ALGORİTMA ANALİZİ

Algoritma Analizi Final Projesi

Hashing

BİLAL KAHRAMAN

17011062

YÖNTEM

Cache boyutu kullanıcıdan alınmıştır ve LoadFactor ile en küçük asal sayı hesaplanarak hash tablo uzunluğu elde edilmiştir. Ardından kontrol edilmesi gerek 3 durum mevcuttur. Bunlar:

1. Aranan Elemanın Cache'de mevcut olması; Yapılması gereken hash tablosundan elemanın bilgilerini bulmak ve sıra değerini 0'a eşitlemek, ardından cache'de o elemana gelene kadar karşımıza çıkan tüm elemanların sıra değerlerini 1 arttırmak ve id değeri ile bulunan elemanı cache'in başına taşımak.

2. Eleman cache'de değil ve cache'de yer var; Yapılması gereken Hash tablosunda eleman için yeri bulmak ve sıra değerini 0 yapmak, ardından cache'de kullanılmayan yere gelene kadar karşımıza çıkan tüm elemanların sıra değerlerini 1 arttırmak ve boştaki yere bilgileri yüklemek ardından elemanıu cache'in başına taşımak

3. Eleman cache'de değil ve cache'de yer yok ise; Yapılması gereken cache'in sonundaki elemanı hem cache'den hem de hash tablosundan çıkarmak ve yeni elemanı yukarıdaki adımlarda ortak olduğu gibi yüklemektir.

```
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <stdbool.h>
7. #include <string.h>
8.
9. typedef struct hashNode
10.{
11.    char id[10];
12.    int order;
13.    bool is_cached;
14.} hashNode;
15.
16.typedef struct cacheNode
17.{
18.    char id[10];
19.    char name[10];
20.    char surname[10];
21.    char dateOfBirth[5];
22.    char address[10];
23.    struct cacheNode *next;
24.} cacheNode;
25.
26.int hornerHash(char *id);
27.int doubleHash(hashNode *, char *, int, bool *);
28.bool isPrime(int n);
29.int nextPrime(int n);
30.cacheNode *createCache(int size);
31.hashNode *createHash(int size);
```

```
32.
33.int main()
34.{
35.    //Load faktör
36.    double loadFactor = 0.60;
37.    //cacheSize cache listesinin toplam boyutunu tutar
38.    //cacheLen cache listesinin eleman sayısını tutar
39.    int cacheSize = 4, cacheLen = 0, i;
40.    printf("Enter the size of the cache: ");
41.    scanf("%d", &cacheSize);
42.    //Hash Tablosunun boyutu hesaplanır.
43.    int hashTableSize = nextPrime((int)(cacheSize / loadFactor));
44.    hashNode *hashTable = createHash(hashTableSize);
45.    cacheNode *cache = createCache(cacheSize);
46.    cacheNode *p, *q;
47.    bool exist, temp;
48.    char id[10], name[10], surname[10], dateOfBirth[5], address[10],
   filename[20] = "test.txt";
49.    FILE *fp;
50.    printf("Enter the name of the file: ");
51.    scanf("%s", filename);
52.    fp = fopen(filename, "r");
53.    /**
54.     * @brief Okunan dosyasnın sonuna kadar çalışır.
55.     *        Okunan girdini id'sine göre bir hash indisi hesaplanır.
56.     *        Eğer bu id cache var ise ekrana yazdırılır ve konumu cachein
   başına getirilir.
57.     *        Ondan önce olan elemanlar ise bir adım ileri taşınarak hash
   tablosundaki order değerleri bir arttırılır.
58.     *
59.     *        Eğer girilen değer cachede yok ve cachede yer var ise cache'e
   eklenir.
60.     *        Eğer cache dolu ise sondaki elemana gidilir ve bu elemanın
   bilgileri hash tablosundan silinir.Ardından eklenecek olan
61.     *        eleman için tekrardan hash değeri hesaplanır ve bu değerle hash
   tablosuna eklenir.
62.     *        Her cache'e eleman eklemesi yapıldığında eğer eleman en başta
   değilse, diğer elemanların hash tablosundaki
63.     *        order değerleri bir arttırılır ve bir adım taşınır.
64.     *        (Aslında birer adım taşınma işleminden kasıt, cache'in başına
   gidecek olan liste elemanının yerinden koparılıp başa taşınması işlemidir.)
65.     *
66.     */
67.    while (!feof(fp))
68.    {
```

```
69.         fscanf(fp, "%s %s %s %s %s", id, name, surname, dateOfBirth, address);
70.         int doubleHashIndex = doubleHash(hashTable, id, hashTableSize,
    &exist);
71.         //Eğer hash tablosunda bu id varsa doubleHashIndex hash tablosu
    sınırlarında bir sayı dönecektir. Eğer id yok ise random bir sayı dönecektir.
72.         //Görsel amaçla yazdırılmıştır.
73.         printf("Hash Table Index: %d\n", doubleHashIndex);
74.         //Eğer hash tablosunda bu id varsa, order bir değer dönecektir. Aksi
    halde random bir sayı dönecektir.
75.         printf("Cache Order: %d\n", hashTable[doubleHashIndex].order);
76.         if (exist)
77.         {
78.             i = 0;
79.             p = cache;
80.             // id var ise, id'ye varana kadar gelecek olan tüm nodeların order
    değerlerini bir arttırır.
81.             while (i < hashTable[doubleHashIndex].order)
82.             {
83.                 q = p;
84.                 hashTable[doubleHash(hashTable, q->id, hashTableSize,
