VERİ YAPILARI VE ALGORİTMALAR

- ÖDEV 2 -

GRUP 1

İrem ATILGAN

17061036

06.04.2020

GİRİŞ

Ödev main fonksiyonu da dahil olmak üzere 14 fonksiyondan oluşmaktadır. Fonksiyonların işlevlerini kısaca özetlemek gerekirse:

scanConsole : Kullanıcı harfleri konsol ekranından girmek isterse kullanılır.

scanFile : Kullanıcı harfleri .txt uzantılı bir dosyadan okutmak isterse kullanılır.

printList : Oluşan linkli listeyi yazdırmak amacıyla kullanılır.

createNode : Yeni düğüm oluşturmak için kullanılır.

searchWord : Linkli listeye harf eklemeden önce, harfin listede olup olmadığını kontrol etmek için kullanılır. Listedeyse bulunduğu düğümün frekansı bir arttırılır, değilse yeni düğüm olarak **createNode** fonksiyonu ile eklenir.

addLinkedList : Linkli liste oluşturma fonksiyonudur. Bu fonksiyon içinde **searchWord** ve **createNode** fonksiyonu çağrılır.

insertionSort : Linkli listeyi oluşturduktan sonra oluşan listedeki düğümleri frekanslarına göre sıralayan fonksiyondur.

createHuffmanTree : Sıralanan linkli listeden Huffman Ağacını oluşturan fonksiyondur. **createNode** ve **insertionSort** fonksiyonlarını kullanır.

findHeight : Oluşan Huffman Ağacının yüksekliğini bulan fonksiyondur.

createQueue : Kuyruk oluşturma fonksiyonudur.

enqueue : Kuyruğa eleman ekleme fonksiyonudur.

dequeue : Kuyrukta sıradaki elemanı çıkarma fonksiyonudur.

printLevelOrder : Huffman Ağacının seviye bazlı olarak yazdırılmasını sağlayan fonksiyondur. Bunun için kuyruk yapısını kullanır. **createQueue**, **enqueue** ve **dequeue** fonksiyonlarını kullanır.

PROGRAM

scanConsole:

```c
//Cümle konsoldan okunmak istenirse
void scanConsole(char sentence[]){

    int i = 0;
    char c;

    //Daha önce kullanıcıdan input karakter alındığından, arkasından gelen '\n' karakterini
    //alması için getchar() yeniden kullanılır
    getchar();

    printf("Cumleniz : ");

    while((i < MAX) && (c = getchar())!= '\n')
    {
        sentence[i] = c;
        i++;
    }
    sentence[i] = '\0'; //Cümle sonu olduğunun bilgisayar tarafından anlaşılabilmesi için
                        //dizinin sonuna '\0' yerleştirilir

}
```

scanFile:

```c
//Cümle dosyadan okunmak istenirse
void scanFile(char sentence[]){

    char fname[20],tmp;
    int i = 0;

    //Kullanıcıdan dosyanın adı alınır
    printf("Dosya adini giriniz (txt) : "); scanf("%s",&fname);
    strcat(fname,".txt");
    FILE* fp = fopen(fname,"r");


    if(fp != NULL){

        while((sentence[i] = fgetc(fp)) != EOF){
            i++;
        }
        sentence[i] = '\0';

    }
    else{
        printf("Dosya Acilamadi!..");
        exit(1);
    }

    fclose(fp);

```

printList & createNode:

```
61
62  //Linked list'i yazdırma fonksiyonu
63  void printList(NODE* root){
64
65      NODE* current;
66
67      for(current = root; current->next != NULL; current = current->next)
68      {
69
70          printf("[  %c  ( %d )  ] --> ",current->word, current->frequency);
71
72      }
73      printf("[  %c  ( %d )  ]",current->word, current->frequency);
74  }
75
76  NODE* createNode(char key){
77
78      NODE* node = (NODE*)malloc(sizeof(NODE));        //node için hafızada yer ayrılır
79      node->word = key;                                //yerleştirmek istediğimiz harf word'e atanır
80      node->frequency = 1;
81      node->right = node->left = node->next = NULL;
82      return node;
83  }
```

