Modul 5. Queue

Jump to bottom

Ivan Sholana edited this page 2 weeks ago · 16 revisions

Queue

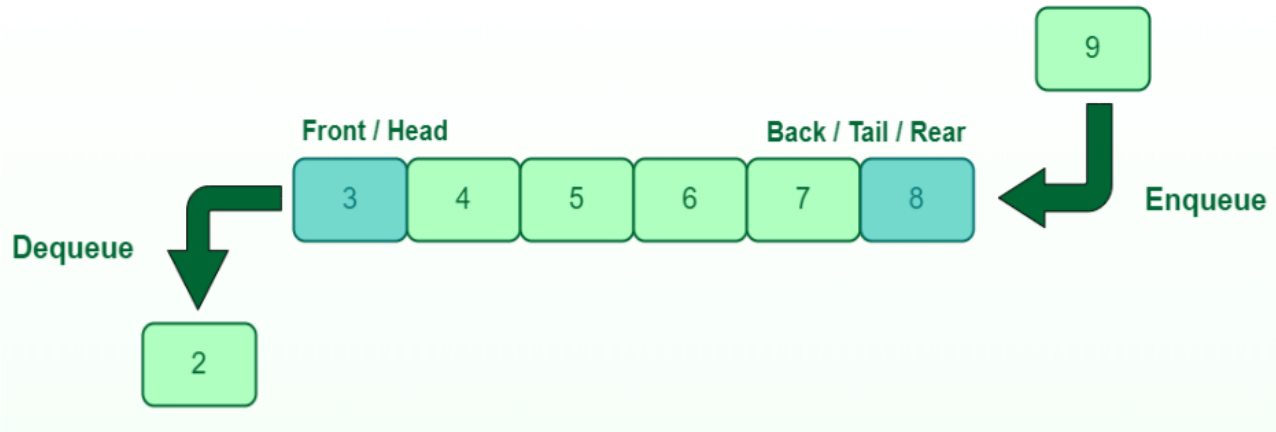
Terminologi

- **front** – merupakan data yang paling depan pada queue.
- **rear** – merupakan data yang paling belakang pada queue.

Definisi

Queue merupakan struktur data linear yang menggunakan prinsip **First In First Out** (FIFO). Dengan prinsip FIFO, elemen pertama yang dimasukkan akan menjadi elemen pertama yang akan dikeluarkan. Setiap elemen pada queue selalu ditambahkan di akhir dan dikeluarkan di depan. Contoh penerapannya adalah barisan orang yang menunggu bus. Orang pertama yang pada antrian menjadi yang pertama yang dapat menaiki bus.

Dalam modul ini, ADT Queue akan diimplementasikan pada struktur data Array Statis. Implementasi pada struktur data lainnya, seperti Linked List akan berbeda secara teknis tetapi secara umum idenya tetap sama.



Queue Data Structure

Sumber Gambar: Geeks for Geeks

Penerapan queue biasa digunakan dalam BFS (Breadth-First-Search) graph transversal.

Operasi Dasar

- **isEmpty** – untuk memeriksa apakah queue kosong atau tidak.
- **size** – untuk mendapatkan data size pada queue.
- **enqueue** – operasi untuk menambahkan data pada antrian dari belakang.
- **dequeue** – operasi untuk menghapus data terdepan pada antrian.
- **front** – untuk mendapatkan data terdepan pada antrian.

Kompleksitas waktu semua operasi dilakukan secara konstan O(1), kecuali operasi dequeue yang memiliki kompleksitas waktu linear O(N).

Aplikasi Queue

Queue biasa digunakan pada BFS (Breafth First Search) graph transersal yang nantinya akan dijelaskan pada modul selanjutnya. Contoh problem lain yang dapat diselesaikan dengan queue adalah melakukan generate binary number dari 1 hingga n.

Implementasi ADT: `Queue` (Static Array-Based)

Dalam modul ini, Queue diimplementasikan dengan menggunakan struct dan array, sehingga konsekuensinya adalah queue yang dibuat tidak fleksibel. Hal ini disebabkan karena tidak adanya alokasi memori secara dinamis. Untuk implementasi menggunakan Linked List, akan ada pada modul di kemudian hari.

