево.

1 Описание

Красно-черное дерево — это разновидность сбалансированного двоичного дерева поиска (BST), где каждый узел имеет дополнительный атрибут: цвет, который может быть либо красным, либо черным. Красно-черные деревья обеспечивают балансировку, которая гарантирует, что время выполнения основных операций (поиск, вставка, удаление) остаётся логарифмическим относительно количества узлов в дереве.

Основные свойства красно-черного дерева:
1. Каждый узел имеет цвет: красный или черный.
2. Корень дерева всегда черный.
3. Листья (NIL-узлы) всегда черные: В данном контексте листья обозначают пустые узлы.
4. Красный узел не может иметь красных детей: Это свойство предотвращает появление последовательностей красных узлов в дереве.
5. Каждый путь от узла до всех его листовых потомков содержит одно и то же количество черных узлов.

Поиск.
Поиск осуществляется также, как и в бинарном дереве поиска. Сложность поиска: O(log n), где n — количество узлов в дереве.

Вставка.
Шаги вставки:
1.Стандартная вставка:
    1. Новый узел всегда вставляется как красный.
    2. Выполняется обычная вставка, как в двоичном дереве поиска.
    3. Новый узел вставляется как лист (ссылки на NIL-узлы).
2. Восстановление свойств:
    1. Если родитель нового узла черный, ничего не требуется.
    2. Если родитель красный, возможны нарушения свойств дерева.
    3.Восстановление включает перекраску и повороты:
        Случай 1: Дядя нового узла красный — перекрашиваем дядю и родителя в черный, дедушку в красный и повторяем для дедушки.
        Случай 2: Дядя черный и новый узел находится на одной стороне с родителем — выполняется один поворот.
        Случай 3: Дядя черный и новый узел на противоположной стороне от родителя — выполняется два поворота и перекраска.
Сложность вставки: O(log n)

Удаление.

Шаги удаления:
1.Стандартное удаление:

1.Найдите узел, который нужно удалить.

2.Если у узла два дочерних узла, замените его на узел с минимальным значением в правом поддереве (преемник).

2. Восстановление свойств:

1.Если удаленный узел или его единственный потомок красный, просто перекрашиваем оставшийся узел в черный.

2.Если удаленный узел черный и его единственный потомок черный, это может нарушить баланс дерева.

3.Восстановление включает перекраску и повороты:

Случай 1: Брат узла красный — перекрашиваем брата в черный и родителя в красный, затем выполняем поворот.

Случай 2: Брат черный и оба его дочерних узла черные — перекрашиваем брата в красный и повторяем для родителя.

Случай 3: Брат черный и один из его дочерних узлов красный — выполняем поворот и перекраску, чтобы восстановить свойства.

Сложность удаления: O(log n)

2 Исходный код

Основные функции и методы
Сравнение строк
Метод compareString сравнивает две строки по длине и лексикографически.

Преобразование строки к нижнему регистру
Метод toLower преобразует все символы строки к нижнему регистру.

Левый поворот
Метод leftRotate выполняет левый поворот поддерева с корнем в заданном узле.

Правый поворот
Метод rightRotate выполняет правый поворот поддерева с корнем в заданном узле.

Восстановление свойств дерева после вставки
Метод fixInsertRBTree восстанавливает свойства красно-черного дерева после вставки нового узла.

Восстановление свойств дерева после удаления
Метод fixDeleteRBTree восстанавливает свойства красно-черного дерева после удаления узла.

Сохранение дерева в файл
Метод saveFile сохраняет дерево в бинарный файл с предварительной записью длины ключа, ключа и значения каждого узла.

Загрузка дерева из файла
Метод loadFile загружает данные из бинарного файла и восстанавливает дерево.

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef bool Tcolor;
5  const Tcolor RED = true;
6  const Tcolor BLACK = false;
7
8  typedef unsigned long long ull;
9
10 struct Node{
```

```
11      Node * parent;
12      Node * left;
13      Node * right;
14      Tcolor color;
15      string key;
16      ull value;
17  };
18
19
20  class RBTree
21  {
22      private:
23          Node *NullNode;
24      protected:
25          void rotateLeft(Node *);
26          void rotateRight(Node *);
27          Node *minRightNode(Node *);
28          Node *maxLeftNode(Node *);
29          void fixInsertRBTree(Node *);
30          void fixDeleteRBTree(Node *);
31          int compareString(const string &, const string &);
32      public:
33          Node *root;
34          RBTree();
35          ~RBTree();
36          string insertValue(const string&, ull);
37          string deleteValue(const string&);
38
39          void erace(Node * x);
40          void get(const string &key);
41
42          void saveFile(ofstream &file, Node * x);
43          void loadFile(ifstream &file);
44
45          void print(Node * x, int level);
46
47          static std::string toLower(std::string str);
48  };
49
50
51  RBTree::RBTree(){
52      NullNode = new Node();
53      NullNode->color = BLACK;
54      root = NullNode;
55  }
56
57  RBTree::~RBTree(){
58      erace(root);
59      delete NullNode;
```