main:

```
356 int main(){
357
358     char input[MAX];
359     int key, height;
360
361     printf("\nCumlenizi konsoldan yazmak isterseniz 1, dosyadan yazdirmak isterseniz 0 giriniz : ");
362     scanf("%d",&key);
363
364
365     if(key)
366         scanConsole(input);
367     else{
368         scanFile(input);
369         printf("\n\nCumle : \"%s\"",input);
370     }
371     printf("\n");
372
373
374
375     NODE* root = addLinkedList(input);
376
377     printf("\nLinkli Liste :\n");
378     printList(root);
379
380     NODE* newRoot = insertionSort(root);
381
382     printf("\n\nLinkli Liste (Insertion Sort) :\n");
383     printList(newRoot);
384
385     NODE* newRoot2 = createHuffmanTree(newRoot);
386
387
388
389     return 0;
390 }
```

addLinkedList:

```
107 NODE* addLinkedList(char input[]){
108
109     NODE *root = createNode(input[0]);      //root node oluşturulur
110     int i = 1;
111
112     while(input[i] != '\0'){
113
114         root = searchWord(root,input[i]);   //Alınan input harf, linkli listede olup olmadığını kontrol
115                                             //etmek üzere searchWord fonksiyonuna gönderilir
116         i++;
117         if(input[i] == ' ')                 //Cümledeki boşlukların sayılmaması için boşluk görüldüğünde
118             i++;                            //bir sonraki karaktere geçilir
119
120     }
121
122     return root;
123 }
```

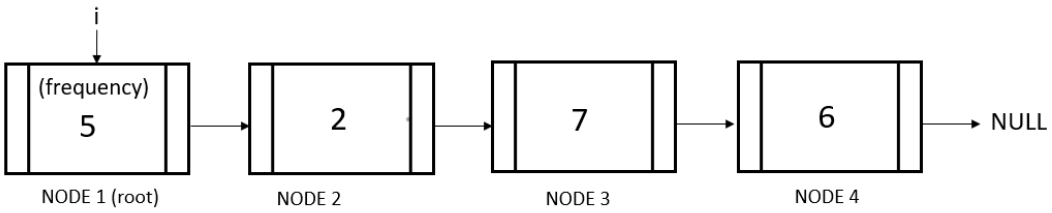
searchWord:

```
86  //Gelen harfin node'u oluşturulmadan önce linkli listede olup olmadığına bakılır
87  NODE* searchWord(NODE* root, char key){
88
89      NODE* current = root;
90
91
92      while(current->next != NULL && key != current->word ){
93          current = current->next;
94      }
95      //Eğer listenin sonuna geldiysek ve hala bulamadıysak
96      if(current->word != key)
97          current->next = createNode(key);    //Yeni node yaratılır
98      else
99          current->frequency++;               //Harf listede bulunduysa ilgili node'un frekansı arttırılır
100
101     return root;
102
103
104
105 }
```

insertionSort:

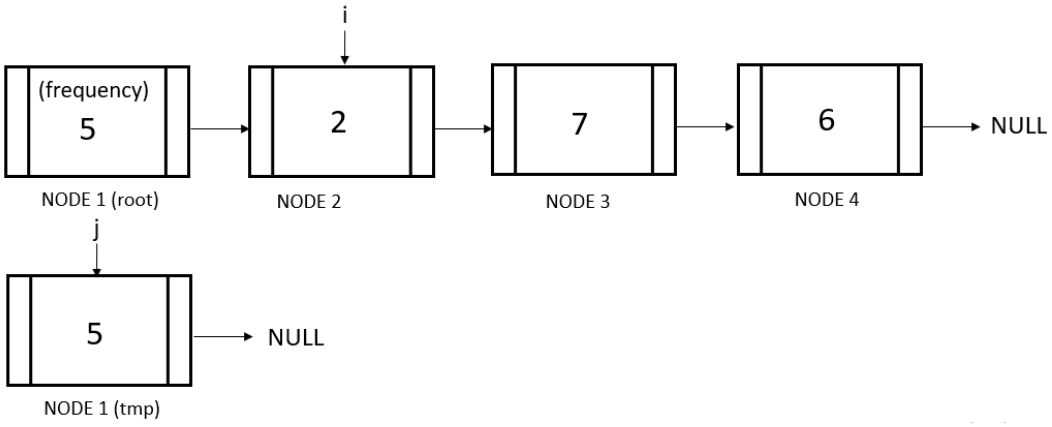
```
126 //Linkli liste oluşturulduktan sonra insertion sort ile sıralanır
127 NODE* insertionSort(NODE* root){
128
129     NODE* i = root;
130     NODE* j;
131
132     NODE* next;
133     NODE* tmp = NULL;
134
135     while(i != NULL){
136
137         next = i->next; //i node'u üzerinde değişiklik yapacağımız için gelecek node'unu
138                         //kaybetmemek için next NODE pointer'ında tutarız
139
140         if(tmp == NULL ||(tmp->frequency >= i->frequency)){ //tmp henüz bir node'u göstermiyorsa (başlangıç) veya gösterdiği node'un frekansı
141                                                             //bulunduğumuz node'un frekansından büyük ve eşitse (küçükten büyüğe sıralama yapıyoruz)
142             //İki node arasında swap işlemi yapılır
143             i->next = tmp;
144             tmp = i;
145         }
146         else{
147             //Eğer tmp'nin gösterdiği node'un frekansı, bulunduğumuz node'un frekansından küçükse
148             //Frekansı kendinden küçük bir node bulunana dek tüm liste gezilir
149
150             j = tmp;
151
152             while(j->next != NULL && (j->next->frequency < i->frequency)){
153                 j = j->next;
154             }
155             //İki node arasında swap işlemi yapılır
156             i->next = j->next;
157             j->next = i;
158
159         }
160
161         //Listede bir sonraki node'a geçilir
162         i = next;
163     }
164
165     root = tmp;
166
167
168
169     return root;
170
171 }
```

