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. Ю. Курносов
Преподаватель: А. А. Кухтичев
Группа: М8О-208Б

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 264 - 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.

Различия вариантов заключаются только в используемых структурах данных:

Вариант 2:Красно-чёрное дерево.

Примеры: Входные данные:

+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
− A
a
Результат работы:
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord

1 Описание

Требуется написать реализацию структуры красночёрного дерева, которое будет хранить введённые значения,а так же запрограммировать интерфейс для взаимодействия пользователя с программой-словарём в соответствии с заданием.
Основная идея красно-чёрного дерева — это двоичное дерево поиска, в котором каждый узел имеет атрибут цвета. При этом:
Узел может быть либо красным, либо чёрным и имеет двух потомков;
Корень — как правило чёрный. Это правило слабо влияет на работоспособность модели, так как цвет корня всегда можно изменить с красного на чёрный;
Все листья, не содержащие данных — чёрные.
Оба потомка каждого красного узла — чёрные.
Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.
Благодаря этим ограничениям, путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего и дерево примерно сбалансировано. Операции вставки, удаления и поиска требуют в худшем случае времени, пропорционального длине дерева, что позволяет красно-чёрным деревьям быть более эффективными в худшем случае, чем обычные двоичные деревья поиска..

2 Исходный код

Программа считывает значения из файла, подающегося на поток, и выполняет действие в соответствии с заданием т. е. добавляет элемент в словарь или удаляет его оттуда, или находит элемент по ключу и выводит его значение, так же программа может сохранить содержимое словаря в файл или загрузить из файла словарь(предполагается что файл взаранее сохранён программой).

На каждой непустой строке входного файла располагается тройка «команда-ключ-значение» или ключ по которому надо найти элемент в словаре поэтому создадим новую структуру $TNode$, в которой будем хранить ключ и значение и параметры нужные программе для словаря. Для хранения пар создадим структуру (дерево) $TTree$. Для взаимодействия пользователя со словарём напишем интерфейс в $main$. После выполнения каждой команды программа выводит сообщение в соответствии с заданием.