    &temp)].order++;
85.                 p = p->next;
86.                 i++;
87.             }
88.             printf("Information about %s is in cache\n", id);
89.             printf("%s %s %s %s %s\n", p->id, p->name, p->surname,
    p->dateOfBirth, p->address);
90.             // Eğer id cacde ilk eleman değilse başa getirilir.
91.             if (p != cache)
92.             {
93.                 q->next = p->next;
94.                 p->next = cache;
95.                 cache = p;
96.                 hashTable[doubleHashIndex].order = 0;
97.             }
98.         }
99.         else
100.             {
101.                 if (cacheLen < cacheSize)
102.                 {
103.                     p = cache;
104.                     // Eğer cachede yer var ise boşta olan node id char
    dizisinin ilk elemanın kontrolü ile bulunur. Ona varana kadar gelecek
105.                     // olan tüm nodeların order değerlerini bir arttırır.
106.                     while (p->id[0] != '\0' || p->id[0] != '\000')
```

```
107.                     {
108.                         q = p;
109.                         hashTable[doubleHash(hashTable, p->id,
    hashTableSize, &temp)].order++;
110.                         p = p->next;
111.                     }
112.                     strcpy(p->id, id);
113.                     strcpy(p->name, name);
114.                     strcpy(p->surname, surname);
115.                     strcpy(p->dateOfBirth, dateOfBirth);
116.                     strcpy(p->address, address);
117.
118.                     if (p != cache)
119.                     {
120.                         q->next = p->next;
121.                         p->next = cache;
122.                         cache = p;
123.                     }
124.                     // id 'ye ait bilgiler hashtablosuna işlenir.
125.                     hashTable[doubleHashIndex].order = 0;
126.                     hashTable[doubleHashIndex].is_cached = true;
127.                     strcpy(hashTable[doubleHashIndex].id, id);
128.
129.                     cacheLen++;
130.                 }
131.                 else
132.                 {
133.                     p = cache;
134.                     // Eğer cache dolu ise sondaki elemana gidilir ve gidene
    kadar gelecek olan elemanların order değerlerini bir arttırır.
135.                     while (p->next != NULL)
136.                     {
137.                         q = p;
138.                         hashTable[doubleHash(hashTable, p->id,
    hashTableSize, &temp)].order++;
139.                         p = p->next;
140.                     }
141.                     // Sondaki elemanın bilgileri silinir. Ardından
    eklenecek olan eleman için tekrardan hash değeri hesaplanır ve bu değerle hash
    tablosuna eklenir.
142.                     hashTable[doubleHash(hashTable, p->id, hashTableSize,
    &temp)].is_cached = false;
143.                     int newHashIndex = doubleHash(hashTable, id,
    hashTableSize, &temp);
144.                     strcpy(p->id, id);
```

```
145.                     strcpy(p->name, name);
146.                     strcpy(p->surname, surname);
147.                     strcpy(p->dateOfBirth, dateOfBirth);
148.                     strcpy(p->address, address);
149.
150.                     if (p != cache)
151.                     {
152.                         q->next = NULL;
153.                         p->next = cache;
154.                         cache = p;
155.                     }
156.
157.                     hashTable[newHashIndex].order = 0;
158.                     hashTable[newHashIndex].is_cached = true;
159.                     strcpy(hashTable[newHashIndex].id, id);
160.                 }
161.             }
162.         }
163.         fclose(fp);
164.
165.         // hash tablosu ve cache hafızadan silinir.
166.         free(hashTable);
167.         while (cache)
168.         {
169.             cacheNode *temp = cache;
170.             cache = cache->next;
171.             free(temp);
172.         }
173.         printf("finished\n");
174.         return 0;
175.     }
176.
177.     /**
178.      * @brief çift hash ile hash tablosunun indisini bulmaya yarayan
    fonksiyon.
179.      *        Eğer girilen id'ye ait bir kayıt varsa, exist değişkeni true
    olur ve indis döner
180.      *        Eğer boş bir alan bulunursa, exist değişkeni false olur ve bu
    kayıtın indisi döndürür.
181.      *        Eğer kayıt yoksa, exist false olur ve  -1 döndürür.
182.      *
183.      * @param hashTable
184.      * @param id
185.      * @param hashTableSize
186.      * @param exist
```

