MAKALAH

TENTANG QUEUE

Disusun oleh :

Nama	: L Hafid Alkhair
NIM	: 2023903430060
Kelas	: TRKJ 1.C
Jurusan	: Teknologi Informasi dan Komputer
Program Studi	: Teknologi Rekayasa Komputer Jaringan
Dosen Pengajar	: Indrawati, SST. MT

JURUSAN TEKNOLOGI INFORMASI KOMPUTER

PRODI TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEUMAWE

TAHUN AJARAN 2023/2024

DAFTAR ISI

BAB I

PEMBAHASAN

A. Pengertian Queue

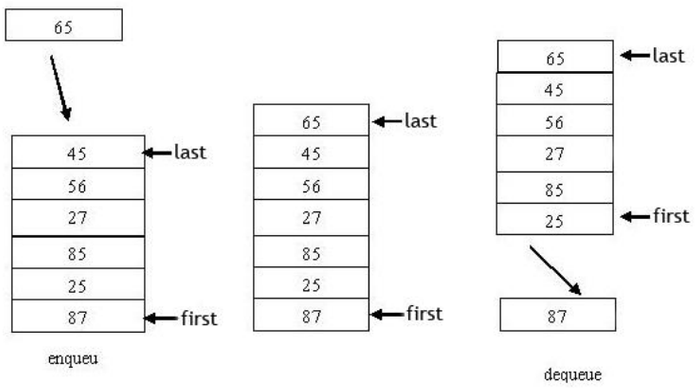
Queue jika diartikan secara harfiah, queue berarti antrian, Queue merupakan suatu struktur data linear. Konsepnya hampir sama dengan Stack, perbedaannya adalah operasi penambahan dan penghapusan pada ujung yang bebeda. Pada Stack atau tumpukan menggunakan prinsip “Masuk terakhir keluar pertama”atau LIFO (Last In First Out), Maka pada Queue atau antrian prinsip yang digunakan adalah “Masuk Pertama Keluar Pertama” atau FIFO (First In First Out). Data-data di dalam antrian dapat bertipe integer, real, record dalam bentuk sederhana atau terstruktur.

Pada saat kita mengantri, setiap kali ada orang yang datang, otomatis orang tersebut akan berada di belakang antrian bukan? Dan siapakah orang yang pertama kali dilayani?. Orang yang pertama kali datang atau orang yang berada di sisi depan antrian. Dalam antrian, ada beberapa poin penting, diantaranya adalah siapa orang yang berada paling depan, siapa yang berada di paling belakang, serta jumlah orang yang mengantri ada berapa.

Nah, queue atau antrian pada struktur data juga sama, hanya saja orang diganti menjadi data/elemen. Maka queue adalah sekumpulan elemen yang jika ada elemen baru yang ditambahkan, maka elemen tersebut akan berada di bagian belakang antrian Jika ada elemen yang harus dihapus atau keluar dari antrian, maka elemen yang keluar adalah elemen yang berada di sisi depan antrian. Atau konsep ini sering juga disebut dengan konsep FIFO (First In First Out).

Gambar 1.1 Gambaran Untuk Queue

Queue (antrian) adalah salah satu list linier dari struktur data yang beroperasi dengan cara First In First Out (FIFO) yaitu elemen pertama yang masuk merupakan elemen yang pertama keluar. Contohnya, ialah dalam sebuah antrian pembelian tiket bagi yang pertama masuk maka dia pulalah yang pertama keluar/selesai. Untuk penyisipan (INSERT) hanya dapat dilakukan pada satu sisi yaitu sisi belakang (REAR), sedangkan untuk penghapusan (REMOVE) pada sisi depan (FRONT) dari list.



Gambar 1.2 penghapusan (REMOVE) pada sisi

depan (FRONT) dari list.

Queue/antrian adalah ordered list dengan penyisipan di satu ujung, sedang penghapusan di ujung lain. Ujung penyisipan biasa disebut rear/tail, sedang ujung penghapusa disebut front/head. Fenomena yang muncul adalah elemen yang lebih dulu disisipkan akan juga lebih dulu diambil. Queue merupakan kasus khusus ordered list. Dengan karakteristik terbatas itu maka kita dapat melakukan optimasi representasi ADT Queue untuk memperoleh kerja paling optimal.