1

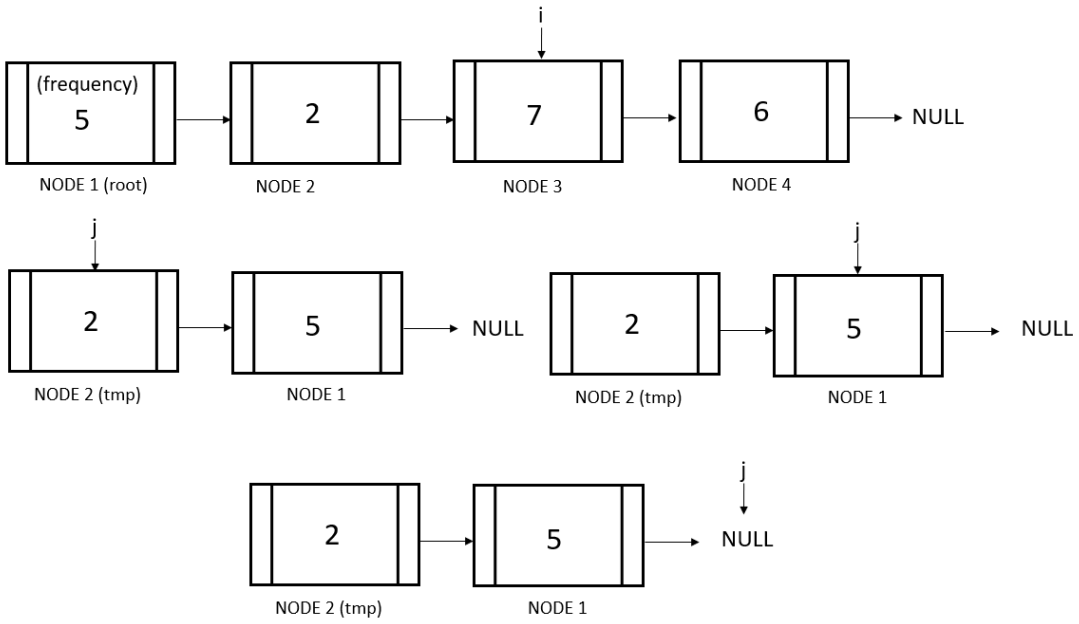

i
(frequency)
5
2
7
6
NULL
NODE 1 (root)
NODE 2
NODE 3
NODE 4

tmp → NULL

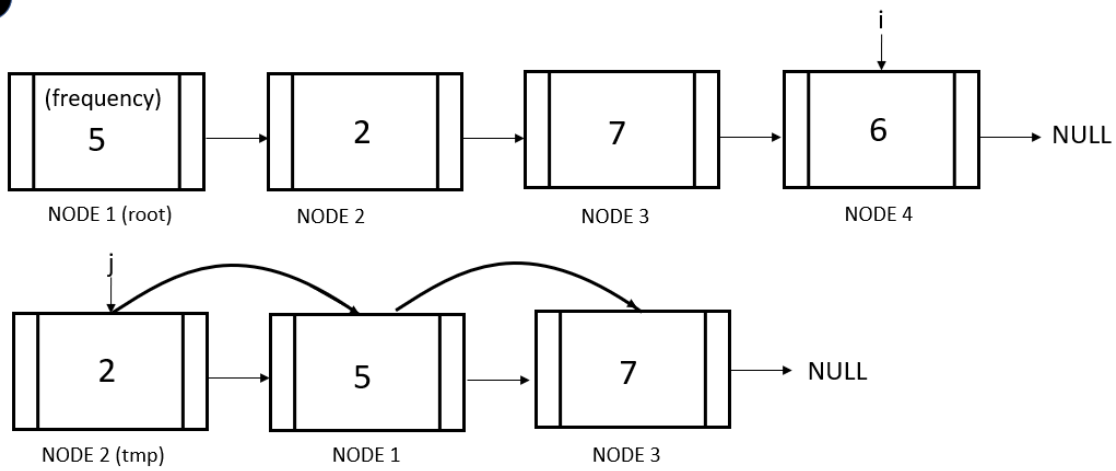
2


i
(frequency)
5
2
7
6
NULL
NODE 1 (root)
NODE 2
NODE 3
NODE 4
j
5
NULL
NODE 1 (tmp)

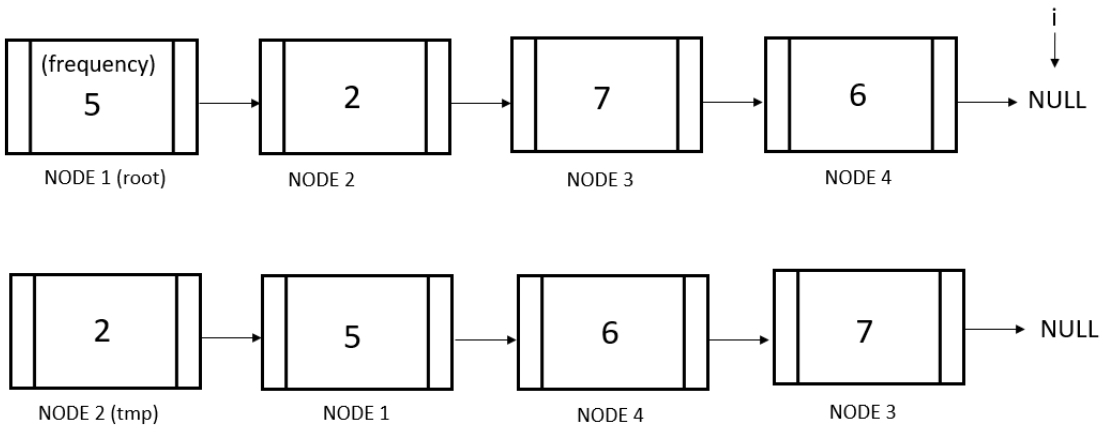
3


i
(frequency)
5
2
7
6
NULL
NODE 1 (root)
NODE 2
NODE 3
NODE 4
j
2
5
NULL
NODE 2 (tmp)
NODE 1
j
2
5
NULL
NODE 2 (tmp)
NODE 1
j
2
5
NULL
NODE 2 (tmp)
NODE 1

