Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Дискретный анализ»

Студент: А. Д. Волков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22

Дата: 05.05.2024
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №3

Для реализации словаря из предыдущей лабораторной работы, необходимо провести исследование скорости выполнения и потребления оперативной памяти. В случае выявления ошибок или явных недочётов, требуется их исправить.

Минимальный набор используемых средств должен содержать утилиту gprof и библиотеку dmalloc, однако их можно заменять на любые другие аналогичные или более развитые утилиты (например, Valgrind или Shark) или добавлять к ним новые (например, gcov).

1 Описание

Утилита grprof, которая позволяет пользователям получить данные профилирования программ на языках C, Pascal и Fortran77. По сути, gprof рассчитывает количество времени, которое проходит в процессе исполнения каждой из процедур или функций. "После этого полученные измерения распределяются по вершинам графа вызовов. В процессе выявляются циклы и вызовам из них ставится в соответствие время исполнения этих циклов."

Valgrind — инструментальное программное обеспечение, предназначенное для отладки использования памяти, обнаружения утечек памяти, а также профилирования.

2 Исходный код

```
#include <iostream>
#include <string>
#include <cstdint>
#include <fstream>

const size_t DEFAULT = 1;

enum TColour
{
    red,
    black,
};

template <class T>
class TQueue
{
public:
    T *buffer;
    size_t size;
    size_t capacity;
    size_t head;
    TQueue()
    {
        buffer = new T[DEFAULT];
        size = 0;
        capacity = 1;
        head = 0;
    }
    void Expand()
    {
        T *new_buffer = new T[capacity * 2];
        capacity *= 2;
        for (std::size_t i = 0; i < size; ++i)
        {
            new_buffer[i] = buffer[i];
        }
        delete[] buffer;
        buffer = new_buffer;
    }
    void PushBack(T& value)
    {
        if (size == capacity)
        {
            Expand();
        }
        buffer[size] = value;
        ++size;
```

```
48      }
49      T PopFront()
50      {
51          T res = buffer[head];
52          size--;
53          head++;
54          return res;
55      }
56      size_t Size()
57      {
58          return size;
59      }
60      T& operator[](std::size_t idx)
61      {
62          return buffer[idx + head];
63      }
64      void Clear()
65      {
66          delete[] buffer;
67          size = 0;
68          head = 0;
69          capacity = 1;
70          buffer = new T[capacity];
71      }
72      ~TQueue()
73      {
74          if (size > 0)
75          {
76              delete[] buffer;
77          }
78      }
79  };
80
81  class TNode
82  {
83  public:
84      std::string key;
85      uint64_t value;
86      TColour colour = TColour::black;
87      TNode *left = nullptr;
88      TNode *right = nullptr;
89      TNode *parent = nullptr;
90      TNode() = default;
91      TNode(std::string key, uint64_t value);
92  };
93
94  class TRBTree
95  {
96  public:
```

```
97      TNode *root = nullptr;
98      TNode *nil;
99      TRBTree();
100     bool Insert(std::string key, uint64_t value);
101     bool Erase(std::string key);
102     void Save(std::ofstream& file);
103     void Load(std::ifstream& file);
104     TNode *FindNode(std::string key);
105     TNode *GetNil();
106     ~TRBTree();
107 private:
108     TNode *FindMinNode(TNode *root);
109     void Transplant(TNode *a, TNode *b);
110     void FixAfterInsert(TNode *node);
111     void FixAfterErase(TNode *node);
112     void RightRotate(TNode *node);
113     void LeftRotate(TNode *node);
114     void RecursiveDestroy(TNode *node);
115 };
116
117
118 TNode::TNode(std::string key, uint64_t value)
119 {
120     this->key = key;
121     this->value = value;
122     this->colour = TColour::red;
123 }
124
125 TRBTree::TRBTree()
126 {
127     nil = new TNode();
128     root = nil;
129 }
130
131 TRBTree::~TRBTree()
132 {
133     RecursiveDestroy(root);
134     delete nil;
135 }
136
137 void TRBTree::RecursiveDestroy(TNode *node)
138 {
139     if (node != nil)
140     {
141         RecursiveDestroy(node->left);
142         RecursiveDestroy(node->right);
143         delete node;
144     }
145 }
```

```
146
147 TNode *TRBTree::GetNil()
148 {
149     return nil;
150 }
151
152 TNode *TRBTree::FindMinNode(TNode *root)
153 {
154     TNode *currentNode = root;
155     while (currentNode->left != nil)
156     {
157         currentNode = currentNode->left;
158     }
159     return currentNode;
160 }
161
162 TNode *TRBTree::FindNode(std::string key)
163 {
164     TNode *currentNode = this->root;
