Veri Yapıları ve Algoritmalar 3. Ödev

Çoklu Öncelikli Kuyruk

Öğrenci Adı: Mehmet Ali Duran

Öğrenci Numarası: 21011090

Dersin Eğitmeni: Mehmet Amaç Güvensan

Video Linki: https://youtu.be/3POuqgzZ70U

1- Problem Tanımı:

Bu ödevde NxM'lik bir matrisin her satırında max-heap-tree özelliği taşıyan öncelikli kuyruk oluşturulması istenmektedir. Bu kuyrukların kapasitesi farklı olabilmektedir. Kuyruklar matrisin satırlarına yerleştirildikten sonra heap özelliği kazandırılır. Her adımda öncelikli kuyrukların ilk elemanlarından en yüksek önceliğe sahip olan ilgili kuyruktan çıkartılır. Bu adımlar kuyruklarda eleman kalmayıncaya kadar devam eder. Sonuç olarak ise kuyrukların bitme sırası ekrana basılmalıdır. Bu problem çözülürken matris yapısının kullanılması istenmiştir.

2- Problemin Çözümü:

İlk olarak kullanıcıdan geçerli N ve M değerleri alınmalıdır. Bu değerler 1'den küçük olursa ortada bir matris oluşamaz bu sebeple program en az 1x1 girilecek şekilde tasarlanmıştır. Matris bellekte dinamik olarak oluşturulduktan sonra bütün hücreler -1 değeri ile doldurulmuştur. Daha sonra her bir satırdaki kuyruğun boyutu kullanıcıdan istenir. Burada dikkat edilmesi gereken husus girilen değer 1'den küçük olamaz ve M değerinden büyük olamaz. Geçerli aralıkta sayılar alındıktan sonra ilgili kuyruklar rastgele eşsiz sayılar ile doldurulmalıdır. Bunun için **fillMatrixWithRandomNumbers()** fonksiyonu yazılmıştır. Gelinen noktada matris işlemeye hazır bir şekilde elde edilmiştir. Sırada rastgele doldurulan kuyruklara heap ağacı özelliği kazandırılması gerekmektedir. **max_heapify()** fonksiyonu bu işlemi gerçekleştirir. Bütün kuyruklar bitmediği sürece dönen bir döngünün içinde artık her adımda en büyük eleman kuyruktan çıkarılır, çıkarıldığı kuyruk tekrar heapify işlemine tabi tutulur. Bu süreçte biten bir kuyruk olursa bu kuyruğun matristeki satır indisi(aslında bu bize bitme sırasını belirtmektedir) result dizisinde tutulur. Döngü bitiminde bütün kuyruklar boşalmıştır ve bitme sırası result dizisinde elde edilmiştir. Son olarak bu dizi ekrana bastırılır.

3- Karşılaşılan Sorunlar:

Karşılaşılan en kritik sorunlardan birisi rastgele üretilen sayıların eşsiz olmasını sağlamak. Her adımda 0 – K aralığında rastgele bir sayı üretilse ve bu değer kuyruğa yazılsa birbirinin aynısı olan değerlerin kuyruğa gelme ihtimalleri var. Bu durumu önlemek için şöyle bir yol izlendi. 0 – K aralığındaki sayılar ilk olarak bir dizide tutulur. Daha sonra sıra ile her sayı rastgele bir indisteki sayı ile yer değiştirir. Bu işlem dizi boyunca yapılır ve sonuç olarak içinde her sayıdan bir adet bulunan bir dizi elde edilir. Bu dizideki elemanlar ilgili kuyruğa kuyruğun eleman sayısı kadar sıra ile eklenirse sorun çözülmüş olacaktır.