```
60 }
61
62 void RBTree::rotateLeft(Node * x){
63     Node * xRight = x->right;
64     x->right = xRight->left;
65     if (xRight->left != NullNode) {
66         xRight->left->parent = x;
67     }
68     xRight->parent = x->parent;
69     if (x->parent == nullptr) {
70         root = xRight;
71     }
72     else if (x == x->parent->left) {
73         x->parent->left = xRight;
74     }
75     else {
76         x->parent->right = xRight;
77     }
78     xRight->left = x;
79     x->parent = xRight;
80 }
81
82 void RBTree::rotateRight(Node * x){
83     Node * xLeft = x->left;
84     x->left = xLeft->right;
85     if (xLeft->right != NullNode) {
86         xLeft->right->parent = x;
87     }
88     xLeft->parent = x->parent;
89     if (x->parent == nullptr) {
90         root = xLeft;
91     }
92     else if (x == x->parent->right) {
93         x->parent->right = xLeft;
94     }
95     else {
96         x->parent->left = xLeft;
97     }
98     xLeft->right = x;
99     x->parent = xLeft;
100 }
101
102 Node * RBTree::minRightNode(Node * x){
103     while(x->left != NullNode){
104         x = x->left;
105     }
106     return x;
107 }
108
```

```
109 Node * RBTree::maxLeftNode(Node * x){
110     while(x->right != NullNode){
111         x = x->right;
112     }
113     return x;
114 }
115 
116 void RBTree::fixInsertRBTree(Node * x){
117     while(x->parent != nullptr && x->parent->color == RED){
118         if(x->parent == x->parent->parent->left){
119             Node * uncle = x->parent->parent->right;
120             if(uncle->color == RED){
121                 uncle->color = BLACK;
122                 x->parent->color = BLACK;
123                 x->parent->parent->color = RED;
124                 x = x->parent->parent;
125             } else {
126                 if(x == x->parent->right){
127                     x = x->parent;
128                     rotateLeft(x);
129                 }
130                 x->parent->color = BLACK;
131                 x->parent->parent->color = RED;
132                 rotateRight(x->parent->parent);
133             }
134         } else {
135             Node * uncle = x->parent->parent->left;
136             if(uncle->color == RED){
137                 uncle->color = BLACK;
138                 x->parent->color = BLACK;
139                 x->parent->parent->color = RED;
140                 x = x->parent->parent;
141             } else {
142                 if(x == x->parent->left){
143                     x = x->parent;
144                     rotateRight(x);
145                 }
146                 x->parent->color = BLACK;
147                 x->parent->parent->color = RED;
148                 rotateLeft(x->parent->parent);
149             }
150         }
151     }
152     root->color = BLACK;
153 }
154 
155 
156 void RBTree::fixDeleteRBTree(Node * x) {
157     Node * bro;
```

```
158     while(x != root && x->color == BLACK){
159         if(x == x->parent->left){
160             bro = x->parent->right;
161             if(bro->color == RED){
162                 bro->color = BLACK;
163                 x->parent->color = RED;
164                 rotateLeft(x->parent);
165                 bro = x->parent->right;
166             }
167             if(bro->left->color == BLACK && bro->right->color == BLACK){
168                 bro->color = RED;
169                 x = x->parent;
170             } else {
171                 if(bro->right->color == BLACK){
172                     bro->left->color = BLACK;
173                     bro->color = RED;
174                     rotateRight(bro);
175                     bro = x->parent->right;
176                 }
177                 bro->color = x->parent->color;
178                 x->parent->color = BLACK;
179                 bro->right->color = BLACK;
180                 rotateLeft(x->parent);
181                 x = root;
182             }
183         } else {
184             bro = x->parent->left;
185             if(bro->color == RED){
186                 bro->color = BLACK;
187                 x->parent->color = RED;
188                 rotateRight(x->parent);
189                 bro = x->parent->left;
190             }
191             if(bro->right->color == BLACK && bro->left->color == BLACK){
192                 bro->color = RED;
193                 x = x->parent;
194             } else {
195                 if(bro->left->color == BLACK){
196                     bro->right->color = BLACK;
197                     bro->color = RED;
198                     rotateLeft(bro);
199                     bro = x->parent->left;
200                 }
201                 bro->color = x->parent->color;
202                 x->parent->color = BLACK;
203                 bro->left->color = BLACK;
204                 rotateRight(x->parent);
205                 x = root;
206             }
```