Misalnya Queue Q= (a1,a2,a3…,an), maka:

1. Elemen a1 adalah elemen paling depan
2. Elemen ai adalah diatas elemen ai-1, di mana 1<i<n.
3. Elemen an adalah elemen paling belakang.

Head (Front) menunjuk ke awal antrian Q (elemen terdepan), sedangkan tail (rear) menunjuk akhir antrian Q (elemen paling belakang). Disiplin FIFO pada Queue berimplikasi jika elemen A, B, C, D, E dimasukkan ke Queue, maka penghapusan/ pengambilan elemen akan terjadi dengan urutan A, B, C, D, E.

Sebagai gambaran, cara kerja queue dapat disamakan pada sebuah antrean di suatu loket dimana berlaku prinsip ‘Siapa yang duluan antre dia yang akan pertama kali dilayani‘, sehingga dapat dikatakan prinsip kerja queue sama dengan prinsip sebuah antrean.

B. Karakteristik Queue

Karakteristik penting antrian sebagai berikut:

1. Elemen antrian yaitu item-item data yang terdapat di elemen antrian.
2. Head/Front (elemen terdepan dari antrian).
3. Tail/Tear (elemen terakhir dari antrian).
4. Jumlah elemen pada antrian (count).
5. Status/kondisi antrian.

Ada beberapa kondisi yang bisa kita temukan dalam queue. Kondisi antrian yang menjadi perhatian adalah:

- PenuhBila elemen di antrian mencapai kapasitas maksimum antrian. Pada kondisi ini, tidak mungkin dilakukan penambahan ke antrian. Penambahan elemen menyebabkan kondisi kesalahan Overflow.
- Kosong Bila tidak ada elemen di antrian. Pada kondisi ini, tidak mungkin dilakukan pengambilan elemen dari antrian. Pengambilan elemen menyebabkan kondisi kesalahan Underflow.

C. Operasi-Operasi Pada Queue

1. Create()

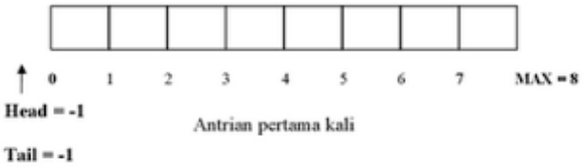
Untuk menciptakan dan menginisialisasi Queue dengan cara membuat Head dan Tail = -1

DEKLARASI QUEUE

```
#define MAX 8
typedef   struct{
                    int data[MAX];
                    int head;
                    int tail;
                } Queue;

Queue antrian;
```

Gambar 1.3 Deklarasi Queue

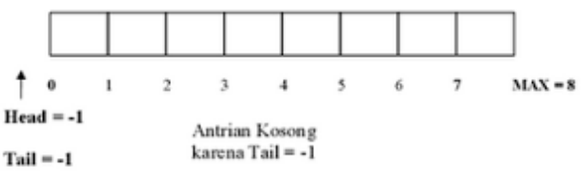



```
void Create(){
     antrian.head=antrian.tail=-1;
}
```

Gambar 1.4 Deklarasi Queue operasi create ( )

2. IsEmpty

Digunakan untuk memeriksa apakah Antrian sudah penuh atau belum Dengan cara memeriksa nilai Tail, jika Tail = -1 maka empty Kita tidak memeriksa Head, karena Head adalah tanda untuk kepala antrian (elemen pertama dalam antrian) yang tidak akan berubah-ubah Pergerakan pada Antrian terjadi dengan penambahan elemen Antrian kebelakang, yaitu menggunakan nilai Tail.

