ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»**

Московский институт электроники и математики им. А.Н. Тихонова

Департамент прикладной математики

**Отчёт
по лабораторной работе №11
по курсу «Алгоритмизация и программирование»
Задание № 13**

ФИО студента	Номер группы	Дата
Кейер Александр Петрович	БПМ-231	6 июня 2024 г.

Москва, 2023

1 Задание (вариант № 13)

Лабораторная работа 11

1. Реализовать поиск с помощью бинарного дерева поиска для объектов, описанных в варианте ЛР8. Ключом для поиска является первое поле структуры, т.е. если первое поле это, например, ФИО, то по поисковому запросу (строке) “Иванов Иван Иванович” на выходе необходимо получить все объекты с заданным ФИО и любыми значениями других полей.

2. Реализовать хеш-таблицу для поиска объектов (ключ поиска аналогичен п.1) и метод цепочек для разрешения коллизий на основе односвязного списка. В качестве алгоритма хэширования может быть использован любой известный метод.

3. Провести поиск обоими способами на наборах данных (количестве объектов) следующих размеров: 1000, 5000, 10000, 20000, 50000, 100000, 1000000. Засечь (программно) время поиска и по полученным точкам построить графики зависимости времени поиска от размерности массива для каждого из способов на одной оси координат. Полученные графики включить в отчет к работе.

2 Структура заголовочного файла

```c
struct Footballer {
  char* fullName;
  char* clubName;
  char* role;
  int age;
  int numberOfGames;
  int numberOfGoals;
};

typedef struct Footballer Footballer;