165     while (currentNode->key != key)
166     {
167         if (currentNode == nil)
168         {
169             break;
170         }
171         currentNode = (key < currentNode->key) ? currentNode->left : currentNode->right
                ;
172     }
173     return currentNode;
174 }
175
176 void TRBTree::LeftRotate(TNode *node)
177 {
178     TNode *tmp = node->right; // set tmp
179     node->right = tmp->left; // turn tmp's left subtree into node's right subtree
180     if (tmp->left != nil)
181     {
182         tmp->left->parent = node;
183     }
184     tmp->parent = node->parent; // link node's parent to tmp
185     if (node->parent == nil)
186     {
187         this->root = tmp;
188     }
189     else if (node == node->parent->left)
190     {
191         node->parent->left = tmp;
192     }
193     else
```

```
194     {
195         node->parent->right = tmp;
196     }
197     tmp->left = node; // put node on tmp's left
198     node->parent = tmp;
199 }
200 
201 void TRBTree::RightRotate(TNode *node)
202 {
203     TNode *tmp = node->left; // set tmp
204     node->left = tmp->right; // turn tmp's right subtree into node's left subtree
205     if (tmp->right != nil)
206     {
207         tmp->right->parent = node;
208     }
209     tmp->parent = node->parent; // link node's parent to tmp
210     if (node->parent == nil)
211     {
212         this->root = tmp;
213     }
214     else if (node == node->parent->left)
215     {
216         node->parent->left = tmp;
217     }
218     else
219     {
220         node->parent->right = tmp;
221     }
222     tmp->right = node; // put node on tmp's right
223     node->parent = tmp;
224 }
225 
226 void TRBTree::FixAfterInsert(TNode *node)
227 {
228     TNode *uncle;
229     while (node->parent->colour == TColour::red)
230     {
231         if (node->parent == node->parent->parent->left)
232         {
233             uncle = node->parent->parent->right;
234             if (uncle->colour == TColour::red)
235             {
236                 node->parent->colour = TColour::black; // Case 1
237                 uncle->colour = TColour::black; // Case 1
238                 node->parent->parent->colour = TColour::red; // Case 1
239                 node = node->parent->parent; // Case 1
240             }
241             else
242             {
```

```
243                 if (node == node->parent->right)
244                 {
245                     node = node->parent; // Case 2
246                     this->LeftRotate(node); // Case 2
247                 }
248                 node->parent->colour = TColour::black; // Case 3
249                 node->parent->parent->colour = TColour::red; // Case 3
250                 this->RightRotate(node->parent->parent); // Case 3
251             }
252         }
253         else
254         {
255             uncle = node->parent->parent->left;
256             if (uncle->colour == TColour::red)
257             {
258                 node->parent->colour = TColour::black; // Case 1
259                 uncle->colour = TColour::black; // Case 1
260                 node->parent->parent->colour = TColour::red; // Case 1
261                 node = node->parent->parent; // Case 1
262             }
263             else
264             {
265                 if (node == node->parent->left)
266                 {
267                     node = node->parent; // Case 2
268                     this->RightRotate(node); // Case 2
269                 }
270                 node->parent->colour = TColour::black; // Case 3
271                 node->parent->parent->colour = TColour::red; // Case 3
272                 this->LeftRotate(node->parent->parent); // Case 3
273             }
274         }
275     }
276     this->root->colour = TColour::black; // Case 0
277 }
278
279 bool TRBTree::Insert(std::string key, uint64_t value)
280 {
281     TNode *currentNode = this->root;
282     TNode *parentNode = nil;
283     while (currentNode != nil)
284     {
285         if (currentNode->key == key)
286         {
287             return false;
288         }
289         parentNode = currentNode;
290         currentNode = (key < currentNode->key) ? currentNode->left : currentNode->right
                ;
```

```
291     }
292     TNode *newNode = new TNode(key, value);
293     newNode->parent = parentNode;
294     newNode->left = nil;
295     newNode->right = nil;
296     if (parentNode == nil)
297     {
298         this->root = newNode;
299     }
300     else if (key < parentNode->key)
301     {
302         parentNode->left = newNode;
303     }
304     else
305     {
306         parentNode->right = newNode;