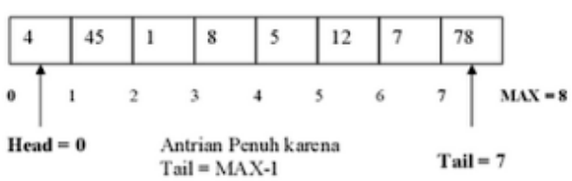



```
int IsEmpty(){
     if(antrian.tail==-1)
          return 1;
     else
          return 0;
}
```

Gambar 1.5 Deklarasi Queue operasi IsEmpty

3. IsFull

Untuk mengecek apakah Antrian sudah penuh atau belum Dengan cara mengecek nilai Tail, jika Tail >= MAX-1 (karena MAX-1 adalah batas elemen array pada C) berarti sudah penuh

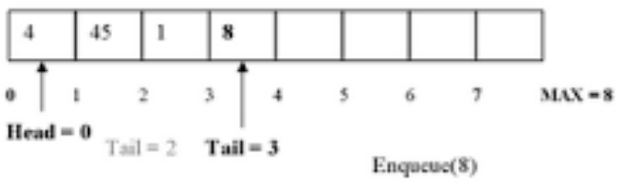



```
int IsFull(){
     if(antrian.tail==MAX-1) return 1;
     else return 0;
}
```

Gambar 1.6 Deklarasi Queue operasi IsFull

4. Enqueue

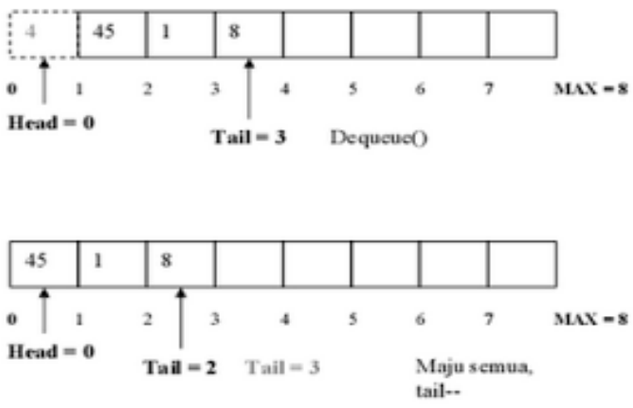
Untuk menambahkan elemen ke dalam Antrian, penambahan elemen selalu ditambahkan di elemen paling belakang Penambahan elemen selalu menggerakan variabel Tail dengan cara increment counter Tail terlebih dahulu




```
void Enqueue(int data){
     if(IsEmpty()==1){
          antrian.head=antrian.tail=0;
          antrian.data[antrian.tail]=data;
          printf("%d masuk!",antrian.data[antrian.tail]);
     } else
     if(IsFull()==0){
          antrian.tail++;
          antrian.data[antrian.tail]=data;
          printf("%d masuk!",antrian.data[antrian.tail]);
     }
}
```

Gambar 1.7 Deklarasi Queue operasi Enquene

5. Dequeue()



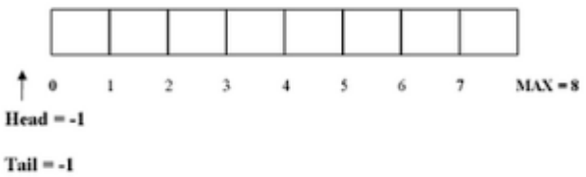

```
int Dequeue(){
     int i;
     int e = antrian.data[antrian.head];
     for(i=antrian.head;i<=antrian.tail-1;i++){
          antrian.data[i] = antrian.data[i+1];
     }
     antrian.tail--;
     return e;
}
```

Gambar 1.8 Deklarasi Queue operasi Dequene

Digunakan untuk menghapus elemen terdepan/pertama (head) dari Antrian Dengan cara menggeser semua elemen antrian kedepan dan mengurangi Tail dgn 1Penggeseran dilakukan dengan menggunakan looping.