```

3 Решение

```c
#include <stdio.h> // Input/output library.
#include <stdlib.h> // Memory allocation.
#include <time.h> // Time library.
#include <assert.h> // Assertion library.
#include <string.h> // String functions library.

#include "football.h" // Footballer info.

#define TIME 1000000000

// Function printing horizontal line.
void printHr(int length) {
  for (int j = 0; j < length; j++) {
    printf("- ");
  }
  printf("\n");
}

// Function printing table header.
void printTableHeader() {
  printHr(65);
  printf("%4s|%30s|%30s|%30s|%10s|%10s|%10s|\n", "ID", "
FULL NAME", "CLUB NAME", "ROLE", "AGE", "GAMES", "GOALS");
  printHr(65);
}

// Function printing footballer.
```

```
27      void printFootballer(Footballer* footballer) {
28        printf("%4s|%30s|%30s|%30s|%10d|%10d|%10d|\n", "?",
    footballer->fullName, footballer->clubName, footballer->
    role, footballer->age, footballer->numberOfGames,
    footballer->numberOfGoals);
29        printHr(65);
30      }
31
32      // Function printing footballers.
33      void printFootballersArray(Footballer* footballers, int
    length) {
34        if (footballers == NULL) {
35          printf("Incorrect array.\n");
36          return;
37        }
38
39        printTableHeader();
40
41        for (int i = 0; i < length; i++) {
42          printFootballer(footballers + i);
43        }
44      }
45
46      // Function generating string.
47      char* generateString(int length, int countOfUsedSymbols)
    {
48        assert(countOfUsedSymbols <= 26);
49
50        char* out = (char*)malloc(sizeof(char) * (length + 1));
51        char alphabet[26] = {
52          'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'
    , 'l', 'm',
53          'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x'
    , 'y', 'z'
54        };
55
56        for (int i = 0; i < length; i++) {
57          out[i] = alphabet[rand() % countOfUsedSymbols];
58        }
59
60        out[length] = '\0';
61
62        return out;
63      }
64
```

```
65      // Function generating footballer.
66      Footballer generateFootballer(int rand1, int rand2) {
67        Footballer out = {
68          .fullName=generateString(rand1 % 26, rand2 % 26),
69          .clubName=generateString(10, 26),
70          .role=generateString(10, 26),
71          .age=(rand() % 100),
72          .numberOfGames=(rand() % 100),
73          .numberOfGoals=(rand() % 100),
74        };
75
76        return out;
77      }
78
79      // Function generating footballer array.
80      Footballer* generateFootballersArray(int length, int
    rand1, int rand2) {
81        Footballer* arr = (Footballer*)malloc(sizeof(Footballer
    ) * length);
82
83        for (int i = 0; i < length; i++) {
84          arr[i] = generateFootballer(rand1, rand2);
85        }
86
87        // printf("Successfully generated %d footballers array
    .\n", length);
88        return arr;
89      }
90
91      // Function comparing footballers.
92      int compareFootballers(Footballer* footballer1,
    Footballer* footballer2, int direction) {
93        return strcmp(footballer1->fullName, footballer2->
    fullName);
94      }
95
96      struct Node {
97        Footballer* footballer;
98        struct Node* left;
99        struct Node* right;
100     };
101     typedef struct Node Node;
102
103     Node* createNode(Footballer* footballer) {
104       Node* node = (Node*)malloc(sizeof(Node));
```

```
105
106       node->footballer = footballer;
107       node->left = node->right = NULL;
108
109       return node;
110     }
111
112     Node* insertNode(Node* root, Node* insertionNode) {
113       if (root == NULL) {
114         return insertionNode;
115       }
116
117       if (compareFootballers(insertionNode->footballer, root
    ->footballer, 1) >= 0) {
118         root->right = insertNode(root->right, insertionNode);
119       } else {
120         root->left = insertNode(root->left, insertionNode);
121       }
122
123       return root;
124     }
125
126     void insertNodeByFootballer(Node* root, Footballer*
    footballer) {
127       insertNode(root, createNode(footballer));
128     }
129
130     Node* createTreeFromFootballersArray(Footballer*
    footballers, int footballersCount) {
131       Node* root = createNode(&footballers[0]);
132
133       for (int i = 1; i < footballersCount; i++) {
134         insertNodeByFootballer(root, footballers + i);
135       }
136
137       return root;
138     }
139
140     void printFootabllersBinarySearchTree(Node* root) {
141       if (root == NULL) {
142         return;
143       }
144
145       printFootabllersBinarySearchTree(root->left);
146       printFootballer(root->footballer);
```

```
147       printFootabllersBinarySearchTree(root->right);
148     }
149
150     void printNodesArray(Node** nodes, int nodesCount) {
151       if (nodesCount == 0) {
152         printf("Footballers with this name were not found.\n"
    );
153       }
154
155       for (int i = 0; i < nodesCount; i++) {
156         printFootballer(nodes[i]->footballer);
157       };
158     }
159
160     Node* findNodeInBinarySearchTreeByFootballerFullName(Node
    * root, char* fullName) {
161       if (root == NULL || strcmp(root->footballer->fullName,