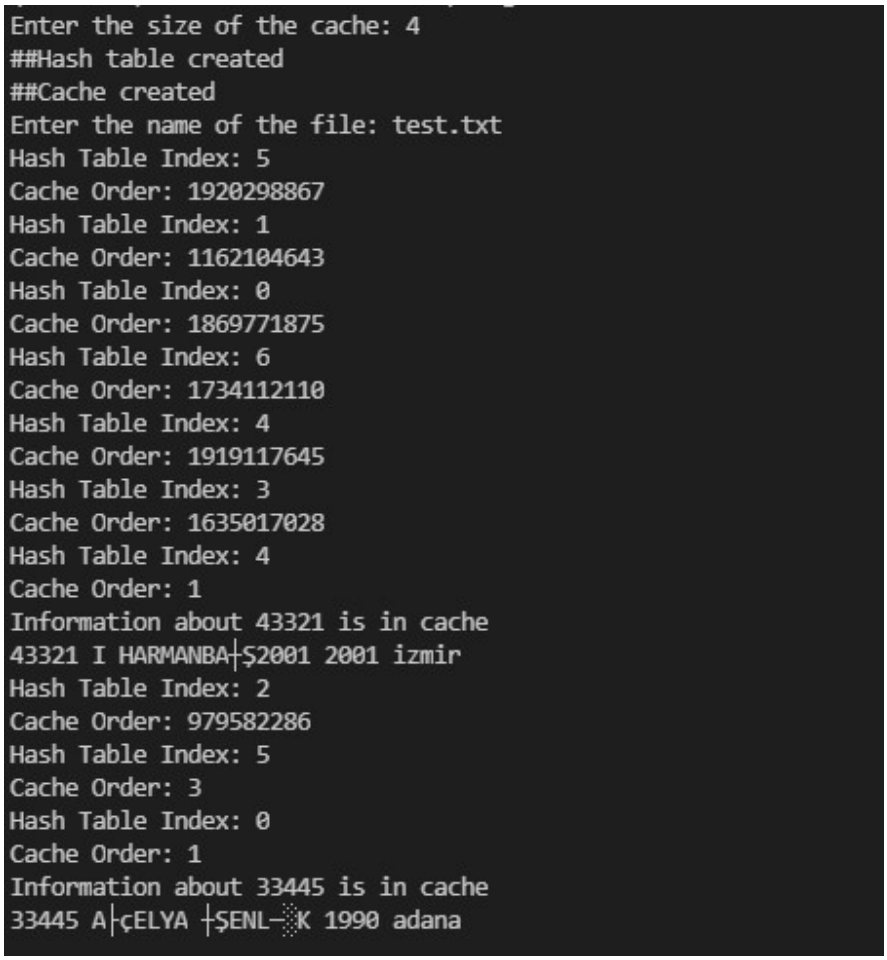
```
187.       * @return int
188.       */
189.      int doubleHash(hashNode *hashTable, char *id, int hashTableSize, bool
   *exist)
190.      {
191.          int i = 0;
192.          int hash = hornerHash(id);
193.          int index = ((hash % hashTableSize) + i * (1 + (hash % (hashTableSize
   - 1)))) % hashTableSize;
194.          while (strcmp(hashTable[index].id, id) != 0 &&
   hashTable[index].is_cached == true && i < hashTableSize)
195.          {
196.              i++;
197.              index = ((hash % hashTableSize) + i * (1 + (hash % (hashTableSize
   - 1)))) % hashTableSize;
198.          }
199.          if (hashTable[index].is_cached == false)
200.          {
201.              *exist = false;
202.              return index;
203.          }
204.          else if (strcmp(hashTable[index].id, id) == 0)
205.          {
206.              *exist = true;
207.              return index;
208.          }
209.          else
210.          {
211.              *exist = false;
212.              return -1;
213.          }
214.      }
215.
216.      /**
217.       * @brief hornerHash fonksiyonu
218.       *
219.       * @param id
220.       * @return int
221.       */
222.      int hornerHash(char *id)
223.      {
224.          int hash = 0;
225.          int i = 0;
226.          while (id[i] != '\0')
227.          {
```

```
228.             hash = (hash * 37) + id[i];
229.             i++;
230.         }
231.         return hash;
232.     }
233.
234.     /**
235.      * @brief Girilen integer sayısının asal olup olmadığının kontrolü
   yapılır.
236.      *
237.      * @param n
238.      * @return true
239.      * @return false
240.      */
241.     bool isPrime(int n)
242.     {
243.         int i = 2;
244.         while (i < n && n % i != 0)
245.         {
246.             i++;
247.         }
248.         if (i == n)
249.         {
250.             return true;
251.         }
252.         return false;
253.     }
254.
255.     /**
256.      * @brief Girilen bir integer değerinden daha büyük en küçük asal sayıyı
   bulur.
257.      *
258.      * @param n
259.      * @return int
260.      */
261.     int nextPrime(int n)
262.     {
263.         while (!isPrime(n))
264.         {
265.             n++;
266.         }
267.         return n;
268.     }
269.
270.     /**
```

```
271.       * @brief Cache listesini oluşturur. Listede sırradaki elemanın boş olup
   olmadığının kontrolü için
272.       *        id char dizisinin ilk karakterine \0 ataması yapılır.
273.       *
274.       * @param size
275.       * @return cacheNode*
276.       */
277.      cacheNode *createCache(int size)
278.      {
279.          cacheNode *cache = (cacheNode *)malloc(sizeof(cacheNode));
280.          cacheNode *head = cache;
281.          int i = 0;
282.          for (i = 0; i < size - 1; i++)
283.          {
284.              cache->id[0] = '\0';
285.              cache->next = (cacheNode *)malloc(sizeof(cacheNode));
286.              cache = cache->next;
287.          }
288.          cache->next = NULL;
289.          cache->id[0] = '\0';
290.          printf("##Cache created\n");
291.          return head;
292.      }
293.
294.      /**
295.       * @brief Hash tablosunu oluşturur. Girilen boyutta bir dizi oluşturur.
296.       *
297.       * @param size
298.       * @return hashNode*
299.       */
300.      hashNode *createHash(int size)
301.      {
302.          hashNode *hashTable = (hashNode *)malloc(sizeof(hashNode) * size);
303.          int i = 0;
304.          for (i = 0; i < size; i++)
305.          {
306.              hashTable[i].is_cached = false;
307.          }
308.          printf("##Hash table created\n");
309.          return hashTable;
310.      }
311.
```