solution.c:

```
// This is a personal academic project. Dear PVS-Studio, please check it.
// PVS-Studio Static Code Analyzer for C, C++ and C#: http://www.viva64.com
#include <ctype.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

const int DATA_STR_LEN = 257;
const int RED = 1;
const int VAL_L = 21;
typedef unsigned long long int TVAL;

struct TNode
{
    TVAL val;
    int red;
    char *data;
    struct TNode *ref[2];
};
struct TTree
{
    struct TNode *root;
};
short int TRm(struct TTree *tr, char *data);
struct TNode *RRm(struct TNode *root, char *data, int *done);
struct TNode *RmBalance(struct TNode *root, int dir, int *done);
short int IsR(struct TNode *root);
short int TInsert(struct TTree *tr, const char *data, TVAL val);
struct TNode *RInsert(struct TNode *root, const char *data, TVAL val);
```

```
32 void TFind(struct TNode *root, const char *data);
33 struct TNode *Rotate2(struct TNode *root, int dir);
34 short int IsAlpha(const char *str);
35 short int DeStructPrint(struct TNode *root, FILE *f);
36 void StructPrint(struct TNode *root, FILE *f);
37 struct TNode *ConstructTNode(const char *data, TVAL val);
38 struct TNode *NodeConstruct(const char *data, TVAL val, int red);
39
40 void DelTree(struct TNode *root)
41 {
42     if (root)
43     {
44         DelTree(root->ref[0]);
45         DelTree(root->ref[1]);
46         free(root->data);
47         free(root);
48         root = NULL;
49     }
50 }
51 short int IsR(struct TNode *root)
52 {
53     return root != NULL && root->red == RED;
54 }
55 struct TNode *Rotate1(struct TNode *root, int dir)
56 {
57     struct TNode *n = root->ref[!dir];
58     root->ref[!dir] = n->ref[dir];
59     n->ref[dir] = root;
60     root->red = 1;
61     n->red = 0;
62     return n;
63 }
64 struct TNode *Rotate2(struct TNode *root, int dir)
65 {
66     root->ref[!dir] = Rotate1(root->ref[!dir], !dir);
67     return Rotate1(root, dir);
68 }
69 struct TNode *ConstructTNode(const char *data, TVAL val)
70 {
71     struct TNode *n = malloc(sizeof *n);
72     n->data = malloc(DATA_STR_LEN);
73     if (n != NULL)
74     {
75         strcpy(n->data, data);
76         n->val = val;
77         n->red = 1;
78         n->ref[0] = NULL;
79         n->ref[1] = NULL;
80     }
```

```
81      return n;
82  }
83
84  struct TNode *NodeConstruct(const char *data, TVAL val, int red)
85  {
86      struct TNode *n = malloc(sizeof *n);
87      n->data = malloc(DATA_STR_LEN);
88      if (n != NULL)
89      {
90          strcpy(n->data, data);
91          n->val = val;
92          n->red = red;
93          n->ref[0] = NULL;
94          n->ref[1] = NULL;
95      }
96      return n;
97  }
98
99  void TFind(struct TNode *root, const char *data)
100 {
101     if (root == NULL)
102     {
103         printf("NoSuchWord\n");
104         return;
105     }
106     short int n = strcmp(data, root->data);
107     int dir = 0;
108     if (n == 0)
109     {
110         printf("OK: %llu\n", root->val);
111     }
112     else
113     {
114         dir = n > 0;
115         TFind(root->ref[dir], data);
116     }
117     return;
118 }
119
120 struct TNode *RInsert(struct TNode *root, const char *data, TVAL val)
121 {
122     if (root == NULL)
123     {
124         root = ConstructTNode(data, val);
125         if (root != NULL)
126         {
127             printf("OK\n");
128         }
129         else
```

```
130         {
131             printf("ERROR: Allocate error\n");
132         }
133     }
134     else
135     {
136         int dir;
137         short int n = strcmp(data, root->data);
138         if (n == 0)
139         {
140             printf("Exist \n");
141             return root;
142         }
143         else
144         {
145             dir = n > 0;
146         }
147         root->ref[dir] = RInsert(root->ref[dir], data, val);
148         if (IsR(root->ref[dir]))
149         {
150             if (IsR(root->ref[!dir]))
151             {
152                 root->red = 1;
153                 root->ref[0]->red = 0;
154                 root->ref[1]->red = 0;
155             }
156             else
157             {
158                 if (IsR(root->ref[dir]->ref[dir]))
159                     root = Rotate1(root, !dir);
160                 else if (IsR(root->ref[dir]->ref[!dir]))
161                     root = Rotate2(root, !dir);
162             }
163         }
164     }
165     return root;
166 }
167 short int TInsert(struct TTree *tr, const char *data, TVAL val)
168 {
169     tr->root = RInsert(tr->root, data, val);
170     tr->root->red = 0;
171     return 0;
172 }
173
174 struct TNode *RRm(struct TNode *root, char *data, int *done)
175 {
176     if (root == NULL)
177     {
178         printf("NoSuchWord\n");
```

```
179         *done = 1;
180     }
181     else
182     {
183         int dir;
184         short int n = strcmp(data, root->data);
185         if (n == 0)
186         {
187             if (root->ref[0] == NULL || root->ref[1] == NULL)
188             {
189                 struct TNode *save = root->ref[root->ref[0] == NULL];
190                 if (IsR(root))
191                 {
192                     *done = 1;
193                 }
194                 else if (IsR(save))
195                 {
196                     save->red = 0;
197                     *done = 1;
198                 }
199                 printf("OK\n");
200                 free(root->data);
201                 free(root);
202                 root = NULL;
203                 return save;
204             }
205             else
206             {
207                 struct TNode *nod = root->ref[0];
208                 while (nod->ref[1] != NULL)
209                 {
210                     nod = nod->ref[1];
211                 }
212 
213                 strcpy(root->data, nod->data);
214                 strcpy(data, nod->data);
215                 root->val = nod->val;
216             }
217         }
218         n = strcmp(data, root->data);
219         dir = n > 0;
220         root->ref[dir] = RRm(root->ref[dir], data, done);
221         if (!*done)
222         {
223             root = RmBalance(root, dir, done);
224         }
225     }
226     return root;
227 }
```

```
228 short int TRm(struct TTree *tree1, char *data)
229 {
230     int done = 0;
231     tree1->root = RRm(tree1->root, data, &done);
232     if (tree1->root != NULL)
233     {
234         tree1->root->red = 0;
235     }
236     return 0;
237 }
238 
239 struct TNode *RmBalance(struct TNode *root, int dir, int *done)
240 {
241     struct TNode *p = root;
242     struct TNode *s = root->ref[!dir];
243     if (IsR(s))
244     {
245         root = Rotate1(root, dir);
246         s = p->ref[!dir];
247     }
248     if (s != NULL)
249     {
250         if (!IsR(s->ref[0]) && !IsR(s->ref[1]))
251         {
252             if (IsR(p))
253             {
254                 *done = 1;
255             }
256             p->red = 0;
257             s->red = 1;
258         }
259         else
260         {
261             int save = p->red;
262             int new_root = (root == p);
263             if (IsR(s->ref[!dir]))
264                 p = Rotate1(p, dir);
265             else
266                 p = Rotate2(p, dir);
267             p->red = save;
268             p->ref[0]->red = 0;
269             p->ref[1]->red = 0;
270             if (new_root)
271             {
272                 root = p;
273             }
274             else
275             {
276                 root->ref[dir] = p;
```

```
277             }
278 
279             *done = 1;
280         }
281     }
282     return root;
283 }
284 
285 short int IsAlpha(const char *str)
286 {
287     for (int i = 0; i < strlen(str); i++)
288     {
289         if (!isalpha(str[i]))
290         {
291             return 0;
292         }
293     }
294     return 1;
295 }
296 
297 TVAL Dig(const char *val1)
298 {
299     TVAL val = 0;
300     for (int i = 0; i < strlen(val1); i++)
301     {
302         val = val * 10 + (val1[i] - '0');
303     }
304     return val;
305 }
306 
307 void StructPrint(struct TNode *root, FILE *f)
308 {
309     if (root->ref[0])
310     {
311         fprintf(f, "L\n");
312         StructPrint(root->ref[0], f);
313     }
314     if (root->ref[1])
315     {
316         fprintf(f, "R\n");
317         StructPrint(root->ref[1], f);
318     }
319     fprintf(f, "%s\n%llu\n%d\n", root->data, root->val, root->red);
320     return;
321 }
322 
323 unsigned short int SaveTr(struct TTree *tr, const char *path)
324 {
325     FILE *f = fopen(path, "wb");
```

```
326     if (!f)
327     {
328         return 1;
329     }
330     if (tr->root)
331     {
332         fputs("[\n", f);
333         StructPrint(tr->root, f);
334     }
335     else
336     {
337         fputs("[", f);
338         fputs("\n", f);
339         fputs("]", f);
340         fclose(f);
341         printf("OK\n");
342         return 0;
343     }
344     fputs("]", f);
345     fclose(f);