6. Clear()

Untuk menghapus elemen-elemen Antrian dengan cara membuat Tail dan Head = -1 Penghapusan elemen-elemen Antrian sebenarnya tidak menghapus arraynya, namun hanya mengeset indeks pengaksesan-nya ke nilai -1 sehingga elemen-elemen Antrian tidak lagi terbaca




```
void Clear(){
     antrian.head=antrian.tail=-1;
     printf("data clear");
}
```

Gambar 1.9 Deklarasi Queue operasi clear

7. Tampil()

Untuk menampilkan nilai-nilai elemen Antrian Menggunakan looping dari head sampai tail

```
void Tampil(){
     if(IsEmpty()==0){
          for(int i=antrian.head;i<=antrian.tail;i++){
               printf("%d ",antrian.data[i]);
          }
     }else printf("data kosong!\n");
}
```

Gambar 1.10 Deklarasi Queue operasi Tampil

D. Implementasi Queue

1. Struktur Data Queue

Dalam C, sebuah Queue dapat diimplementasikan menggunakan struktur data yang sederhana. Struktur data ini biasanya terdiri dari elemen-elemen atau simpul-simpul yang saling terhubung. Setiap elemen menyimpan data dan alamat (pointer) ke elemen berikutnya.

```c
struct Node {
    int data;
    struct Node* next;
};
```

Gambar 1.11 Struktur Data Queue

2. Enqueue (Penambahan elemen)

Operasi Enqueue digunakan untuk menambahkan elemen baru ke dalam Queue. Elemen ini ditambahkan pada ujung belakang atau rear dari Queue. Sebuah fungsi Enqueue sederhana dapat ditulis sebagai berikut:

```c
void enqueue(struct Node** rear, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;

    if (*rear == NULL) {
        // Jika Queue kosong, elemen baru menjadi elemen pertama
        *rear = newNode;
    } else {
        // Jika Queue tidak kosong, tambahkan elemen baru pada ujung belakang
        (*rear)->next = newNode;
        *rear = newNode;
    }
}
```

Gambar 1.12 Penambahan elemen

3. Dequeue (Penghapusan elemen)

Operasi Dequeue digunakan untuk menghapus elemen dari depan atau front Queue. Fungsi Dequeue dapat diimplementasikan sebagai berikut:

```c
int dequeue(struct Node** front) {
    if (*front == NULL) {
        // Jika Queue kosong, kembalikan nilai khusus (misalnya, -1)
        return -1;
    } else {
        struct Node* temp = *front;
        int data = temp->data;
        *front = (*front)->next;

        // Bebaskan memori yang digunakan oleh elemen yang dihapus
        free(temp);

        return data;
    }
}
```

Gambar 1.13 Penghapusan Elemen

4. Front dan Rear dalam Queue

Front adalah elemen pertama dalam Queue, sedangkan Rear adalah elemen terakhir. Kedua pointer ini digunakan untuk memanipulasi Queue.

Misalnya, saat menambahkan elemen baru, pointer Rear diperbarui; saat menghapus elemen, pointer Front diperbarui.

5. Contoh Kode Implementasi Queue

Berikut adalah contoh penggunaan fungsi-fungsi di atas untuk membuat dan mengoperasikan Queue:

```c
int main() {
    struct Node* front = NULL;
    struct Node* rear = NULL;

    // Enqueue beberapa elemen
    enqueue(&rear, 10);
    enqueue(&rear, 20);
    enqueue(&rear, 30);

    // Dequeue dan cetak elemen pertama
    int removedItem = dequeue(&front);
    printf("Elemen pertama yang dihapus: %d\n", removedItem);

    // Cetak elemen-elemen yang tersisa dalam Queue
    printf("Elemen-elemen dalam Queue: ");
    while (front != NULL) {
        printf("%d ", front->data);
        front = front->next;
    }

    return 0;
}
```

Gambar 1.14 Kode Implementasi Queue

Contoh di atas menciptakan sebuah Queue, menambahkan beberapa elemen menggunakan Enqueue, menghapus elemen pertama menggunakan Dequeue, dan mencetak elemen-elemen yang tersisa dalam Queue. Pastikan untuk menyesuaikan dan mengintegrasikan kode ini sesuai dengan kebutuhan proyek Anda.

E. Jenis Jenis Queue

1. Queue Linier

Queue linier adalah tipe paling sederhana dari Queue, di mana elemen-elemen disusun dalam urutan linier atau sejajar. Enqueue dilakukan pada ujung belakang, dan Dequeue dilakukan pada ujung depan. Implementasi sederhana dijelaskan pada poin 2.

