BLM 112- Programlama Dilleri II

Ders 9: Sıralama & Arama

Yrd. Doç. Dr. Ümit ATİLA

Sıralama

- Bir grup veriyi azalan veya artan şekilde yerleştirme.
- Bilgisayar sistemleri için veri sıralama çok önemlidir.
- Sıralama işlemi, hem arama işlemlerini hem de bir grup veriyi listeleme işlemini hızlandırmaya yarar.
- En popüler sıralama algoritmaları:
  - Insertion sort (Araya ekleme sıralaması)
  - Selection sort (Seçim sıralaması)
  - Bubble sort (Kabarcık sıralaması)
  - Quick sort (Hızlı sıralama)

Kabarcık Sıralaması (Bubble Sort)

- Zaman karmaşıklığı O(n2)
- Eğer n adet elemandan c adet eleman sıralı değilse zaman karmaşıklığı O(c n)
- Algoritmanın tasarımı kolaydır ancak algoritma verimli değildir.
- Az sayıda eleman üzerinde ya da çoğu elemanı zaten sıralanmış listeler üzerinde kullanılabilir.

Kabarcık Sıralaması (Bubble Sort)

- Sıralanacak dizi: [7,3,5,1,2]

• Hareket 1- Çevrim-1 [3,7,5,1,2] • Hareket 1- Çevrim-2 [3,5,7,1,2] • Hareket 1- Çevrim-3 [3,5,1,7,2] • Hareket 1- Çevrim-4 [3,5,1,2,7]	• Hareket 2- Çevrim-1 [3,5,1,2,7] • Hareket 2- Çevrim-2 [3,1,5,2,7] • Hareket 2- Çevrim-3 [3,1,2,5,7] • Hareket 2- Çevrim-4 [3,1,2,5,7]	• Hareket 3- Çevrim-1 [1,3,2,5,7] • Hareket 3- Çevrim-2 [1,2,3,5,7] • Hareket 3- Çevrim-3 [1,3,2,5,7] • Hareket 3- Çevrim-4 [1,3,2,5,7]	• Hareket 4- Çevrim-1 [1,3,2,5,7] • Hareket 4- Çevrim-2 [1,2,3,5,7] • Hareket 4- Çevrim-3 [1,2,3,5,7] • Hareket 4- Çevrim-4 **[1,2,3,5,7]**

Kabarcık Sıralaması (Bubble Sort)

```
26  void bubbleSort(int dizi[],int n)
27  {
28      int gecici;
29
30      for (int i = 0; i < n; i++)
31      {
32          for (int k = 0; k < n - 1 - i; k++)
33          {
34              if(dizi[k]>dizi[k+1])
35              {
36                  gecici=dizi[k];
37                  dizi[k] = dizi[k + 1];
38                  dizi[k + 1] = gecici;
39              }
40          }
41      }
42  }
```

Kabarcık Sıralaması (Bubble Sort)Sıralaması

```c
#include <stdio.h>

void bubbleSort(int [],int);
int main(void)
{
    int i=0,a[5];
    printf("Siralamak istediğin 5 sayi gir\n");
    while(i<5){
        scanf("%d",&a[i]);
        i++;
    }
    i=0;
    bubbleSort(a,5);

    printf("Bubble sort isleminden sonra...\n");
    while(i<5){
        printf("%d ",a[i]);
        i++;
    }
    return 0;
}
```

Araya Ekleme Sıralaması(Insertion Sort)

- Sıralı bir listeye eleman eklemek için uygundur.
- Sıralı bir listeye eleman eklemenin karmaşıklığı : O(n)
- Eğer liste veya dizi sıralı değilse karmaşıklığı : O (n^2)

Araya Ekleme Sıralaması(Insertion Sort)

- Sıralanacak dizi: [7,3,5,8,2]
- Başlangıç durumu : [7][3,5,8,2]

Önce	Sonra
[7,3][5,8,2]	[3,7][5,8,2]
[3,7,5][8,2]	[3,5,7][8,2]
[3,5,7,8][2]	[3,5,7,8][2]
[3,5,7,8,2][ ]	[2,3,5,7,8][ ]

Araya Ekleme Sıralaması(Insertion Sort)

```
23 void insertionSort(int dizi[], int n)
24 {
25     int ekle,k,i;
26     for (i = 1; i < n; i++)
27     {
28         ekle = dizi[i];
29         for (k = i - 1; k >= 0 && ekle <= dizi[k]; k--)
30             dizi[k + 1] = dizi[k]; //Geriye kaydırma
31         dizi[k + 1] = ekle;        //Uygun yer boşaltıldı eklendi
32     }
33 }
```