346     printf("OK\n");
347     return 0;
348 }
349 
350 short int DeStructPrint(struct TNode *root, FILE *f)
351 {
352     char str[DATA_STR_LEN];
353     TVAL val = 0;
354     int red = 0, err = 0;
355     if (fscanf(f, "%s", str) != EOF)
356     {
357         if (str[0] == 'L')
358         {
359             root->ref[0] = NodeConstruct("\0", 0, 0);
360             if (root->ref[0] == NULL)
361             {
362                 printf("ERROR: Allocate error\n");
363                 return 2;
364             }
365             err = DeStructPrint(root->ref[0], f);
366             if (err == 2)
367             {
368                 return 2;
369             }
370             if (fscanf(f, "%s", str) == EOF)
371             {
372                 printf("ERROR: Invalid data\n");
373                 return 2;
374             }
```

```
375         }
376         if (str[0] == 'R')
377         {
378             root->ref[1] = NodeConstruct("\0", 0, 0);
379             if (root->ref[1] == NULL)
380             {
381                 printf("ERROR: Allocate error\n");
382                 return 2;
383             }
384             err = DeStructPrint(root->ref[1], f);
385             if (err == 2)
386             {
387                 return 2;
388             }
389             if (fscanf(f, "%s", str) == EOF)
390             {
391                 printf("ERROR: Invalid data\n");
392                 return 2;
393             }
394         }
395         if (str[0] == ']')
396         {
397             return 1;
398         }
399 
400         if (IsAlpha(str))
401         {
402             if (fscanf(f, "%llu %d", &val, &red) != EOF)
403             {
404                 root->val = val;
405                 root->red = red;
406                 strcpy(root->data, str);
407             }
408             else
409             {
410                 printf("ERROR: Invalid data\n");
411                 return 2;
412             }
413         }
414     }
415     else
416     {
417         printf("ERROR: Invalid data\n");
418         return 2;
419     }
420     return 0;
421 }
422 short int LoadTree(struct TTree *tr, const char *path)
423 {
```

```
424     FILE *f = fopen(path, "rb");
425     if (!f)
426     {
427         return 1;
428     }
429     const short int TWO_SYMBOL = 2;
430     char str[TWO_SYMBOL];
431     short err = 0;
432     if (fscanf(f, "%s", str) != EOF)
433     {
434         if (str[0] == '[')
435         {
436             tr->root = NodeConstruct("\0", 0, 0);
437             if (tr->root == NULL)
438             {
439                 printf("ERROR: Allocate error\n");
440                 return 2;
441             }
442             err = DeStructPrint(tr->root, f);
443             if (err == 2)
444             {
445                 return 2;
446             }
447             if (fscanf(f, "%s", str) != EOF || err == 1)
448             { // ]
449                 if (str[0] == ']' || err == 1)
450                 {
451                     printf("OK\n");
452                 }
453                 else
454                 {
455                     printf("ERROR: Wrong file 1\n");
456                 }
457             }
458             else
459             {
460                 printf("ERROR: Wrong file 2\n");
461             }
462         }
463         else
464         {
465             printf("ERROR: Wrong file 3\n");
466         }
467     }
468     else
469     {
470         fclose(f);
471         return 1;
472     }
```

```
473     fclose(f);
474     return 0;
475 }
476 
477 char Ser(char c)
478 {
479     while (c != '\n')
480     {
481         c = tolower(getchar());
482     }
483     c = tolower(getchar()); // a
484     printf("ERROR: Invalid data\n");
485     return c;
486 }
487 
488 int main()
489 {
490     TVAL val = 0;
491     char str[DATA_STR_LEN];
492     char val1[VAL_L];
493     char c;
494     struct TTree tr;
495     tr.root = NULL;
496     short err = 0;
497     int i = 0;
498     c = tolower(getchar());
499 
500     while (c > 0)
501     {
502         if (c == '\n')
503         {
504             c = tolower(getchar());
505             continue;
506         }
507         memset(val1, 0, VAL_L);
508         memset(str, 0, DATA_STR_LEN); //
509         i = 0;
510         if (c == '+')
511         { //
512             if (tolower(getchar()) != ' ')
513             { //
514                 c = Ser(c); //
515                 continue;
516             }
517             c = tolower(getchar());
518             while (c >= 'a' && c <= 'z')//
519             { //
520                 str[i] = c;
521                 ++i;
```

```
522                 c = tolower(getchar());
523             }
524             i = 0;
525             if (c != ' ')
526             {
527                 c = Ser(c);
528                 continue;
529             }
530             c = tolower(getchar());
531             while (c >= '0' && c <= '9')//
532             {
533                 val1[i] = c;
534                 ++i;
535                 c = tolower(getchar());
536             }
537             if (c != '\n')
538             {
539                 c = Ser(c);
540                 continue;
541             }
542             val = Dig(val1);
543             TInsert(&tr, str, val);
544         }
545         else if (c == '-')
546         { //
547             if (tolower(getchar()) != ' ')
548             {
549                 c = Ser(c);
550                 continue;
551             }
552             c = tolower(getchar());
553             while (c >= 'a' && c <= 'z')
554             {
555                 str[i] = c;
556                 ++i;
557                 c = tolower(getchar());
558             }
559             if (c != '\n')
560             {
561                 c = Ser(c);
562                 continue;
563             }
564             TRm(&tr, str);
565         }
566         else if (c >= 'a' && c <= 'z')
567         { //
568             while (c >= 'a' && c <= 'z')
569             {
570                 str[i] = c;
```

```
571                 ++i;
572                 c = tolower(getchar());
573             }
574             if (c != '\n')
575             {
576                 c = Ser(c);
577                 continue;
578             }
579             TFind(tr.root, str);
580         }
581         else if (c == '!')
582         { //
583             if (tolower(getchar()) != ' ')
584             {
585                 c = Ser(c);
586                 continue;
587             }
588             if (scanf("%s", str) != EOF)
589             {
590                 if (strcmp(str, "Save") == 0)
591                 { // ! Save /path/to/file
592                     if (scanf("%s", str) != EOF)
593                     {
594                         err = SaveTr(&tr, str);
595                         if (err > 0)
596                         {
597                             printf("ERROR: Wrong saving\n");
598                         }
599                     }
600                     else
601                     {
602                         printf("ERROR: Wrong file name\n");
603                     }
604                 }
605                 else if (strcmp(str, "Load") == 0)
606                 { //! Load /path/to/file
607                     struct TTree tmproot;
608                     tmproot.root = NULL;
609                     if (scanf("%s", str) != EOF)
610                     {
611                         err = LoadTree(&tmproot, str);
612                         if (err > 0)
613                         {
614                             printf("ERROR: Wrong load\n");
615                         }
616                         if ((tmproot.root) && (err == 0))
617                         {
618                             DelTree(tr.root);
619                             tr.root = tmproot.root;
```

```
620                         tmproot.root = NULL;
621                     }
622                     else
623                     {
624                         DelTree(tmproot.root);
625                         tmproot.root = NULL;
626                     }
627                     if (tmproot.root)
628                     {
629                         DelTree(tmproot.root);
630                     }
631                 }
632                 else
633                 {
634                     printf("ERROR: Wrong file name\n");
635                 }
636             }
637             c = tolower(getchar());
638             if (c != '\n')
639             {
640                 while (c != '\n')
641                 {
642                     c = tolower(getchar());
643                 }
644                 c = tolower(getchar());
645                 continue;
646             }
647         }
648     }
649     else
650     {
651         c = Ser(c);
652         continue;
653     }
654     c = tolower(getchar());
655 }
656 if (tr.root)
657 {
658     DelTree(tr.root);
659 }
660 return 0;
661 }
```