UYGULAMA

Test.txt dosyası kullanılarak elde edilmiş sonuçlar aşağıdaki gibidir.



```
Enter the size of the cache: 10
##Hash table created
##Cache created
Enter the name of the file: test.txt
Hash Table Index: 3
Cache Order: 1982353728
Hash Table Index: 11
Cache Order: 0
Hash Table Index: 15
Cache Order: 6952812
Hash Table Index: 13
Cache Order: 0
Hash Table Index: 8
Cache Order: 1982355152
Hash Table Index: 4
Cache Order: 0
Hash Table Index: 8
Cache Order: 1
Information about 43321 is in cache
43321 I HARMANBA┼Ş2001 2001 izmir
Hash Table Index: 14
Cache Order: 0
Hash Table Index: 3
Cache Order: 6
Information about 12345 is in cache
12345 ┼ŞABAN DEM─▒RHAN 1993 istanbul
Hash Table Index: 14
Cache Order: 1
Information about 33445 is in cache
33445 A├çELYA ┼ŞENL─▒K 1990 adana
```

```
Enter the size of the cache: 2
##Hash table created
##Cache created
Enter the name of the file: test.txt
Hash Table Index: 0
Cache Order: 1869771875
Hash Table Index: 2
Cache Order: 979582286
Hash Table Index: 1
Cache Order: 1162104643
Hash Table Index: 0
Cache Order: 1
Hash Table Index: -1
Cache Order: 314447507
Hash Table Index: -1
Cache Order: 314447507
Hash Table Index: 1
Cache Order: 1
Information about 43321 is in cache
43321 I HARMANBA┼Ş2001 2001 izmir
Hash Table Index: -1
Cache Order: 314447507
Hash Table Index: -1
Cache Order: 314447507
Hash Table Index: 0
Cache Order: 1
Information about 33445 is in cache
33445 A├ÇELYA ┼ŞENL─▒K 1990 adana
```

Karmaşıklıklar:

- Dosyadan okuma: O(n)
- Hash Tablosu erişme O(1)
- Cache Erişme max O(n) min O(1)