307     }
308     this->FixAfterInsert(newNode);
309     return true;
310 }
311 
312 void TRBTree::Transplant(TNode *a, TNode *b)
313 {
314     if (a->parent == nil)
315     {
316         this->root = b;
317     }
318     else if (a == a->parent->left)
319     {
320         a->parent->left = b;
321     }
322     else
323     {
324         a->parent->right = b;
325     }
326     b->parent = a->parent;
327 }
328 
329 void TRBTree::FixAfterErase(TNode *node)
330 {
331     while ((node != this->root) && (node->colour == TColour::black))
332     {
333         if (node == node->parent->left)
334         {
335             TNode *sibling = node->parent->right;
336             if (sibling->colour == TColour::red) // Case 1
337             {
338                 sibling->colour = TColour::black; // Case 1
339                 node->parent->colour = TColour::red; // Case 1
```

```
340                 this->LeftRotate(node->parent); // Case 1
341                 sibling = node->parent->right; // Case 1
342             }
343             if ((sibling->left->colour == TColour::black) && (sibling->right->colour ==
                     TColour::black)) // Case 2
344             {
345                 sibling->colour = TColour::red; // Case 2
346                 node = node->parent; // Case 2
347             }
348             else
349             {
350                 if (sibling->right->colour == TColour::black) // Case 3
351                 {
352                     sibling->left->colour = TColour::black; // Case 3
353                     sibling->colour = TColour::red; // Case 3
354                     this->RightRotate(sibling); // Case 3
355                     sibling = node->parent->right; // Case 3
356                 }
357                 sibling->colour = node->parent->colour; // Case 4
358                 node->parent->colour = TColour::black; // Case 4
359                 sibling->right->colour = TColour::black; // Case 4
360                 this->LeftRotate(node->parent); // Case 4
361                 node = this->root; // Case 4
362             }
363         }
364         else
365         {
366             TNode *sibling = node->parent->left;
367             if (sibling->colour == TColour::red) // Case 1
368             {
369                 sibling->colour = TColour::black; // Case 1
370                 node->parent->colour = TColour::red; // Case 1
371                 this->RightRotate(node->parent); // Case 1
372                 sibling = node->parent->left; // Case 1
373             }
374             if ((sibling->left->colour == TColour::black) && (sibling->right->colour ==
                     TColour::black)) // Case 2
375             {
376                 sibling->colour = TColour::red; // Case 2
377                 node = node->parent; // Case 2
378             }
379             else
380             {
381                 if (sibling->left->colour == TColour::black) // Case 3
382                 {
383                     sibling->right->colour = TColour::black; // Case 3
384                     sibling->colour = TColour::red; // Case 3
385                     this->LeftRotate(sibling); // Case 3
386                     sibling = node->parent->left; // Case 3
```

```
387                 }
388                 sibling->colour = node->parent->colour; // Case 4
389                 node->parent->colour = TColour::black; // Case 4
390                 sibling->left->colour = TColour::black; // Case 4
391                 this->RightRotate(node->parent); // Case 4
392                 node = this->root; // Case 4
393             }
394         }
395     }
396     node->colour = TColour::black;
397 }
398 
399 bool TRBTree::Erase(std::string key)
400 {
401     TNode *deleteNode = this->FindNode(key);
402     TNode *tmp;
403     TColour originalColour = deleteNode->colour;
404     if (deleteNode == nil)
405     {
406         return false;
407     }
408     if (deleteNode->left == nil)
409     {
410         tmp = deleteNode->right; // Case 1
411         this->Transplant(deleteNode, deleteNode->right); // Case 1
412         delete deleteNode; // Case 1
413     }
414     else if (deleteNode->right == nil)
415     {
416         tmp = deleteNode->left; // Case 2
417         this->Transplant(deleteNode, deleteNode->left); // Case 2
418         delete deleteNode; // Case 2
419     }
420     else
421     {
422         TNode *rightMinimum = this->FindMinNode(deleteNode->right); // Case 3
423         originalColour = rightMinimum->colour; // Case 3
424         tmp = rightMinimum->right; // Case 3
425         if (rightMinimum->parent == deleteNode) // Case 3
426         {