    fullName) == 0) {
162         return root;
163       }
164
165       if (strcmp(root->footballer->fullName, fullName) > 0) {
166         return findNodeInBinarySearchTreeByFootballerFullName
    (root->left, fullName);
167       }
168
169       return findNodeInBinarySearchTreeByFootballerFullName(
    root->right, fullName);
170     }
171
172     int findAllNodesInBinarySearchTreeByFootballerFullName(
    Node* root, char* fullName, Node** nodes) {
173       Node* curNode =
    findNodeInBinarySearchTreeByFootballerFullName(root,
    fullName);
174
175       int nodesCount = 0;
176
177       if (curNode == NULL) {
178         nodes = NULL;
179
180         return nodesCount;
181       }
182
183       nodes[0] = curNode;
```

```
184       nodesCount++;
185
186       Node* nextNode = curNode->right;
187
188       while (nextNode != NULL && strcmp(curNode->footballer->
    fullName, nextNode->footballer->fullName) == 0) {
189         nodes[nodesCount] = nextNode;
190         nextNode = nextNode->right;
191         nodesCount++;
192       }
193
194       return nodesCount;
195     }
196
197     // =================
198     // Hash table dope.
199     // Hash table crazy.
200     // Hash table here.
201     // =================
202
203     struct ListItem {
204       Footballer* footballer;
205       struct ListItem* next;
206     };
207     typedef struct ListItem ListItem;
208
209     ListItem* createListItem(Footballer* footballer) {
210       ListItem* listItem = (ListItem*)malloc(sizeof(ListItem)
    );
211
212       listItem->footballer = footballer;
213       listItem->next = NULL;
214
215       return listItem;
216     }
217
218     struct HashTable {
219       int size;
220       ListItem** table;
221     };
222     typedef struct HashTable HashTable;
223
224     HashTable* createHashTable(int size) {
225       HashTable* hashTable = (HashTable*)malloc(sizeof(
    HashTable));
```

```
226       ListItem** table = (ListItem**)malloc(sizeof(ListItem*)
    * size);
227
228       hashTable->table = table;
229       hashTable->size = size;
230
231       for (int i = 0; i < size; i++) {
232         table[i] = NULL;
233       }
234
235       return hashTable;
236     }
237
238     int hashFunction(HashTable* hashTable, char* fullName) {
239       int hash = 0;
240
241       while (*fullName != 0) {
242         hash = (hash + *fullName++) % hashTable->size;
243       }
244
245       return hash;
246     }
247
248     void insertFootballerIntoHashTable(HashTable* hashTable,
    Footballer* footballer) {
249       int hash = hashFunction(hashTable, footballer->fullName
    );
250
251       ListItem* listItem = createListItem(footballer);
252
253       listItem->next = hashTable->table[hash];
254       hashTable->table[hash] = listItem;
255     }
256
257     HashTable* createHashTableFromFootballersArray(Footballer
    * footballers, int footballersCount) {
258       HashTable* hashTable = createHashTable(footballersCount
    );
259
260       for (int i = 0; i < footballersCount; i++) {
261         insertFootballerIntoHashTable(hashTable, footballers
    + i);
262       }
263
264       return hashTable;
```

```
265     }
266
267     int findAllListItemsInHashTableByFootballerFullName(
    HashTable* hashTable, char* fullName, ListItem** listItems
    ) {
268       int hash = hashFunction(hashTable, fullName);
269
270       int listItemsCount = 0;
271
272       ListItem* listItem = hashTable->table[hash];
273
274       while (listItem != NULL) {
275         if (strcmp(listItem->footballer->fullName, fullName)
    == 0) {
276           listItems[listItemsCount] = listItem;
277           listItemsCount++;
278         }
279
280         listItem = listItem->next;
281       }
282
283       return listItemsCount;
284     }
285
286     void printFootballersHashTable(HashTable* hashTable) {
287       for (int hash = 0; hash < hashTable->size; hash++) {
288         if (hashTable->table[hash] == NULL) {
289           continue;
290         }
291
292         ListItem* listItem = hashTable->table[hash];
293
294         while (listItem != NULL) {
295           printFootballer(listItem->footballer);
296
297           listItem = listItem->next;
298         }
299       }
300     }
301
302     void printFootballersListItems(ListItem** listItems, int
    listItemsCount) {
303       if (listItemsCount == 0) {
304         printf("Footballers with this name were not found.\n"
    );
```

```
305         return;
306       }
307
308       for (int i = 0; i < listItemsCount; i++) {
309         printFootballer(listItems[i]->footballer);
310       }
311     }
312
313     // Function to free memory allocated for binary search
    tree nodes.
314     void freeBinarySearchTreeNodes(Node* root) {
315       if (root == NULL) {
316         return;
317       }
318
319       freeBinarySearchTreeNodes(root->left);
320       freeBinarySearchTreeNodes(root->right);
321
322       // Free the node itself.
323       free(root);
324     }
325
326     // Function to free memory allocated for hash table list
    items.
327     void freeHashTableListItems(HashTable* hashTable) {
328       for (int i = 0; i < hashTable->size; i++) {
329         ListItem* listItem = hashTable->table[i];
330