4


i
(frequency)
5
2
7
6
NULL
NODE 1 (root)
NODE 2
NODE 3
NODE 4
j
2
5
7
NULL
NODE 2 (tmp)
NODE 1
NODE 3

5


i
(frequency)
5
2
7
6
NULL
NODE 1 (root)
NODE 2
NODE 3
NODE 4
2
5
6
7
NULL
NODE 2 (tmp)
NODE 1
NODE 4
NODE 3

createHuffmanTree:

```
302  //Linkli liste insertion sort ile sıralandıktan sonra Huffman ağacı oluşturulur
303  NODE* createHuffmanTree(NODE* root){
304
305      NODE* newRoot;
306      NODE* tmp; NODE* node;
307      int h;
308
309
310      //Ağaç tamamen oluştuğunda kökte sadece bir node olacağından,
311      //Gösterdiği gelecek bir node olmayacaktır
312      while(root->next->next != NULL){
313
314
315          node = createNode(0); //Yeni bir node oluşturulur
316          node->next = root->next->next;  //Yeni oluşan node'un linkli listeye bağlanabilmesi için
317                                          //Node'un kökten 2 sonraki node'u göstermesi sağlanır
318          root->next->next = node;        //yeni node, 3. node yapılır
319
320          node->frequency = root->frequency + root->next->frequency;  //İlk iki node'un frekanslarının toplamı
321                                                                      //3. node'a aktarılır
322
323          //node'un solu ilk node'u sağı ise ikinci node'u gösterir
324          node->left = root;
325          node->right = root->next;
326          node->left->next = NULL;
327          node->right->next = NULL;
328
329          //Oluşan son yapı tekrar sıralanmak üzere insertionSort fonksiyonuna gönderilir
330          //ve yeni kök (frekansı en küçük node) belirlenir
331
332          root = insertionSort(node);
333
334      }
335
336      node = createNode(0);
337      node->frequency = root->frequency + root->next->frequency;
338      node->left = root;
339      node->right = root->next;
340      node->left->next = NULL;
341      node->right->next = NULL;
342      root = node;
343
344
345
346      h = findHeight(root);
347      printLevelOrder(root,h);
348
349
350      return root;
351  }
```

findHeight:

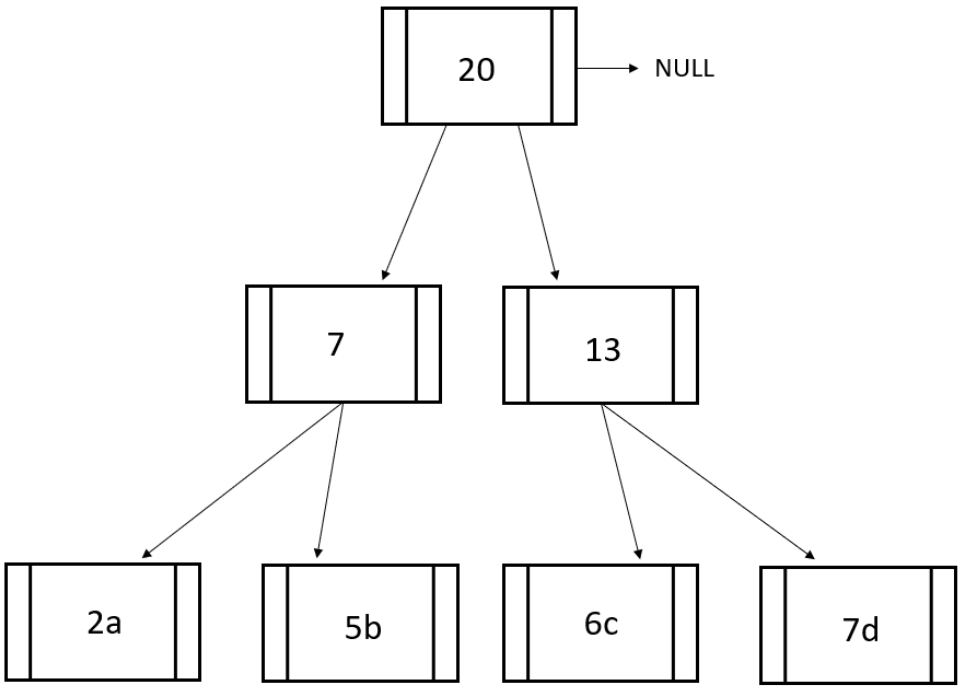
```
199  //Ağaç yüksekliğini bulma fonksiyonu
200  int findHeight(NODE* root)
201  {
202      if (root == NULL) //Kök boşsa
203          return 0;
204      else
205      {
206          //Tüm ağaç gezilir
207          int lheight = findHeight(root->left);
208          int rheight = findHeight(root->right);
209
210          //Bulunan yükseklikler karşılaştırılıp en büyüğü seçilerek
211          //sonunda ağacın en büyük yüksekliği bulunur
212          if (lheight > rheight)
213              return(lheight+1);
214          else return(rheight+1);
215      }
216  }
```