427             tmp->parent = rightMinimum; // Case 3
428         }
429         else
430         {
431             this->Transplant(rightMinimum, rightMinimum->right); // Case 3
432             rightMinimum->right = deleteNode->right; // Case 3
433             rightMinimum->right->parent = rightMinimum; // Case 3
434         }
435         this->Transplant(deleteNode, rightMinimum); // Case 3
```

```
436         rightMinimum->left = deleteNode->left; // Case 3
437         rightMinimum->left->parent = rightMinimum; // Case 3
438         rightMinimum->colour = deleteNode->colour; // Case 3
439         delete deleteNode; // Case 3
440     }
441     if (originalColour == TColour::black)
442     {
443         this->FixAfterErase(tmp);
444     }
445     return true;
446 }
447 
448 void TRBTree::Save(std::ofstream& file)
449 {
450     if (root != nil)
451     {
452         TQueue<TNode *> level;
453         TQueue<TNode *> next_level;
454         level.PushBack(root);
455         while (level.Size() != 0)
456         {
457             next_level.Clear();
458             for (size_t i = 0; i < level.Size(); ++i)
459             {
460                 file << level[i]->key << " " << level[i]->value << "\n";
461                 if (level[i]->left != nil)
462                 {
463                     next_level.PushBack(level[i]->left);
464                 }
465                 if (level[i]->right != nil)
466                 {
467                     next_level.PushBack(level[i]->right);
468                 }
469             }
470             level.Clear();
471             for (size_t i = 0; i < next_level.Size(); ++i)
472             {
473                 level.PushBack(next_level[i]);
474             }
475         }
476     }
477 }
478 
479 void TRBTree::Load(std::ifstream& file)
480 {
481     this->RecursiveDestroy(this->root);
482     this->root = nil;
483     std::string fkey;
484     uint64_t fvalue;
```

```
485     while (file >> fkey >> fvalue)
486     {
487         this->Insert(fkey, fvalue);
488     }
489 }
490 
491 std::string mtolower(std::string str)
492 {
493     for (size_t i = 0; i < str.size(); ++i)
494     {
495         str[i] = tolower(str[i]);
496     }
497     return str;
498 }
499 
500 int main()
501 {
502     std::ios::sync_with_stdio(false);
503     std::cin.tie(0);
504     std::cout.tie(0);
505     std::string input;
506     TRBTree tree;
507     while (std::cin >> input)
508     {
509         if (input == "+")
510         {
511             uint64_t value;
512             std::string key;
513             std::cin >> key >> value;
514             if (tree.Insert(mtolower(key), value))
515             {
516                 std::cout << "OK\n";
517             }
518             else
519             {
520                 std::cout << "Exist\n";
521             }
522         }
523         else if (input == "-")
524         {
525             std::string key;
526             std::cin >> key;
527             if (tree.Erase(mtolower(key)))
528             {
529                 std::cout << "OK\n";
530             }
531             else
532             {
533                 std::cout << "NoSuchWord\n";
```

```
534             }
535         }
536         else if (input == "!")
537         {
538             std::string action;
539             std::string path;
540             std::cin >> action >> path;
541             if (action == "Save")
542             {
543                 std::ofstream file;
544                 try
545                 {
546                     file.open(path);
547                 }
548                 catch (std::exception& ex)
549                 {
550                     std::cout << "ERROR: " << ex.what() << "\n";
551                 }
552                 tree.Save(file);
553             }
554             else if (action == "Load")
555             {
556                 std::ifstream file;
557                 try
558                 {
559                     file.open(path);
560                 }
561                 catch (std::exception& ex)
562                 {
563                     std::cout << "ERROR: " << ex.what() << "\n";
564                 }
565                 tree.Load(file);
566             }
567             std::cout << "OK\n";
568         }
569         else
570         {
571             TNode *FoundNode = tree.FindNode(mtolower(input));
572             if (FoundNode == tree.GetNil())
573             {
574                 std::cout << "NoSuchWord\n";
575             }
576             else
577             {
578                 std::cout << "OK: " << FoundNode->value << "\n";
579             }
580         }
581     }
582 }
```