331         while (listItem != NULL) {
332           ListItem* temp = listItem;
333           listItem = listItem->next;
334
335           // Free the list item itself.
336           free(temp);
337         }
338       }
339     }
340
341     int testsCounter = 1;
342
343     void test(int footballersCount, char* fullName, int rand1
    , int rand2, int* binaryTreeAccTime, int* hashTableAccTime
    , int printEveryTest) {
344       if (printEveryTest) {
345         printf("\n
```

```
    ================================================================
     Test %d
    ================================================================\
    n\n", testsCounter++);
346       printf("Name=\"%s\"\n", fullName);
347     }
348
349     // Prepare program and generate footballers array.
350
351     Footballer* footballers = generateFootballersArray(
    footballersCount, rand1, rand2);
352
353     // printf("\nPrint footballers array.\n");
354     // printFootballersArray(footballers, footballersCount)
    ;
355
356     // printf("\n==== Binary tree ====\n");
357
358     // Create binary search tree.
359
360     Node* root = createTreeFromFootballersArray(footballers
    , footballersCount);
361
362     // printf("\nPrint footballers binary tree.\n");
363     // printTableHeader();
364     // printFootablllersBinarySearchTree(root);
365
366     // Work with binary search tree.
367
368     Node** nodes = (Node**)malloc(sizeof(Node*) *
    footballersCount);
369
370     clock_t start = clock();
371
372     int nodesCount =
    findAllNodesInBinarySearchTreeByFootballerFullName(root,
    fullName, nodes);
373
374     clock_t stop = clock();
375
376     if (printEveryTest) {
377       printf("BinarySearchTree %4s%7d: %.00fns\n", "n=",
    footballersCount, (double)(stop - start) / CLOCKS_PER_SEC
    * TIME);
378     }
```

```
379
380       *binaryTreeAccTime += (double)(stop - start) /
    CLOCKS_PER_SEC * TIME;
381
382       // printf("\nAll nodes with footballer fullName = \"%s
    \".\n", fullName);
383       // printTableHeader();
384       // printNodesArray(nodes, nodesCount);
385
386       // printf("\n==== Hash table ====\n");
387
388       // Create hash table.
389
390       HashTable* hashTable =
    createHashTableFromFootballersArray(footballers,
    footballersCount);
391
392       // printf("\nPrint footballers hash table.\n");
393       // printTableHeader();
394       // printFootballersHashTable(hashTable);
395
396       // Work with hash table.
397
398       ListItem** listItems = (ListItem**)malloc(sizeof(
    ListItem*) * footballersCount);
399
400       start = clock();
401
402       int listItemsCount =
    findAllListItemsInHashTableByFootballerFullName(hashTable,
     fullName, listItems);
403
404       stop = clock();
405
406       if (printEveryTest) {
407         printf("HashTable %11s%7d: %.00fns\n", "n=",
    footballersCount, (double)(stop - start) / CLOCKS_PER_SEC
    * TIME);
408       }
409
410       *hashTableAccTime += (double)(stop - start) /
    CLOCKS_PER_SEC * TIME;
411
412       // printf("\nAll list items with footballer fullName =
    \"%s\".\n", fullName);
```

```
413       // printTableHeader();
414       // printFootballersListItems(listItems, listItemsCount)
      ;
415
416       // End program and free allocated memory.
417
418       freeBinarySearchTreeNodes(root);
419       freeHashTableListItems(hashTable);
420
421       free(nodes);
422       free(listItems);
423
424       // Free the memory allocated for the footballers array
      and the strings inside it.
425       for (int i = 0; i < footballersCount; i++) {
426         free(footballers[i].fullName);
427         free(footballers[i].clubName);
428         free(footballers[i].role);
429       }
430
431       free(footballers);
432     }
433
434     void runTests(int footballersCount, int testsCount, int
      printEveryTest) {
435       srand(time(NULL)); // Init first random number.
436
437       int rand1 = rand() % 26;
438       int rand2 = rand() % 26;
439
440       int binaryTreeAccTime = 0;
441       int hashTableAccTime = 0;
442
443       for (int i = 0; i < testsCount; i++) {
444         test(footballersCount, generateString(rand1, rand2),
      rand1, rand2, &binaryTreeAccTime, &hashTableAccTime,
      printEveryTest);
445       }
446
447       printf("\n
      ============================================================
       Average from %d tests
      ============================================================\
      n\n", testsCount);
448
```

```
449       printf("BinarySearchTree %4s%7d: %.00fns\n", "
      footballersCount=", footballersCount, (double)(
      binaryTreeAccTime / testsCount));
450       printf("HashTable %24s%7d: %.00fns\n", "
      footballersCount=", footballersCount, (double)(
      hashTableAccTime / testsCount));
451
452       printf("\n\n");
453     }
454
455     int main() {
456       printf("Lab 11. Keyer, BAM231.\n");
457
458       runTests(1000, 100, 0);
459       runTests(5000, 100, 0);
460       runTests(10000, 100, 0);
461       runTests(20000, 100, 0);
462       runTests(50000, 100, 0);
463       runTests(100000, 100, 0);
464       runTests(1000000, 100, 0);
465
466       return 0;
467     }
468
469
```