2. **Circular Queue**

Circular Queue adalah modifikasi dari Queue linier di mana ujung belakang dan depan terhubung satu sama lain membentuk lingkaran. Hal ini memungkinkan penggunaan ulang ruang penyimpanan dan pengoptimalan operasi Enqueue dan Dequeue. Pada implementasinya, ketika Rear mencapai akhir array, elemen baru dapat ditempatkan pada awal array jika ada ruang kosong.

3. **Priority Queue**

Priority Queue adalah jenis Queue di mana setiap elemen memiliki prioritas yang terkait. Elemen dengan prioritas tertinggi diberikan prioritas paling tinggi untuk di-Dequeue terlebih dahulu. Implementasinya melibatkan penyimpanan elemen-elemen beserta prioritasnya, dan operasi Dequeue akan mengembalikan elemen dengan prioritas tertinggi.

4. **Double-ended Queue (Deque)**

Double-ended Queue, atau deque, memungkinkan Enqueue dan Dequeue dilakukan pada kedua ujung (front dan rear) dari Queue. Ini memberikan fleksibilitas tambahan dalam memanipulasi data, seperti menambahkan atau menghapus elemen dari kedua ujung.

Pemilihan jenis Queue tergantung pada kebutuhan spesifik aplikasi atau algoritma yang akan diimplementasikan. Misalnya, Priority Queue cocok digunakan dalam situasi di mana elemen-elemen memiliki tingkat kepentingan yang berbeda-beda, sementara Circular Queue dapat digunakan ketika alokasi memori perlu dioptimalkan dengan penggunaan ulang ruang penyimpanan.

F. Operasi Umum Pada Queue

1. **Inisialisasi Queue**

Sebelum Queue dapat digunakan, perlu dilakukan inisialisasi untuk mengatur keadaan awal. Inisialisasi melibatkan penyiapan pointer Front dan Rear menjadi NULL atau nilai yang menandakan bahwa Queue masih kosong.

```
1  struct Node* front = NULL;
2  struct Node* rear = NULL;
3
```

Gambar 1.15 Inisialisasi Queue

2. Cek apakah Queue Kosong atau Penuh

Sebelum melakukan operasi Dequeue atau Enqueue, seringkali diperlukan pemeriksaan apakah Queue kosong atau penuh. Pemeriksaan ini membantu mencegah kesalahan saat mencoba mengakses elemen dalam kondisi yang tidak valid.

```
int isEmpty(struct Node* front) {
    return (front == NULL);
}

int isFull(struct Node* rear) {
    // Tergantung pada implementasi, misalnya jika menggunakan alokasi dinamis
    // Anda perlu memeriksa keberhasilan alokasi memori.
    return 0;
}
```

Gambar 1.16 Mengecek Queue

3. Menghitung Jumlah Elemen dalam Queue

Untuk melihat berapa banyak elemen yang ada dalam Queue, Anda dapat melakukan iterasi melalui elemen-elemen dan menghitungnya.

```
int countElements(struct Node* front) {
    int count = 0;
    struct Node* temp = front;

    while (temp != NULL) {
        count++;
        temp = temp->next;
    }

    return count;
}
```

Gambar 1.17 Menghitung Jumlah Elemen dalam Queue

4. Mengakses Elemen pada Front dan Rear

Penting untuk dapat mengakses elemen pada Front dan Rear untuk melihat nilai-nilai ini tanpa mengubah struktur Queue.

```
int getFront(struct Node* front) {
    if (front != NULL) {
        return front->data;
    } else {
        // Handle kasus ketika Queue kosong
        return -1; // atau nilai khusus lainnya
    }
}

int getRear(struct Node* rear) {
    if (rear != NULL) {
        return rear->data;
    } else {
        // Handle kasus ketika Queue kosong
        return -1; // atau nilai khusus lainnya
    }
}
```

Gambar 1.18 Mengakses Elemen pada Front dan Rear

Operasi-operasi ini membantu memastikan bahwa manipulasi Queue dapat dilakukan dengan aman dan efisien, dan memungkinkan pengguna untuk mendapatkan informasi tentang keadaan dan isi Queue tanpa mengganggu strukturnya.