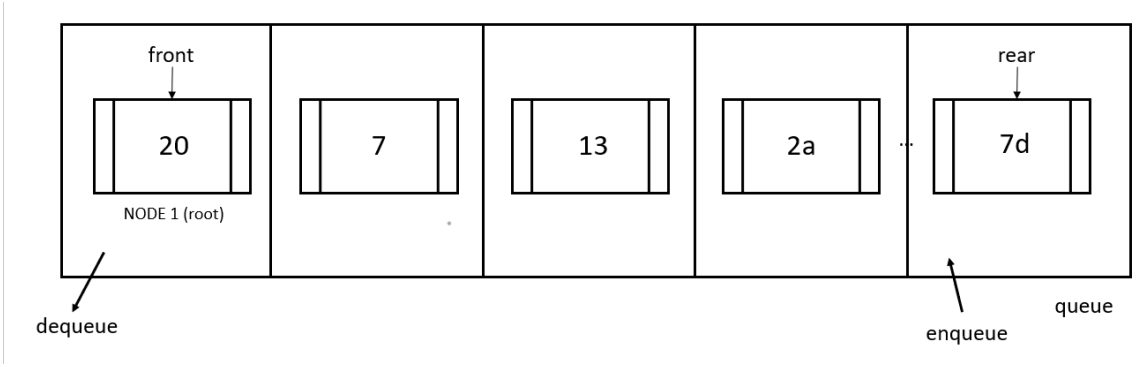
createQueue & enqueue & dequeue:

```
218  //Queue oluşturmak için
219  NODE** createQueue(int* front, int* rear){
220
221      NODE** queue = (NODE**)malloc(sizeof(NODE*)*MAX);
222      *front = *rear = 0;
223
224      return queue;
225  }
226
227  //Kuyruğa eleman ekleme fonksiyonu
228  void enqueue(NODE** queue, NODE* node, int* rear){
229      queue[*rear] = node;
230      (*rear)++;
231  }
232
233  //Kuyrukta öndeki elemanı çıkarma fonksiyonu
234  NODE* dequeue(NODE** queue, int* front){
235      (*front)++;
236      return queue[*front-1];
237  }
```

printLevelOrder:

```
239 //Ağacı yüksekliğine göre yazdırma fonksiyonu
240 void printLevelOrder(NODE* root, int h)
241 {
242     int front,rear;
243 
244     //kuyruk NODE* dizisi olmak üzere oluşturulur
245     NODE** queue = createQueue(&front,&rear);
246     NODE* current = root;
247     NODE* node;
248 
249     int tmp = 2; int i = 0, j;
250     int counter = 0;
251 
252     //Her seviyede kaç eleman olması gerektiği 2'nin kuvveti alınarak (binary tree) hesaplanır
253     tmp = pow(2,i);
254 
255 
256     //Ağacın yüksekliğine ulaşılana kadar
257     while(i != h){
258 
259         //Eğer NODE'un frekansı 0 değilse yani NODE boş (NULL) değilse
260         if(current->frequency != 0){
261             printf("%d",current->frequency);
262             if(current->word != 0)  //NODE harf saklıyorsa
263                 printf("%c",current->word);
264 
265         }
267         printf("\t");
268 
269         counter++; //counter seviyenin tamamen gezilip gezilmediğini kontrol etmek amacıyla kullanılır
270 
271         if(counter == tmp)  //tüm seviye gezildiyse
272         {
273             counter = 0;    //counter sıfırlanır
274             i++;            //bir sonraki seviyeye geçilir
275             tmp = pow(2,i); //yeni seviyede gezilmesi gereken NODE sayısı hesaplanır
276 
277             printf(" \n");  //yeni seviyeye geçildiği için alt satıra geçilir
278         }
279 
280         //NODE'un solunda veya sağında işaret ettiği NODE boş değilse kuyruğa eklenir
281         if(current->left){
282             enqueue(queue,current->left,&rear);
283         }
284         else    //NODE'un solundaki veya sağındaki NODE boş ise kuyruğa boş olduğunu göstermek için 0 frekansa sahip bir node eklenir
285         {
286             node = (NODE*)malloc(sizeof(NODE));
287             node->frequency = 0;
288             queue[rear] = node;
289             rear++;
290         }
291         if(current->right){
292             enqueue(queue,current->right,&rear);
293         }
```