Araya Ekleme Sıralaması(Insertion Sort)

```c
#include <stdio.h>

void insertionSort(int [],int);
int main(void)
{
    int i=0,a[5];
    printf("Siralamak istediğin 5 sayi gir\n");
    while(i<5){
        scanf("%d",&a[i]);
        i++;
    }
    i=0;
    insertionSort(a,5);

    printf("Insertion sort isleminden sonra...\n");
    while(i<5){
        printf("%d ",a[i]);
        i++;
    }
    return 0;
}
```

Seçim Sıralaması (Selection Sort)

- Eğer bir eleman olması gereken yerde ise bulunduğu sıra değiştirilmez.
- Yarı yarıya sıralanmış bir grup veri olması durumunda eleman yeri değişimi azdır.
- Listedeki ilk elemanı al ve bu elemanı diğer elemanlar arasından en küçüğü ile değiştir. Bu işlemi son elemana kadar tekrar et.

Önce	Sonra	
`[][7,3,5,1,2]`	`[1][3,5,7,2]`	1 ve 7 değiştirildi
`[1][3,5,7,2]`	`[1,2][5,7,3]`	2 ve 3 değiştirildi
`[1,2][5,7,3]`	`[1,2,3][7,5]`	3 ve 5 değiştirildi
`[1,2,3][7,5]`	`[1,2,3,5][7]`	5 ve 7 değiştirildi
`[1,2,3,5][7]`	`[1,2,3,5,7][]`	son

Seçim Sıralaması (Selection Sort)

```
25  void selectionSort(int dizi[],int n)
26  {
27      int i,j;
28      int index, enkucuk;
29      for (i = 0; i < n - 1; i++)
30      {
31          enkucuk = dizi[n - 1];
32          index = n - 1;
33          for (j = i; j < n - 1; j++)
34          {
35              if (dizi[j] < enkucuk)
36              {
37                  enkucuk = dizi[j];
38                  index = j;
39              }
40          }
41          dizi[index] = dizi[i];
42          dizi[i] = enkucuk;
43      }
44  }
```

Seçim Sıralaması (Selection Sort)

```c
#include <stdio.h>

void selectionSort(int [],int);
int main(void)
{
    int i=0,a[5];
    printf("Siralamak istediğin 5 sayi gir\n");
    while(i<5){
        scanf("%d",&a[i]);
        i++;
    }
    i=0;
    selectionSort(a,5);

    printf("Selection sort isleminden sonra...\n");
    while(i<5){
        printf("%d ",a[i]);
        i++;
    }
    return 0;
}
```

Hızlı Sıralama (Quick Sort)

- Böl ve yönet stratejisine göre çalışır.
- Liste iki eşit parçaya bölünür.
- Ortadaki değerden küçük değerler sol tarafta, büyük olanlar sağ tarafta toplanır.
- Bu işlem her bir parça için tekrarlanır.
- Algoritma bölünmüş her bir kısım için ayrı ayrı çalıştırılan öz yinelemeli(rekürsif) çağrı ile uygulanır.
- En hızlı algoritma olmasına rağmen, az sayıda elemana sahip listeler veya çoğu elemanı zaten sıralı olan listeler üzerinde uygulanmayabilir.

Hızlı Sıralama (Quick Sort)

```
23 void quickSort(int dizi[],int sol,int sag)
24 {
25     int qSol,qSag,ortadaki,gecici;
26     //Dizi ikiye parçalanıyor
27     qSol=sol;
28     qSag=sag;
29     ortadaki = dizi[(sol+sag)/2];  //Sınır değer
30     do
31     {
32         while(dizi[qSol]< ortadaki && qSol<sag)
33             qSol++;
34         while(ortadaki< dizi[qSag] && qSag>sol)
35             qSag--;
36         if(qSol<=qSag)
37         {
38             gecici = dizi[qSol];
39             dizi[qSol] = dizi[qSag];
40             dizi[qSag] = gecici;
41             qSol++; qSag--;
42         }
43     }
44     while(qSol<=qSag); //parçalama bitti
45     if(sol<qSag) quickSort(dizi,sol,qSag);
46     if(qSol<sag) quickSort(dizi,qSol,sag);
47 }
```

Hızlı Sıralama (Quick Sort)

```c
#include <stdio.h>

void quickSort(int [],int,int);
int main(void)
{
    int i=0,a[5];
    printf("Siralamak istediğin 5 sayi gir\n");
    while(i<5){
        scanf("%d",&a[i]);
        i++;
    }
    i=0;
    quickSort(a,0,4);

    printf("Quick sort isleminden sonra...\n");
    while(i<5){
        printf("%d ",a[i]);
        i++;
    }
    return 0;
}
```

Arama

- Bir dizi içerisinde belli bir elemanı bulma sürecine arama denir.
- İki arama tekniği tartışılacak.
  - Doğrusal Arama (Linear Search)
  - İkili Arama (Binary Search)

Doğrusal Arama (Linear Search)

- Dizinin her bir elemanını aranan ile karşılaştırır.
- Dizi sıralı değilse, aranan ilk elemanda olabilir, son eleman da.
- En kötü durumda N elemanlı dizide algoritmanın karmaşıklığı O(n) dir.
- Büyük boyutlu dizilerde kullanımı uygun değildir.