G. Penggunaan Queue dalam Algoritma dan Pemecahan Masalah.

1. Antrian dalam Algoritma BFS (Breadth-First Search)

Breadth-First Search (BFS) adalah algoritma pencarian yang digunakan untuk menjelajahi atau mencari jalur pada struktur data graf atau pohon secara lebar. Algoritma ini memiliki berbagai aplikasi, dan salah satu elemen kunci dalam implementasinya adalah penggunaan antrian (queue). Dalam konteks BFS, antrian digunakan untuk menyimpan simpul-simpul yang akan dieksplorasi pada tingkat selanjutnya. Pada bagian ini, kita akan menjelaskan secara rinci bagaimana antrian digunakan dalam algoritma BFS.

a) Prinsip Dasar BFS:

   BFS bekerja dengan cara menjelajahi atau mengunjungi simpul-simpul pada tingkat yang sama sebelum beralih ke tingkat yang lebih tinggi. Konsep ini dikenal sebagai pencarian secara lebar atau level-order traversal.

b) Struktur Data Antrian:

   Dalam implementasi BFS, antrian digunakan untuk menyimpan simpul-simpul yang menunggu untuk dieksplorasi pada tingkat selanjutnya. Antrian mengikuti prinsip First-In-First-Out (FIFO), di mana simpul yang pertama kali dimasukkan ke dalam antrian akan dieksplorasi terlebih dahulu.

c) Inisialisasi Antrian:

   Sebelum memulai proses BFS, antrian perlu diinisialisasi. Antrian kosong pada awalnya. Simpul awal (biasanya disebut sebagai simpul sumber) dimasukkan ke dalam antrian untuk memulai proses BFS.

d) Proses Eksplorasi:

   Selama proses eksplorasi, simpul yang ada dalam antrian dieksplorasi satu per satu. Setiap kali simpul dieksplorasi, semua tetangganya yang belum dieksplorasi dimasukkan ke dalam antrian. Hal ini dilakukan untuk memastikan bahwa simpul pada tingkat yang lebih rendah dieksplorasi sebelum beralih ke tingkat yang lebih tinggi.

e) Pencatatan dan Penandaan:

   Setiap kali simpul dieksplorasi, simpul tersebut dicatat atau ditandai untuk menghindari eksplorasi ganda. Penandaan dapat berupa penanda pada simpul atau struktur data terpisah untuk mencatat simpul-simpul yang sudah dieksplorasi.

f) Penghentian Algoritma:

   Algoritma BFS berhenti ketika antrian kosong, yang berarti semua simpul yang dapat dijangkau dari simpul sumber sudah dieksplorasi. Hasil dari algoritma ini adalah peta jarak terpendek dari simpul sumber ke semua simpul lainnya dalam graf.

g) Contoh Implementasi:

   Sebagai contoh, pertimbangkan graf berarah atau tidak berarah dengan simpul-simpul dan sambungan-sambungan antara

mereka. Antrian akan menyimpan simpul-simpul yang akan dieksplorasi selama iterasi algoritma BFS.

h) Aplikasi Lain:

Selain pencarian jalur terpendek, BFS juga digunakan dalam berbagai aplikasi seperti deteksi siklus dalam graf, pengurutan topologis, dan pengembangan permainan.

i) Keunggulan dan Kelemahan:

BFS memiliki keunggulan dalam menemukan jalur terpendek, tetapi dapat memerlukan ruang memori yang besar terutama pada graf yang sangat besar.

Breadth-First Search (BFS) adalah algoritma yang sangat berguna dalam menjelajahi graf dan pohon secara lebar. Dengan menggunakan konsep antrian, BFS memberikan cara efisien untuk menemukan jalur terpendek dan menjelajahi struktur data berhierarki secara sistematis.