solution.c	
struct TNode	Структура узла, так же используется для хранения значений
struct TTree	Структура дерева, которое я использую для организации словаря и его хранения
short int TRm(struct TTree *tr, char *data)	удаление дерева
struct TNode *RRm(struct TNode *root, char *data, int *done)	удаление элемента из словаря
struct TNode *RmBalance(struct TNode *root, int dir, int *done)	Балансировка после удаления
short int IsR(struct TNode *root)	проверка цвета узла
short int TInsert(struct TTree *tr, const char *data, TVAL val)	запускает вставку в дерево
struct TNode *RInsert(struct TNode *root, const char *data, TVAL val)	вставка в дерево
void TFind(struct TNode *root, const char *data)	поиск элемента в словаре по ключу
struct TNode *Rotate2(struct TNode *root, int dir)	двойной поворот
struct TNode *Rotate1(struct TNode *root, int dir)	поворот в дереве
short int IsAlpha(const char *str)	проверка буквы
short int DeStructPrint(struct TNode *root, FILE *f)	считывание структуры из файла
void StructPrint(struct TNode *root, FILE *f)	печать структуры для сохранения в файл
struct TNode *ConstructTNode(const char *data, TVAL val)	создание узла
struct TNode *NodeConstruct(const char *data, TVAL val, int red)	создание узла при считывании из файла