20
NULL
7
13
2a
5b
6c
7d


front
20
NODE 1 (root)
7
13
2a
rear
7d
dequeue
enqueue
queue

SONUÇ

```
Cumlenizi konsoldan yazmak isterseniz 1, dosyadan yazdirmak isterseniz 0 giriniz : 0
Dosya adini giriniz (txt) : deneme


Cumle : "huffman coding is a data compression algorithm"

Linkli Liste :
[  h  ( 2 )  ] --> [  u  ( 1 )  ] --> [  f  ( 2 )  ] --> [  m  ( 3 )  ] --> [  a  ( 5 )  ] --> [  n  ( 3 )  ] --> [  c  ( 2
 2 )  ] --> [  p  ( 1 )  ] --> [  r  ( 2 )  ] --> [  e  ( 1 )  ] --> [  l  ( 1 )  ]

Linkli Liste (Insertion Sort) :
[  l  ( 1 )  ] --> [  e  ( 1 )  ] --> [  p  ( 1 )  ] --> [  u  ( 1 )  ] --> [  r  ( 2 )  ] --> [  t  ( 2 )  ] --> [  g  ( 2
 3 )  ] --> [  m  ( 3 )  ] --> [  i  ( 4 )  ] --> [  o  ( 4 )  ] --> [  a  ( 5 )  ]


TREE :

40
16      24
8       8       10      14
4       4       4       4       5a      5       6       8
2f      2c      2       2h      2t      2g      2       2r                      2d      3s      3m      3n      4i      4o
                                1u      1p                                      11      1e


--------------------------------
Process exited after 1.819 seconds with return value 0
Press any key to continue . . .
```

```
Cumlenizi konsoldan yazmak isterseniz 1, dosyadan yazdirmak isterseniz 0 giriniz : 1
Cumleniz : huffman coding is a data compression algorithm


Linkli Liste :
[  h  ( 2 )  ] --> [  u  ( 1 )  ] --> [  f  ( 2 )  ] --> [  m  ( 3 )  ] --> [  a  ( 5 )  ] --> [  n  ( 3 )  ] --> [  c  ( 2
 2 )  ] --> [  p  ( 1 )  ] --> [  r  ( 2 )  ] --> [  e  ( 1 )  ] --> [  l  ( 1 )  ]

Linkli Liste (Insertion Sort) :
[  l  ( 1 )  ] --> [  e  ( 1 )  ] --> [  p  ( 1 )  ] --> [  u  ( 1 )  ] --> [  r  ( 2 )  ] --> [  t  ( 2 )  ] --> [  g  ( 2
 3 )  ] --> [  m  ( 3 )  ] --> [  i  ( 4 )  ] --> [  o  ( 4 )  ] --> [  a  ( 5 )  ]

TREE :

40
16      24
8       8       10      14
4       4       4       4       5a      5       6       8
2f      2c      2       2h      2t      2g      2       2r                      2d      3s      3m      3n      4i      4o
                                1u      1p                                      11      1e


--------------------------------
Process exited after 9.853 seconds with return value 0
Press any key to continue . . .
```