```
207         }
208     }
209     x->color = BLACK;
210 }
211 
212 string RBTree::insertValue(const string& key, ull value){
213     Node * newNode = new Node();
214     newNode->key = key;
215     newNode->value = value;
216     newNode->left = NullNode;
217     newNode->right = NullNode;
218     newNode->parent = nullptr;
219     newNode->color = RED;
220 
221     Node * curr = root;
222     Node * currPar = nullptr;
223 
224     while(curr != NullNode){
225         currPar = curr;
226         int comp = compareString(curr->key, key);
227         if(comp == 1){
228             curr = curr->left;
229         } else if (comp == -1){
230             curr = curr->right;
231         } else {
232             delete newNode;
233             return "Exist\n";
234         }
235     }
236 
237     newNode->parent = currPar;
238     if(currPar == nullptr){
239         root = newNode;
240     } else {
241         int comp = compareString(newNode->key, currPar->key);
242         if(comp == 1){
243             currPar->right = newNode;
244         } else {
245             currPar->left = newNode;
246         }
247     }
248 
249     if(newNode->parent == nullptr){
250         newNode->color = BLACK;
251         return "OK\n";
252     } else if(newNode->parent->parent == nullptr){
253         return "OK\n";
254     }
255     fixInsertRBTree(newNode);
```

```
256     return "OK\n";
257 }
258 
259 string RBTree::deleteValue(const string &key){
260     Node * curr = root;
261     Node * rem = NullNode, * fix, * temp;
262     while(curr != NullNode){
263         int comp = compareString(curr->key, key);
264         if(comp == 1){
265             curr = curr->left;
266         } else if (comp == -1){
267             curr = curr->right;
268         } else {
269             rem = curr;
270             break;
271         }
272     }
273 
274     if(rem == NullNode){
275         return "NoSuchWord\n";
276     }
277 
278     temp = rem;
279     Tcolor TempColor = temp->color;
280     if(rem->left == NullNode){
281         fix = rem->right;
282         if(rem->parent == nullptr){
283             root = rem->right;
284         } else {
285             if(rem == rem->parent->left){
286                 rem->parent->left = rem->right;
287             } else {
288                 rem->parent->right = rem->right;
289             }
290         }
291         rem->right->parent = rem->parent;
292     } else if(rem->right == NullNode){
293         fix = rem->left;
294         if(rem->parent == nullptr){
295             root = rem->left;
296         } else {
297             if(rem == rem->parent->left){
298                 rem->parent->left = rem->left;
299             } else {
300                 rem->parent->right = rem->left;
301             }
302         }
303         rem->left->parent = rem->parent;
304     } else {
```

```
305         temp = minRightNode(rem->right);
306         TempColor = temp->color;
307         fix = temp->right;
308         if(temp->parent == rem){
309             fix->parent = temp;
310         } else {
311             if(temp->parent == nullptr){
312                 root = temp->right;
313             } else {
314                 if(temp == temp->parent->left){
315                     temp->parent->left = temp->right;
316                 } else {
317                     temp->parent->right = temp->right;
318                 }
319             }
320             temp->right->parent = temp->parent;
321             temp->right = rem->right;
322             temp->right->parent = temp;
323         }
324         if(rem->parent == nullptr){
325             root = temp;
326         } else {
327             if(rem == rem->parent->left){
328                 rem->parent->left = temp;
329             } else {
330                 rem->parent->right = temp;
331             }
332         }
333         temp->parent = rem->parent;
334         temp->left = rem->left;
335         temp->left->parent = temp;
336         temp->color = rem->color;
337     }
338     delete rem;
339     if(TempColor == BLACK){
340         fixDeleteRBTree(fix);
341     }
342     return "OK\n";
343 }
344 
345 void RBTree::erace(Node * x){
346     if(x == NullNode){
347         return;
348     }
349     if(x->left != NullNode){
350         erace(x->left);
351     }
352     if(x->right != NullNode){
353         erace(x->right);
```