3 Консоль

```
lexasy@MSI:$ cat test
+ a 1
+ A 2
+ aaa 18446744073709551615
aaa
A
-A
a
lexasy@MSI:$ make
g++ -pg RBtree.cpp main.cpp -o solution
lexasy@MSI:$ ./solution <test >/dev/null
lexasy@MSI:$ ls | grep gmon
gmon.out
lexasy@MSI:$ gprof ./solution <test >tmp
lexasy@MSI:$ valgrind ./solution <test >/dev/null
==276184== Memcheck,a memory error detector
==276184== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==276184== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==276184== Command: ./solution
==276184==
==276184==
==276184== Process terminating with default action of signal 27 (SIGPROF)
==276184==    at 0x4BD0A1A: __open_nocancel (open64_nocancel.c:39)
==276184==    by 0x4BDF56F: write_gmon (gmon.c:370)
==276184==    by 0x4BDFDDE: _mcleanup (gmon.c:444)
==276184==    by 0x4AFCA55: __cxa_finalize (cxa_finalize.c:83)
==276184==    by 0x10A626: ??? (in /home/lexasy/Desktop/Prog/DA_labs/lab2/solution)
==276184==    by 0x400624D: _dl_fini (dl-fini.c:142)
==276184==    by 0x4AFC494: __run_exit_handlers (exit.c:113)
==276184==    by 0x4AFC60F: exit (exit.c:143)
==276184==    by 0x4AE0D96: (below main) (libc_start_call_main.h:74)
==276184==
==276184== HEAP SUMMARY:
==276184==     in use at exit: 227,136 bytes in 8 blocks
==276184==   total heap usage: 11 allocs,3 frees,227,352 bytes allocated
==276184==
==276184== LEAK SUMMARY:
==276184==    definitely lost: 0 bytes in 0 blocks
==276184==    indirectly lost: 0 bytes in 0 blocks
```

```
==276184==      possibly lost: 0 bytes in 0 blocks
==276184==    still reachable: 227,136 bytes in 8 blocks
==276184==         suppressed: 0 bytes in 0 blocks
==276184== Rerun with --leak-check=full to see details of leaked memory
==276184==
==276184== For lists of detected and suppressed errors,rerun with: -s
==276184== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Profiling timer expired
```