- ## Struktur Queue

Queue direpresentasikan oleh struktur yang menyimpan array bertipe data `int` serta `front` dan `rear` yang merepresentasikan indeks paling depan dan paling belakang dari data.

```
#define MAX 10

typedef struct {
    int data[MAX];
    int front;
    int rear;
} Queue;
```

- ## Initialisasi Queue

Queue diinisialisasikan dengan cara memberi nilai index `front` dan `rear` dengan nilai -1 yang menandakan bahwa Queue tersebut kosong.

```
void init(Queue *queue){
    queue->front = -1;
    queue->rear = -1;
}
```

- ## isEmpty

Untuk memeriksa apakah Queue kosong, periksa apakah nilai dari index `front` dan `rear` berinilai -1 atau tidak. Sedangkan, untuk memeriksa apakah Queue penuh, cukup periksa nilai dari rear apakah sama dengan ukuran array data atau tidak.

```
bool isEmpty(Queue *queue){
    return queue->front == -1 && queue->rear == -1;
}

bool isFull(Queue *queue){
    return queue->rear == MAX - 1;
}
```

- ## enqueue

Enqueue adalah operasi pada Queue untuk menambahkan data baru di belakan antrean. Untuk melakukan Queue, langkah-langkahnya adalah sebagai berikut.

- Periksa apakah Queue penuh atau tidak. Jika Queue tidak penuh, Proses enqueue dapat dilakukan. Sedangkan apabila Queue penuh, keluar dari prosedur karena data baru tidak bisa

ditambahkan pada antrean yang sudah penuh (untuk kasus implementasi pada array statis).

- Periksa apakah Queue kosong atau tidak. Jika kosong, tambahkan nilai index `front` dan `rear` dengan 1, kemudian isikan data baru pada index `rear`.
- Jika tidak kosong, tambahkan index `rear` dengan 1, kemudian isikan data baru pada index `rear` tersebut.

```
void enqueue(Queue *queue, int newValue){
    if(!isFull(queue)){
        if(isEmpty(queue)){
            queue->front++;
            queue->data[++queue->rear] = newValue;
            return;
        }
        queue->data[++queue->rear] = newValue;
        return;
    }
    printf("Queue penuh!\n");
}
```

- **dequeue**

Dequeue merupakan operasi pada Queue untuk menghapus data paling depan pada antrean. Untuk melakukan dequeue, langkah-langkahnya adalah sebagai berikut:

- Periksa apakah Queue kosong atau tidak. Jika Queue tidak kosong, maka proses dequeue dapat dilanjutkan. Apabila kosong, keluar dari prosedur karena tidak ada data yang bisa dihapus dari Queue yang sudah kosong.
- Terdapat pengecualian, yaitu jika hanya ada 1 data dalam array (ditandai dengan nilai `front` dan `queue` sama-sama bernilai 0). Maka buat Queue menjadi dalam kondisi kosong dengan cara membuat index `front` dan `rear` bernilai -1.
- Apabila tidak masuk dalam pengecualian (data lebih dari 1), lakukan penghapusan pada data paling depan dengan cara menumpuk data tersebut dengan data di belakangnya.
- Setelah proses penghapusan, akan ada duplikasi data di akhir array, sehingga mundurkan nilai index `rear` sebanyak 1 kali.

```
void dequeue(Queue *queue){
    if(!isEmpty(queue)){
        if(queue->front == 0 && queue->rear == 0){
            queue->front--;
            queue->rear--;
            return;
        }
        for(int i = queue->front + 1; i < queue->rear + 1; i++){
            queue->data[i - 1] = queue->data[i];
        }
        queue->rear--;
```

```
        return;
    }
    printf("Queue kosong!\n");
}
```