```
354     }
355     delete x;
356     root = NullNode;
357 }
358 
359 void RBTree::get(const string &key){
360     Node * curr = root;
361     while(curr != NullNode){
362         int comp = compareString(curr->key, key);
363         if(comp == 1){
364             curr = curr->left;
365         } else if (comp == -1){
366             curr = curr->right;
367         } else {
368             cout << "OK: " << curr->value << "\n";
369             return;
370         }
371     }
372     cout << "NoSuchWord\n";
373 }
374 
375 void RBTree::saveFile(ofstream &file, Node * x){
376     if(x == NullNode){
377         return;
378     } else {
379         size_t n = x->key.size();
380         string key = x->key;
381         ull value = x->value;
382         file.write((char *)&n, sizeof(size_t));
383         file.write(key.c_str(), sizeof(char) * n);
384         file.write((char *)&value, sizeof(ull));
385         saveFile(file, x->left);
386         saveFile(file, x->right);
387     }
388 }
389 
390 void RBTree::loadFile(ifstream &file){
391     erace(root);
392     if(file.peek() == EOF){
393         cout << "OK\n";
394         return;
395     }
396     size_t n;
397     string key;
398     ull value;
399     file.read((char *)&n, sizeof(size_t));
400     while(n != -1){
401         key.clear();
402         key.resize(n);
```

```
403         file.read((char *)key.data(), sizeof(char) * n);
404         file.read((char *)&value, sizeof(ull));
405         insertValue(key, value);
406         file.read((char *)&n, sizeof(size_t));
407     }
408     cout << "OK\n";
409 }
410 
411 void RBTree::print(Node * x, int level){
412     if(x != NullNode){
413         cout << level << " " << x->key << " " << x->value << " " << x->color << "\n";
414         level++;
415         print(x->left, level);
416         print(x->right, level);
417         cout << "--" << x->key << "--\n";
418     }
419 }
420 
421 std::string RBTree::toLower(std::string str) {
422     std::transform(str.begin(), str.end(), str.begin(), ::tolower);
423     return str;
424 }
425 
426 int RBTree::compareString(const string &x, const string &y){
427     if(x.length() > y.length()){
428         return 1;
429     } else if (x.length() < y.length()){
430        return -1;
431     } else {
432     for(int i = 0; i < x.length(); i++){
433         if(x[i] > y[i]){
434             return 1;
435         } else if (x[i] < y[i]){
436             return -1;
437         }
438     }
439     }
440     return 0;
441 }
442 
443 int main() {
444     // ios::sync_with_stdio(false);
445     // cin.tie(0);
446     RBTree tree;
447     string command, key, path, answer;
448     ull value;
449     while (cin >> command) {
450         try {
451             if (command == "+") {
```

```
452                 cin >> key >> value;
453                 key = tree.toLower(key);
454                 answer = tree.insertValue(key, value);
455                 cout << answer;
456             } else if (command == "-") {
457                 cin >> key;
458                 key = tree.toLower(key);
459                 answer = tree.deleteValue(key);
460                 cout << answer;
461             } else if (command == "!") {
462                 cin >> command;
463                 if (command == "Save") {
464                     cin >> path;
465                     ofstream file;
466                     file.open(path, ios_base::binary);
467                     if (!file) throw runtime_error("Unable to open file for writing");
468                     tree.saveFile(file, tree.root);
469                     cout << "OK\n";
470                     size_t i = -1;
471                     file.write((char *)&i, sizeof(size_t));
472                     file.close();
473                 } else if (command == "Load") {
474                     cin >> path;
475                     ifstream file;
476                     file.open(path, ios_base::binary);
477                     if (!file) throw runtime_error("Unable to open file for reading");
478                     tree.loadFile(file);
479                     file.close();
480                 }
481             } else {
482                 key = tree.toLower(command);
483                 tree.get(key);
484             }
485         } catch (const bad_alloc &) {
486             cout << "ERROR: Not enough memory\n";
487         } catch (const runtime_error &e) {
488             cout << "ERROR: " << e.what() << '\n';
489         } catch (...) {
490             cout << "ERROR: n";
491         }
492     }
493     return 0;
494 }
```

3 Консоль

```
[±main ]-> cat input.txt                                    ~/Documents/MAI/ [17:43]
```

```
+ a 1
+ A 2
+ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
A
- A
a
 [±main ]-> g++ main.cpp                                    ~/Documents/MAI/ [17:43]
 [±main ]-> ./a.out < input.txt                             ~/Documents/MAI/ [17:43]
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

4 Выводы

Лабораторная работа оказалась довольно сложной. Реализация красно-черного дерева потребовала много сил и времени. Однако красно-черное дерево - одна из самых популярных структур данных для реализации словарей, множеств и т.д. Благодаря этой лабораторной работе, я разобрался с работой красно-черного дерева. Теперь, при использовании различных контейнеров, в которых применяется эта структура данных, я буду понимать, как они работают.

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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008