4 Анализ результатов

Сначала необходимо разобраться что мы сделали. С помощью make мы скомпилировали нашу программу с ключами -pg. Эти ключи нужны для утилиты gprof. Далее мы запустили программу и перенаправили ее вывод в /dev/null. После этого у нас появился файл gmon.out. Далее запустили программу еще раз, уже с помощью утилиты gprof и перенапрвили ее вывод в файл tmp, где будет лежать информация о времени работы методов. Далее мы запустили еще раз программу с помощью утилиты valgrind и перенаправили вывод программы в /dev/null, чтобы не засорять консоль. В стандартный вывод утилита вывела результаты работы.

Теперь проанализируем результаты работы утилит. Начнем с утилиты gprof.

В начале файла tmp есть подобная надпись Each sample counts as 0.01 seconds. no time accumulated. T.e. gprof не записал время работы методов. Нас такой результат не устраивает, поэтому попробуем запустить программу на большем количестве тестов.

Вывод утилиты gprof состоит из двух частей, информация о том сколько проработали по времени методы и граф вызовов методов. Проанализируем только время работы методов.

```
Flat profile:

Each sample counts as 0.01 seconds.
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
24.56      0.42     0.42                             _init
16.37      0.70     0.28  5000000     0.00     0.00  mtolower
14.04      0.94     0.24  2437674     0.00     0.00  TRBTree::FixAfterInsert
11.11      1.13     0.19        1   190.00   190.00  TRBTree::RecursiveDestroy
10.23      1.30     0.17 102991681     0.00     0.00  __gnu_cxx::__enable_if
9.65      1.47     0.17 102867028     0.00     0.00  bool std::operator<
7.60      1.60     0.13  2499492     0.00     0.00  TRBTree::Insert
4.09      1.67     0.07  2500508     0.00     0.00  TRBTree::FindNode
1.17      1.69     0.02   708985     0.00     0.00  TRBTree::LeftRotate
0.58      1.70     0.01  1125431     0.00     0.00  std::char_traits::compare
0.29      1.71     0.01  2437674     0.00     0.00  TNode::TNode(str,ulong)
0.29      1.71     0.01                             frame_dummy
0.00      1.71     0.00 10001016     0.00     0.00  bool std::operator==
0.00      1.71     0.00  2500508     0.00     0.00  TRBTree::GetNil
0.00      1.71     0.00  2437675     0.00     0.00  TNode::~TNode
```

```
0.00      1.71     0.00   709685     0.00     0.00  TRBTree::RightRotate
0.00      1.71     0.00        1     0.00     0.00  TNode::TNode()
0.00      1.71     0.00        1     0.00     0.00  TRBTree::TRBTree
0.00      1.71     0.00        1     0.00   190.00  TRBTree::~TRBTree
```

Примечание: Для наглядности были убраны некоторые незначащие методы и аргументы почти у всех методов.

Разберем что означает каждый столбец в табличке[1].

% time - процентное отношение времени, затраченного на исполнение данной функции, к общему времени работы программы.

cumulative seconds - общее количество секунд, затраченное на исполнение данной функции и функций выше по списку.

self seconds - количество секунд, затраченное на исполнение лишь данной функции.

calls - количество вызовов данной функции.

self ms/call - среднее количество миллисекунд, проведенное в данной функции после каждого из вызовов.

total ms/call - Среднее количество миллисекунд, проведенное в данной функции и ее подфункциях после каждого из вызовов.

name - имя данной функции.

Теперь проанализируем результат работы утилиты valgrind.

Вывод этой утилиты говорит, что после выполнения программы все еще остаются доступными 8 блоков памяти общего размера 227кб. Попробуем исправить получившуюся утечку.

После некоторых манипуляций в коде программы попробуем запустить ее еще раз с помощью утилиты valgrind.

```
lexasy@MSI:$ make
g++ -pg RBtree.cpp main.cpp -o solution
lexasy@MSI:$ valgrind ./solution <test >/dev/null
==282308== Memcheck,a memory error detector
==282308== Copyright (C) 2002-2017,and GNU GPL'd,by Julian Seward et al.
==282308== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==282308== Command: ./solution
==282308==
==282308==
==282308== HEAP SUMMARY:
==282308==     in use at exit: 0 bytes in 0 blocks
==282308==   total heap usage: 7 allocs,7 frees,112,480 bytes allocated
```

```
==282308==
==282308== All heap blocks were freed --no leaks are possible
==282308==
==282308== For lists of detected and suppressed errors,rerun with: -s
==282308== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Теперь после выполнения программы не осталось доступных блоков памяти, что означает отсутствие утечек. До этого утечка была связана с отключением синхронизации с stdio.

5 Выводы

Выполнив третью лабораторную работу по курсу «Дискретный анализ», я узнал о существовании полезной утилиты gprof, которую раньше никогда не использовал. Она позволяет замерять время работы каждой функции программы, что бывает полезным при оптимизации. Если программа работает медленно, то с помощью утилиты gprof можно узнать какие функции работают медленнее ожидаемо и все исправить. Также я воспользовался утилитой valgrind, которая также предоставляет возможности для профилирования, но в большенстве случаев ее используют для выявления утечек памяти. С valgind'ом я был знаком и до этого. Он помогает мне писать программы без утечек. Есть еще много утилит профилирования для программ на C++, которые позволяют максимально улучшить свою программу и не допустить критических ошибок во время ее написания.

Список литературы

[1] https://sourceware.org/binutils/docs/gprof/index.htmlTop (Документация по gprof) (дата обращения: 05.05.2024)