Priority Queue: Havalimanı Uçuş Yönetim Sistemi

Muhammet Cüneyd Kurtbaş
Kocaeli Üniversitesi
Bilgisayar Mühendisliği (I.Öğretim)
200201132
cuneydkurtbas@gmail.com

***Abstract*—Standart kuyruk veri yapısı önceliklendirme eksiği nedeniyle, bir çok durumda veya problemde kullanılmak için uygun olmayabilir. Gerçek hayatta uygulanan kuyruk yapılarında öncelik durumu dikkate alınır. Önceliği yüksek olanlar her zaman önce işlem görürler.**

Örneğin, hastane poliklinik muayene sıralarında, 65 yaş üstü, 7 yaş altı, Acil vakalar, Personel, Personel yakını gibi öncelik dereceleri vardır. Öncelik derecesi yüksek olan kişiler son sıradan girse bile en önce muayene olmaktadırlar. Diğer bir örnek ise, dolmuş/otobüs oturma alanları verilebilir. 65 yaş üstü, 7 yaş altı, hamileler, engelliler gibi insanlar dolmuşta oturma önceliğindedirler.

Öncelik kriteri kuyruk türüne göre farklılık göstermektedir. Kuyruğa eklenen elemanın kendisi veya herhangi bir özelliği öncelik kriteri olabilir. Öncelik kuyrukları Dizi veya Bağlı Liste olarak uygulanabilir.

***Index Terms*—priority, queue, linked list, array, data structure**

I. GİRİŞ

Diğer veri yapılarında olduğu gibi kuyrukta bulunan elemanlar, string veya integer gibi basit veri türünde olabileceği gibi bir nesne de olabilir. Öncelik kriteri kuyruk türüne göre farklılık göstermektedir. Bu uygulamanın amacı; Bir iniş bir kalkış olmak üzere 2 pisti bulunan İstanbul Havalimanı'nda gün içerisinde (1-24 saat dilimi boyunca) yapılan uçuşların yönetimi için bir sistem geliştirilecektir. Havalimanında aynı anda sadece bir uçak kalkış yapabiliyorken sadece bir uçak iniş yapabilmektedir. Uçakların her biri iniş ve kalkışta farklı önceliklere sahiptir ve bir günde en fazla 24 uçak iniş için izin isteyebilmektedir. Havalimanındaki uçakların öncelik sırası, iniş saati, gecikme süresi ve kalkış saati bilgileri kullanılarak; iniş pistini ve kalkış pistini kullanım sırasının belirlenmesi hedeflenmektedir.

II. YÖNTEM

Havalimanına iniş yapacak uçaklar öncelikle kuleden iniş yapabilmek için izin talep edecektir. İniş izni talep eden her bir uçak için havalimanında yeterli kapasite olup olmadığı kontrol edilecek, pist boş ise iniş yapılmak istenen saate izin verilecek ve uygun yere eklenecektir. Aksi halde uçakların iniş sıralaması önceliğe göre belirlenmelidir. Uçakların iniş ve kalkışları, 1.Ambulans uçağı, 2.Savaş uçağı, 3.Yolcu uçağı, 4.Kargo uçağı önceliğine göre belirlenecektir. Havalimanına iniş talep eden uçakların önceliği, uçak numarası ve talep ettiği iniş saati input.txt dosyasından okunacaktır. Havalimanına iniş yapan her uçağın, kalkış için bekleme süresi 1 saat ve uçakların kalkış saatine, ötelenmeden dolayı oluşan gecikme süreleri dâhil edilecektir. Aynı önceliğe sahip iki uçak, aynı saatte kalkış yapacaksa öncelik ilk iniş yapan uçağa verilecektir.

Uygulamada önceliği yüksek olan uçaklar nedeniyle önceliği düşük olan herhangi bir uçağın uçuş saati, en fazla 3 kez ertelenebilecek, 3'ten fazla ertelenme durumu söz konusuysa, öncelik gözetilmeksizin beklemede olan uçağın kalkışı gerçekleştirilecektir. Her yeni input satırı okunduğunda, kalkış yapacak olan uçakların bulunduğu output.txt dosyası güncellenecektir. output.txt içeri her satırda Öncelik ID, Uçak ID, Talep Saati, İzin Verilen İniş Saati, Erteleme Süresi ve Kalkış Saati bilgilerini içerecektir.

III. UYGULAMA

Uygulamamızın kod tarafına geçmeden önce aşağıda gösterilen örnek bir modele göre iniş izni talepleri ve uçaklara ait bilgileri alarak adım adım öncelikli kuyruğumuzu gerçekleştirelim;

Fig. 1.

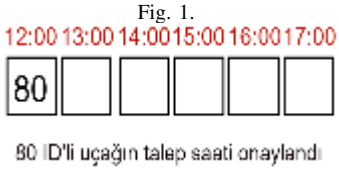

Fig. 2.

Fig. 3.