4 Тесты

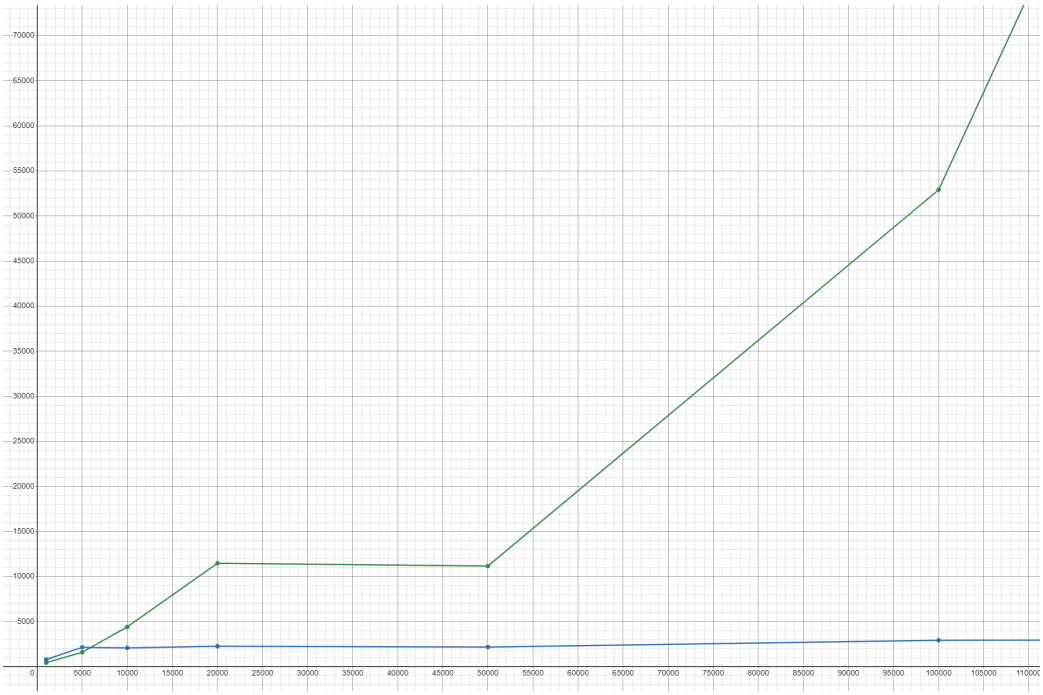
```
Lab 11. Keyer, BAM231.

============================================================ Average from 1000 tests ==========

BinarySearchTree footballersCount=   1000: 792ns
HashTable        footballersCount=   1000: 445ns



============================================================ Average from 1000 tests ==========

BinarySearchTree footballersCount=   5000: 2151ns
HashTable        footballersCount=   5000: 1607ns



============================================================ Average from 500 tests ===========

BinarySearchTree footballersCount=  10000: 2084ns
HashTable        footballersCount=  10000: 4416ns



============================================================ Average from 500 tests ===========

BinarySearchTree footballersCount=  20000: 2268ns
HashTable        footballersCount=  20000: 11474ns



============================================================ Average from 250 tests ===========

BinarySearchTree footballersCount=  50000: 2184ns
HashTable        footballersCount=  50000: 11164ns



============================================================ Average from 250 tests ===========

BinarySearchTree footballersCount= 100000: 2932ns
HashTable        footballersCount= 100000: 52908ns



============================================================ Average from 100 tests ===========

BinarySearchTree footballersCount=1000000: 4240ns
HashTable        footballersCount=1000000: 2006220ns
```

Для каждого количества футболистов было проведено свое количество тестов и взято среднее арифметическое в наносекундах, потраченных на поиск всех футболистов с определенным именем.