4- Ekran çıktıları:

İlk örnekte takibi kolay olması açısından ödev dosyasında verilen örneği ele aldım. Rastgele değerler birbiri ile uyuşmayacağından bunları kendim elle atadım.

```
int main(){

    int N, M, i, max, resultOrder=0;
    int* lengthOfEachQueue;
    int* result;

    N = 4;
    M = 5;


    int** matrix = createMatrix(N, M);

    result = (int*)malloc(N*sizeof(int));

    lengthOfEachQueue = (int*)malloc(N*sizeof(int));
    lengthOfEachQueue[0] = 2;
    lengthOfEachQueue[1] = 3;
    lengthOfEachQueue[2] = 4;
    lengthOfEachQueue[3] = 4;
```

```
    matrix[0][0] = 20;
    matrix[0][1] = 19;
    matrix[0][2] = -1;
    matrix[0][3] = -1;
    matrix[0][4] = -1;

    matrix[1][0] = 17;
    matrix[1][1] = 5;
    matrix[1][2] = 18;
    matrix[1][3] = -1;
    matrix[1][4] = -1;

    matrix[2][0] = 1;
    matrix[2][1] = 9;
    matrix[2][2] = 8;
    matrix[2][3] = 3;
    matrix[2][4] = -1;

    matrix[3][0] = 12;
    matrix[3][1] = 14;
    matrix[3][2] = 13;
    matrix[3][3] = 15;
    matrix[3][4] = -1;
```

Bu değerler sonucunda oluşan çıktı şu şekildedir:
İlk olarak rastgele sayılar ile doldurulan matris ekrana basılır.(1) Ardından her satırındaki kuyruğa heap özelliği kazandırılan matris ekrana basılır(2).

```
Matrix filled with random numbers:
 20  19  -1  -1  -1
 17   5  18  -1  -1
  1   9   8   3  -1
 12  14  13  15  -1
```

1

```
Matrix constructed as heap tree:
 20  19  -1  -1  -1
 18   5  17  -1  -1
  9   3   8   1  -1
 15  14  13  12  -1
```

2

```
 19  -1  -1  -1  -1
 18   5  17  -1  -1
  9   3   8   1  -1
 15  14  13  12  -1

 -1  -1  -1  -1  -1
 18   5  17  -1  -1
  9   3   8   1  -1
 15  14  13  12  -1

 -1  -1  -1  -1  -1
 17   5  -1  -1  -1
  9   3   8   1  -1
 15  14  13  12  -1
```

```
 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  9   3   8   1  -1
 15  14  13  12  -1

 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  9   3   8   1  -1
 14  12  13  -1  -1

 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  9   3   8   1  -1
 13  12  -1  -1  -1

 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  9   3   8   1  -1
 12  -1  -1  -1  -1

 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  9   3   8   1  -1
 -1  -1  -1  -1  -1

 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  8   3  -1   1  -1
```

```
 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  8   3  -1   1  -1
 -1  -1  -1  -1  -1

 -1  -1  -1  -1  -1
  5  -1  -1  -1  -1
  3   1  -1  -1  -1
 -1  -1  -1  -1  -1

 -1  -1  -1  -1  -1
 -1  -1  -1  -1  -1
  3   1  -1  -1  -1
 -1  -1  -1  -1  -1

 -1  -1  -1  -1  -1
 -1  -1  -1  -1  -1
  1  -1  -1  -1  -1
 -1  -1  -1  -1  -1

 -1  -1  -1  -1  -1
 -1  -1  -1  -1  -1
 -1  -1  -1  -1  -1
 -1  -1  -1  -1  -1

Finished order: 1 4 2 3
```

Devam eden adımlarda bütün kuyruklar boşalmış ve sonuç olarak kuyruklar önce bitme sırasına göre ödev dosyasında olduğu gibi bulunmuştur.

İkinci örnek için N ve M değerleri 3 olarak seçilmiştir. Kuyruk boyutları ise 3 2 1 şeklinde girilmiştir. Oluşan matris aşağıdaki gibidir.

```
Please enter the number of rows N and columns M values(N M): 3 3
Enter the length of the queue in each line
length of 1th row: 3
length of 2th row: 2
length of 3th row: 1
Matrix filled with random numbers:
   7   0   4
   3   8  -1
   2  -1  -1
Matrix constructed as heap tree:
   7   0   4
   8   3  -1
   2  -1  -1
```

Sonuç ise bu şekilde olmuştur.