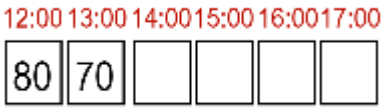

Fig. 4.

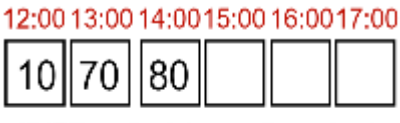

Fig. 5.

Fig. 6.

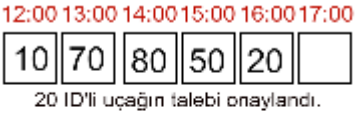

Fig. 7.

C KODLARI

```c
struct plane{
    int planeID;
    int priorityID;
    int landingTime;
    int delayTime;
    int departureTime;
    struct plane *next;
};

struct plane *landingFirst = NULL,
             *landingLast = NULL;
struct plane *departureFirst = NULL,
             *departureLast = NULL;
int capacity = 24;
```

```
void enqueue(int priorityID, int planeID,
int landingTime, int departureTime,
int delayTime){

struct plane *temp =
(struct plane*) malloc(sizeof(struct plane));
temp->planeID = planeID;
temp->priorityID = priorityID;
temp->delayTime = delayTime;
temp->landingTime = landingTime;
temp->departureTime = departureTime;
temp->next = NULL;

    if(departureFirst == NULL){
        departureFirst = temp;
        departureLast = temp;
    }else{

        departureLast->next = temp;
        departureLast = temp;
    }
}
```

```
void dequeueLanding(){
    if(landingFirst != NULL){
        if((landingFirst)->next == NULL){
            free(landingFirst);
            landingFirst = NULL;
            landingLast = NULL;
        }else{
            struct plane *temp = landingFirst;
            landingFirst = temp->next;
            free(temp);
        }
```

```
void CreateDepartureQueue(){
    clearDeparture(&departureFirst, &departureLast);
    struct plane *temp = landingFirst;
    while(temp != NULL){
        enqueue(temp->priorityID, temp->planeID,
          temp->landingTime, temp->departureTime,
            temp->delayTime);
        temp = temp->next;
```

```
int FreePlace(){
    if(landingFirst == NULL)
        return 0;
    else if(landingFirst == landingLast)
    else{
        struct plane *temp = landingFirst;
        int counter = 0;
        while(temp != NULL){
            counter++;
            temp = temp->next;
        }
        return counter;
    }
```

```
void removeFromDeparture(int planeID){
    if(departureFirst == NULL){
        return;
    }
    else if(planeID == departureFirst->planeID){
        if(landingFirst == departureLast){
            free(departureFirst);
            departureFirst = NULL;
            departureLast = NULL;
            return;
        }
        struct plane *temp = departureFirst;
        departureFirst = departureFirst->next;
        free(temp);
        return;
    }else if(planeID == departureLast->planeID){
        struct plane *temp = departureFirst;
        while(temp->next != departureLast){
            temp = temp->next;
        }
        struct plane *destroy = departureLast;
        departureLast = temp;
        departureLast->next = NULL;
        free(destroy);
        return;
    }
```

IV. DENEYSEL SONUÇLAR

Öncelikli kuyruk prensipte yine bir kuyruk çeşidi olmakla birlikte sadece bir davranışı normal kuyruktan farklılık gösterir. O da ilk giren ilk çıkar (first-in, first-out) şeklinde değil, nasıl sıraya girilirse girilsin daha önemli olan önce çıkar, şeklinde çalışmasıdır.

Temel olarak fonksiyon/metodlarında bir farklılık bulunmaz. Sadece önceliklendirme işleminin nasıl yapılacağının tanımlanması gerekir.

Bağlı liste oluşturma işleminde öncelikli olarak bir tane root düğümü oluşturup ardından gelen düğümleri sıra ile birbirlerini next pointer'ı ile işaret etmektedir. Kuyruk işlemi de bu işleme çok benzemektedir. Sadece kuyruğa eklenen en son elemanı ayrı bir biçimde tutmamız gerekmektedir. Kısacası root düğümünün yanında bir tane de next düğümü oluşturulur. Dolayısıyla Kuyruk birbirine bağlı liste haline dönüşmüş olacaktır. Kuyruğa Eleman ekleme ve kuyruktan eleman çıkarma işlemlerinin mantığı ise Aynı şekilde eleman ekleme işlemi rear üzerinden, çıkarma işlemi ise front üzerinden yapılır. Başlangıçta ilk eklenen eleman hem öndeki eleman hem de arkadaki eleman olur. Bu mantık üzerinden bağlı liste ile kuyruk yapısı oluşturulur.

Öncelikli bir kuyruğa alma mekanizmasının kullanılması aşağıdaki avantajları sağlayabilir:

- Uygulamaların kullanılabilirlik veya performans için öncelik belirlenmesini gerektiren iş gereksinimlerini karşılamasına olanak sağlar. Örneğin, belirli müşteri gruplarına farklı düzeylerde hizmet sunulabilir.