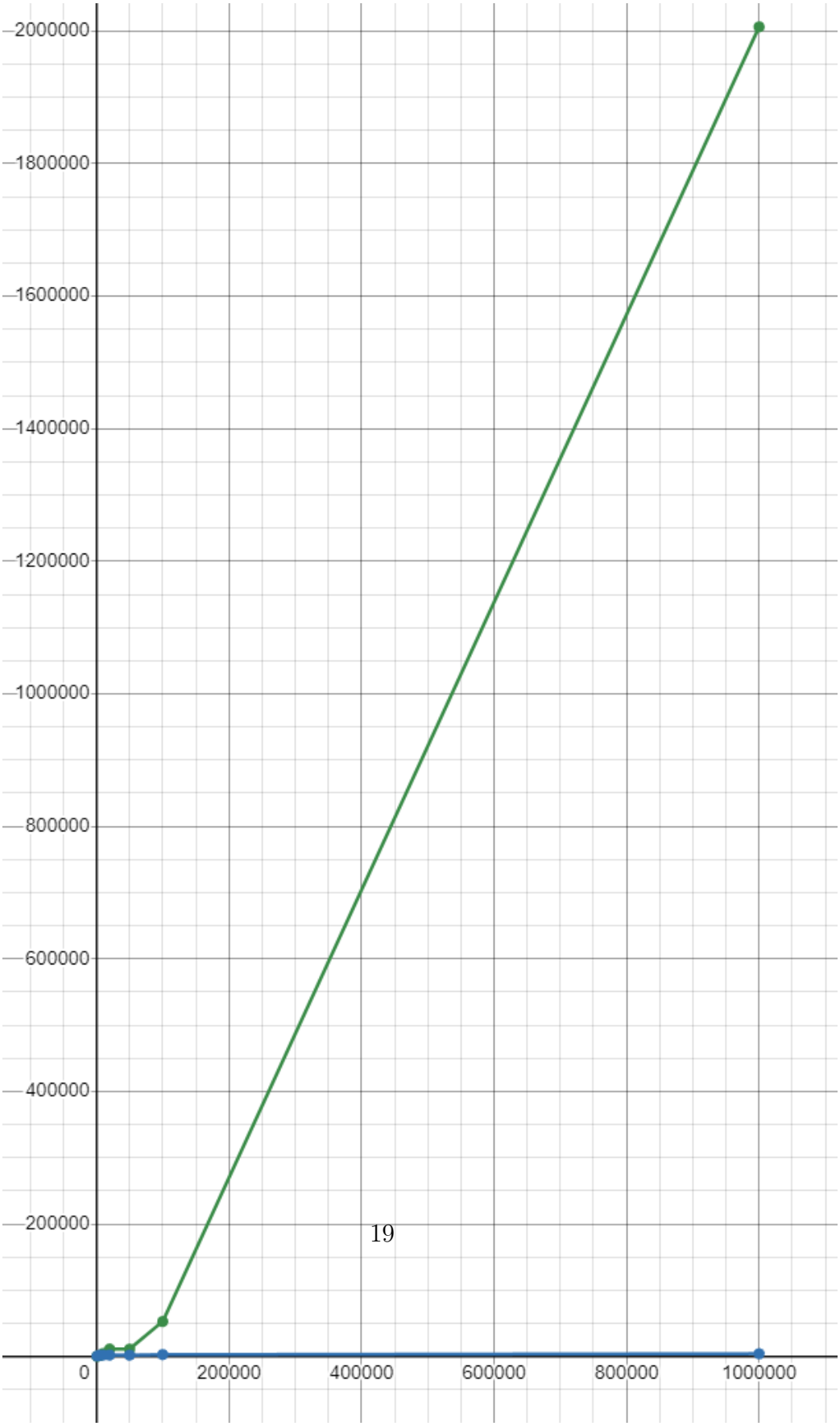
1. Имена в каждом тести генерировались автоматически: максимальное количество букв в имени 26, использовать можно было тоже только 26 латинских букв. Суммарно тесты покрывают каждую возможную длину имени и разное количество букв, из которых оно состоит

2. Забавно, но это замеры на Mac M3. Если взять Intel и Windows 11, но все тот же GCC, то там поиск по бинарному дереву всегда 0 наносекунд, а вот поиск по хеш-таблице оставляет желать лучшего.

5 Графики



У бинарного дерево, как и должно быть, идет нечто похожее на логарифм, а у хеш-таблицы на обычную линейную функцию.


2000000
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
200000
400000
600000
800000
1000000