Doğrusal Arama (Linear Search)

```
21  int linearSearch(int dizi[],int aranan,int n)
22  {
23      for (int i = 0; i < n; i++)
24      {
25          if (dizi[i] == aranan)
26              return i;
27      }
28      return -1;
29  }
```

Doğrusal Arama (Linear Search)

```c
#include <stdio.h>

int linearSearch(int [],int,int);
int main(void)
{
    int dizi[] = { 1, 3, 5, 7, 8, 10, 11 };
    int sonuc, aranan,i;
    for (i = 0; i < 7; i++)
        printf("%d ",dizi[i]);

    printf("Arananı giriniz:");
    scanf("%d",&aranan);

    sonuc = linearSearch(dizi, aranan,7);
    if (sonuc == -1)
        printf("\nAranan dizide yok\n");
    else
        printf(sonuc + ". sırada bulundu\n");
}
```

İkili Arama (Binary Search)

- Doğrusal arama küçük boyutlu veya sıralanmamış dizilerde uygulanabilir.
- Ancak büyük boyutlu dizilerde doğrusal arama verimli değildir.
- Eğer dizi sıralanmış ise yüksek hızlı olan ikili arama kullanılabilir.
- İkili arama her karşılaştırmadan sonra sırlanmış dizinin bir tarafını değerlendirmeden eler.

İkili Arama (Binary Search)

- Algoritma, dizinin ortasında yer alan elemanı aranan ile karşılaştırır.
  - Eğer eşit ise aranan bulunmuş olur.
  - Eğer aranan < ortadaki ise aramayı dizinin ilk yarısında sürdür.
  - Eğer aranan > ortadaki ise aramayı dizinin diğer yarısında sürdür.
  - Aranan, ortadakine eşit olana kadar ya da alt dizide aranana eşit olmayan tek bir eleman kalana kadar bu işlemleri tekrarla.

İkili Arama (Binary Search)

```c
#include <stdio.h>

int binarySearch(int [],int,int,int);
int main(void)
{
    int dizi[] = { 1, 3, 5, 7, 8, 10, 11 };
    int sonuc, aranan,i;
    for (i = 0; i < 7; i++)
        printf("%d ",dizi[i]);

    printf("Arananı giriniz:");
    scanf("%d",&aranan);

    sonuc = binarySearch(dizi, aranan,0,6);
    if (sonuc == -1)
        printf("\nAranan dizide yok\n");
    else
        printf(sonuc + ". sırada bulundu\n");
}
```

İkili Arama (Binary Search)

```
21  int binarySearch(int dizi[],int aranan,int sol,int sag)
22  {
23      int orta;
24      while (sol <= sag)
25      {
26          orta = (sol + sag) / 2; //Ortadaki elemanın indisi hesaplanıyor
27          if (aranan == dizi[orta])
28              return orta;
29          else if (aranan > dizi[orta])
30              sol = orta + 1;
31          else
32              sag = orta - 1;
33      }
34      return -1;
35  }
```

İkili Arama (Binary Search)

- Çok hızlıdır. 2^n > eleman sayısı ise en fazla n adımda biter.
- 30 elemanlı dizide arama 5 adımda biter ($2^5 > 30$)
- En kötü senaryoda 1023 elemanlı bir dizide arama sadece 10 karşılaştırma sürer.
- 1024 ü tekrarlı bir şekilde 2 ile bölersek 512, 256, 128, 64, 32, 16, 8, 4, 2 ve 1 elde ederiz.
- 1024 (2^{10}) 2 ile sadece 10 kez bölündüğünde 1 elde edilir.
- İkili aramada 2 ile bölme bir karşılaştırmaya denktir.

İkili Arama (Binary Search)

- 1048576 (2^{20}) elemanlı dizide en fazla 20 adımda aranan bulunur
- 1 milyar elemanlı dizide en fazla 30 karşılaştırmada aranan bulunur.
- Doğrusal aramaya göre inanılmaz bir performans artışı vardır. Doğrusal arama ortalama dizideki elemanların yarısı ile karşılaştırma yapar.
- 1 milyar eleman düşünüldüğünde bu fark 30 karşılaştırma ile 500 milyon karşılaştırma arasındaki farktır.