- **front**

Untuk melihat data Queue paling depan, periksa dulu apakah array kosong atau tidak. Apabila tidak kosong, nilai data pada front adalah nilai pada index ke-0 pada array. Apabila kosong, kembalikan nilai -1.

```
int front(Queue *queue){
    return !isEmpty(queue) ? queue->data[queue->front] : -1;
}
```

Implementasi Lengkap Queue

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

#define MAX 10

typedef struct {
    int data[MAX];
    int front;
    int rear;
} Queue;

void init(Queue *queue){
    queue->front = -1;
    queue->rear = -1;
}

bool isEmpty(Queue *queue){
    return queue->front == -1 && queue->rear == -1;
}

bool isFull(Queue *queue){
    return queue->rear == MAX - 1;
}

int size(Queue *queue){
    return queue->rear + 1;
}

int front(Queue *queue){
    return !isEmpty(queue) ? queue->data[queue->front] : -1;
```

```
}

void enqueue(Queue *queue, int newValue){
    if(!isFull(queue)){
        if(isEmpty(queue)){
            queue->front++;
            queue->data[++queue->rear] = newValue;
            return;
        }
        queue->data[++queue->rear] = newValue;
        return;
    }
    printf("Queue penuh!\n");
}

void dequeue(Queue *queue){
    if(!isEmpty(queue)){
        if(queue->front == 0 && queue->rear == 0){
            queue->front--;
            queue->rear--;
            return;
        }
        for(int i = queue->front + 1; i < queue->rear + 1; i++){
            queue->data[i - 1] = queue->data[i];
        }
        queue->rear--;
        return;
    }
    printf("Queue kosong!\n");
}

void printQueue(Queue *queue){
    if(!isEmpty(queue)){
        for(int i = queue->front; i < queue->rear + 1; i++){
            printf("%d ", queue->data[i]);
        }
        return;
    }
    printf("Queue kosong!");
}

int main(){
    Queue queue;
    init(&queue);

    int option;

    while(1){
        printf("-----------------------------------------------------------------\n");
        printf("Queue: ");
        printQueue(&queue);
        printf("\t|\tFront: %d", front(&queue));
        printf("\t|\tSize: %d\n", size(&queue));
```

```
            printf("-----------------------------------------------------------------\n");
            printf("1. Enqueue\n");
            printf("2. Dequeue\n");
            printf("3. Exit\n");
            printf("-----------------------------------------------------------------\n");
            printf("Masukkan pilihan: ");
            scanf("%d", &option);
            if(option == 1){
                int value;
                printf("Masukkan nilai baru: ");
                scanf("%d", &value);
                enqueue(&queue, value);
            }else if(option == 2){
                dequeue(&queue);
            }else if(option == 3){
                break;
            }
            else{
                printf("Opsi tidak valid!\n");
            }
            system("clear"); // gunakan system("cls") pada sistem operasi Windows
        }
        return 0;
    }
```

Soal Latihan

Link Pengumpulan: https://docs.google.com/forms/d/e/1FAIpQLSc8klsj2wF24QfQ8edd-VOVT8JHbc1pcKl8VsFM1S7zUEb1SA/viewform?usp=sf_link

Terdapat antrian pasien di rumah sakit dengan tingkat keparahan bervariasi. Tingkat keparahan 1 – 10 dengan syarat pasien dengan tingkat keparahan <= 5 akan dihandle oleh dokter umum jika > 5 maka kan dihandle oleh dokter spesialis. Implementasikan Queue dalam menyelesaikan masalah ini.

Input Sample 1

```
5 1 4 8 9 7 1 2 9 4
```

Output Sample 1

```
ditangani dokter umum
ditangani dokter umum
ditangani dokter umum
ditangani dokter spesialis
ditangani dokter spesialis
ditangani dokter spesialis
ditangani dokter umum
```

```
ditangani dokter umum
ditangani dokter spesialis
ditangani dokter umum
```

Asisten Praktikum Algoritma Struktur Data Informatika 2023

Pages 8

Clone this wiki locally

```
https://github.com/fzl-22/Algoritma-dan-Struktur-Data_Informatika.wiki.git
```