2. Penjadwalan Tugas dengan Menggunakan Queue

Penjadwalan tugas (Task Scheduling) adalah aspek penting dalam pengembangan perangkat lunak dan manajemen sistem komputasi. Penjadwalan tugas melibatkan alokasi sumber daya komputasi untuk menyelesaikan tugas atau pekerjaan yang beragam. Pada bagian ini, kita akan menjelaskan secara rinci tentang bagaimana penjadwalan tugas dapat dilakukan dengan menggunakan struktur data antrian (queue) dalam konteks pengembangan perangkat lunak.

a) Konsep Dasar Penjadwalan Tugas:

Penjadwalan tugas mencakup pengaturan dan alokasi sumber daya komputasi untuk menyelesaikan tugas atau pekerjaan. Hal ini mencakup pemilihan tugas yang akan dieksekusi, pengaturan prioritas, dan penentuan urutan eksekusi.

b) Penggunaan Antrian dalam Penjadwalan:

Antrian adalah struktur data yang sangat cocok untuk mengelola penjadwalan tugas. Antrian mengikuti prinsip First-Come-First-Serve (FCFS), yang berarti tugas yang datang lebih awal akan dieksekusi lebih dulu. Ini menciptakan sistem antrian yang adil dan transparan.

c) Inisialisasi Antrian Tugas:

Sebelum memulai penjadwalan tugas, antrian perlu diinisialisasi. Antrian ini dapat disusun untuk menyimpan tugas-tugas yang akan dieksekusi sesuai dengan urutan kedatangan mereka.

d) Enqueue untuk Menambahkan Tugas:

   Proses menambahkan tugas ke dalam antrian disebut Enqueue. Saat tugas baru muncul, tugas tersebut dimasukkan ke dalam antrian di bagian belakang. Ini menciptakan urutan eksekusi berdasarkan waktu kedatangan.

e) Dequeue untuk Menjalankan Tugas:

   Proses eksekusi tugas dilakukan dengan menggunakan operasi Dequeue. Tugas yang berada di depan antrian diambil dan dieksekusi. Setelah tugas selesai, tugas selanjutnya diambil dari depan antrian.

f) Prioritas dalam Antrian:

   Antrian juga dapat digunakan untuk mengimplementasikan penjadwalan tugas berdasarkan prioritas. Setiap tugas memiliki tingkat prioritas tertentu, dan tugas dengan prioritas lebih tinggi akan dieksekusi lebih dulu. Dalam konteks antrian, prioritas ini dapat disimpan bersama dengan tugas.

g) Manajemen Waktu Eksekusi:

   Antrian membantu dalam manajemen waktu eksekusi. Dengan mengikuti prinsip FCFS, antrian memastikan bahwa tugas-tugas dieksekusi secara adil dan sesuai dengan urutan kedatangan. Ini dapat membantu mencegah situasi kelaparan tugas (starvation).

h) Penanganan Tugas Darurat atau Segera:

   Antrian dapat diadaptasi untuk menangani tugas darurat atau tugas yang memerlukan eksekusi segera. Tugas-tugas ini dapat ditempatkan di depan antrian atau memiliki prioritas tertinggi.

i) Siklus Penjadwalan:

   Antrian digunakan untuk menciptakan siklus penjadwalan, di mana tugas-tugas dieksekusi secara berulang sesuai dengan kondisi atau kebutuhan sistem.

j) Penanganan Kesalahan dan Retry:

   Antrian juga berguna dalam menangani kesalahan atau situasi yang memerlukan retry. Jika tugas gagal dieksekusi, tugas

tersebut dapat kembali dimasukkan ke dalam antrian untuk dicoba kembali.

k) Sistem Terdistribusi:

Dalam sistem terdistribusi, antrian dapat digunakan untuk mengelola dan mengkoordinasikan tugas-tugas yang dikerjakan di berbagai node atau komputer.

l) Keuntungan dan Tantangan:

Keuntungan penggunaan antrian dalam penjadwalan tugas termasuk keadilan, transparansi, dan manajemen waktu yang baik. Namun, tantangan dapat muncul dalam situasi di mana prioritas tugas sangat beragam atau di mana ada kebutuhan untuk respons segera.

m) Kesimpulan:

Penjadwalan tugas dengan menggunakan antrian memberikan pendekatan sistematis untuk mengatur dan mengelola eksekusi tugas dalam suatu sistem. Dengan memahami konsep ini, pengembang dapat merancang solusi penjadwalan yang efektif dan efisien sesuai dengan kebutuhan aplikasi atau sistem yang sedang dikembangkan.