- İşletim maliyetlerinin en aza indirilmesine yardımcı olabilir. Tek kuyruklu yaklaşımda gerekirse tüketici sayısının ölçeğini küçültebilirsiniz. Yüksek öncelikli iletiler yine ilk olarak işlenir (ancak büyük olasılıkla daha yavaş bir şekilde) ve düşük öncelikli iletiler daha uzun bir süre için ertelenebilir. Her bir kuyruk için ayrı bir tüketici havuzu içeren birden fazla ileti kuyruğu yaklaşımını uyguladıysanız düşük öncelikli kuyruklar için tüketici havuzunu küçültebilir hatta çok düşük öncelik kuyruklarda ileti olup olmadığını dinleyen tüm tüketicileri durdurarak bu kuyrukların işlemlerini askıya alabilirsiniz.
- Birden fazla ileti kuyruğu yaklaşımı, iletileri işleme gereksinimlerine bağlı olarak bölümleyerek uygulama performansı ve ölçeklenebilirliğinin en üst düzeye çıkarılmasına yardımcı olabilir. Örneğin, çok önemli görevler hemen çalıştırılan alıcılar tarafından işlenecek şekilde önceliklendirilebilir. Önemi daha düşük arka plan görevleri de daha az meşgul zamanlarda çalıştırılması zamanlanan alıcılar tarafından işlenebilir.

Bu düzen, aşağıdakilerin geçerli olduğu senaryolarda kullanışlıdır:

- Sistem farklı önceliklere sahip birden çok görevi işlemelidir.
- Farklı kullanıcı veya kiracılara farklı önceliklerle hizmet sunulması gereklidir.

Uygulama, Windows işletim sistemi üzerinde Dev C++ IDE'si kullanılarak C dilinde gerçeklenmiştir. Sonuçlarına ait ekran görüntüleri;

```
ONCELIKLI KUYRUK:
Oncelik - Ucak - Inis - Erteleme
|    1 -   11 -    1 -    0 |
|    1 -   12 -    2 -    1 |
|    2 -   23 -    3 -    1 |
|    2 -   27 -    4 -    2 |
|    3 -    7 -    5 -    3 |
|    4 -   28 -    6 -    3 |
|    3 -   24 -    8 -    0 |
|    2 -   13 -    9 -    0 |
|    2 -    6 -   10 -    0 |
|    2 -    3 -   11 -    0 |
|    3 -    4 -   12 -    2 |
|    2 -    2 -   13 -    0 |
|    1 -    1 -   14 -    0 |
|    3 -    8 -   15 -    1 |
|    1 -   25 -   16 -    0 |
|    3 -   14 -   17 -    1 |
|    4 -    5 -   18 -    2 |
|    3 -   21 -   19 -    0 |
|    2 -   16 -   20 -    0 |
|    4 -   20 -   21 -    0 |
|    4 -   26 -   22 -    2 |
|    1 -   19 -   23 -    0 |
|    3 -   18 -   24 -    0 |
```

```
+++HAVALIMANI DURUMU+++
KAPASITE: 24
INIS PISTI DOLULUK: 23
INISE ACIK KULLANIM: 1
KALKIS PISTI DOLULUK: 23
KALKISA ACIK: 1
```

output.txt					input.txt
11	1	1	0	2	1 1 14
12	1	2	1	3	2 2 13
23	2	3	1	4	2 3 11
27	2	4	2	5	3 4 10
7	2	5	3	6	4 5 16
28	3	6	3	7	2 6 10
24	8	8	0	9	3 7 2
13	9	9	0	10	3 8 14
6	10	10	0	11	4 9 17
3	11	11	0	12	4 10 13
4	10	12	2	13	1 11 1
2	13	13	0	14	1 12 1
1	14	14	0	15	2 13 9
8	14	15	1	16	3 14 16
25	16	16	0	17	3 15 13
14	16	17	1	18	2 16 20
5	16	18	2	18	3 17 1
21	19	19	0	20	3 18 24
16	20	20	0	21	1 19 23
20	21	21	0	22	4 20 21
26	20	22	2	23	3 21 19
19	23	23	0	24	4 22 1
18	24	24	0	1	2 23 2
					3 24 8
					1 25 16
					4 26 20
					2 27 2
					4 28 3

REFERENCES

- https://docs.microsoft.com/tr-tr/azure/architecture/patterns/priority-queue,Erişim 10.12.2021
- https://hasscript.com/908/oncelikli-kuyruk-prioty-queue-nedir,Erişim 08.12.2021
- Class Priority Queue, baskent.edu.tr/ tkaracay
- Öncelikli Kuyruk Dizi Gerçekleştirimi, Bozok Üniversitesi
- Doğrusal Veri Yapıları, medium.com/@tolgahan.cepel
- Algoritma ve Veri Yapıları İleri Seviye, BTK Akademi
- draw.io
- Overleaf: New to LaTeX?