Листинг структуры TTree:

```
1 struct TTree
2 {
3     struct TNode *root;
4 };
```

Листинг структуры TNode:

```
1 struct TNode
2 {
3     TVAL val;
4     int red;
5     char *data;
6     struct TNode *ref[2];
7 };
```

3 Консоль

```
vorona@DESKTOP-F81SG1A:/mnt/c/Users/Max Kar/Desktop$ gcc -o p solution.c
vorona@DESKTOP-F81SG1A:/mnt/c/Users/Max Kar/Desktop$ ./p
+ f 77
OK
+ ror 99
OK
+ ghf 8
OK
ghf
OK: 8
f
OK: 77
ror
OK: 99
! Save save1
OK
! Load file
OK
f
OK: 7
r
OK: 11
g
OK: 8
ror
NoSuchWord
```

4 Тест производительности

Тест производительности представляет из себя следующее: Вставка большого количества значений из файла с тестовыми значениями или вставка большого количества значений и поиск по словарю

```
vorona@DESKTOP-F81SG1A:/mnt/c/Users/Max Kar/Desktop$ time ./p <test1000_0.txt
real    0m0.264s
user    0m0.026s
sys     0m0.124s
vorona@DESKTOP-F81SG1A:/mnt/c/Users/Max Kar/Desktop$ time ./p <test100_0.txt
real    0m0.062s
user    0m0.002s
sys     0m0.025s
vorona@DESKTOP-F81SG1A:/mnt/c/Users/Max Kar/Desktop$ time ./p <test100_1000.txt
real    0m0.198s
user    0m0.013s
sys     0m0.095s
```

5 Выводы

В результате выполнения лабораторной работы по курсу «Дискретный анализ», были получены навыки использования бинарных деревьев поиска, работы с утилитой valgrind и закреплены навыки использования классов и ввода(вывода) в(из) файла(ов).

Список литературы

[1] *Красно-чёрное дерево - Википедия*
URL:https://ru.wikipedia.org/wiki/Красно-чёрное
(дата обращения: 27.10.2020).

[2] *Лекция 33: Решение задач на динамические структуры данных*
URL: https://intuit.ru/studies/courses/648/504/lecture/11459?page=2
(дата обращения: 27.10.2020).

[3] *Хабр:Красно-черные деревья: коротко и ясно*
URL: https://habr.com/ru/post/330644/(дата обращения: 27.10.2020).

[4] *Понимаем красно-чёрное дерево.*
URL: https://tproger.ru/articles/ponimaem-krasno-chjornoe-derevo-chast-1-vvedenie/
(дата обращения: 27.10.2020).