3. Simulasi Antrian dalam Sistem

Simulasi antrian dalam sistem adalah pendekatan yang digunakan untuk memodelkan dan menganalisis perilaku antrian di berbagai konteks, seperti sistem layanan pelanggan, proses manufaktur, atau jaringan komputer. Dalam simulasi ini, struktur data antrian (queue) sangat penting untuk merepresentasikan entitas yang menunggu untuk diproses. Dalam bagian ini, kita akan menjelaskan secara rinci bagaimana simulasi antrian dalam sistem bekerja dan mengapa antrian menjadi kunci dalam proses tersebut.

Simulasi antrian membantu dalam memahami dan memprediksi perilaku sistem dengan mereplikasi kondisi sebenarnya. Dalam banyak kasus, sistem antrian dapat ditemukan di berbagai lingkungan, seperti kasir, pusat panggilan, atau server komputer.

a) Penggunaan Antrian dalam Model Simulasi:

   Antrian digunakan untuk menyimpan entitas yang sedang menunggu untuk dijalankan atau diproses. Ketika suatu entitas masuk ke dalam sistem, ia dimasukkan ke dalam antrian dan kemudian diambil berdasarkan aturan tertentu untuk diproses.

b) Strategi Antrian:

   Simulasi antrian sering melibatkan penerapan berbagai strategi antrian, seperti First-Come-First-Serve (FCFS), Prioritas, atau Shortest Job Next (SJN). Pilihan strategi antrian ini dapat mempengaruhi kinerja dan efisiensi sistem.

c) Simulasi Peristiwa Diskrit:

   Simulasi antrian umumnya diimplementasikan sebagai simulasi peristiwa diskrit, di mana perubahan keadaan sistem terjadi pada titik waktu tertentu. Setiap peristiwa, seperti kedatangan entitas baru atau penyelesaian suatu tugas, dapat menyebabkan perubahan dalam antrian.

d) Pengamatan dan Pengukuran:

   Simulasi antrian memungkinkan pengamatan dan pengukuran berbagai metrik kinerja, seperti waktu menunggu rata-rata, tingkat utilitas sistem, atau jumlah entitas yang menunggu. Ini membantu dalam evaluasi dan perbaikan desain sistem.

e) Model Antrian M/M/1:

   Salah satu model antrian yang umum digunakan dalam simulasi adalah model antrian M/M/1, yang menggambarkan sistem satu server dengan tingkat kedatangan dan tingkat pelayanan tertentu. Model ini memberikan landasan untuk menganalisis kinerja antrian dalam berbagai skenario.

f) Aplikasi dalam Pelayanan Pelanggan:

   Simulasi antrian sering diterapkan dalam industri layanan pelanggan, seperti pusat panggilan atau layanan konsumen. Dengan menggunakan antrian, perusahaan dapat mengoptimalkan waktu pelayanan dan meningkatkan kepuasan pelanggan.

g) Aplikasi dalam Jaringan Komputer:

Dalam jaringan komputer, simulasi antrian dapat digunakan untuk memodelkan lalu lintas paket data. Antrian digunakan untuk merepresentasikan antrean paket yang menunggu untuk dikirim atau diterima oleh perangkat jaringan.

h) Keuntungan Penggunaan Simulasi Antrian:

Penggunaan simulasi antrian memberikan kemampuan untuk menguji dan mengoptimalkan desain sistem tanpa perlu menerapkan perubahan langsung pada sistem yang sebenarnya. Ini membantu dalam pengambilan keputusan yang informasional dan berbasis bukti.

i) Tantangan dalam Simulasi Antrian:

Meskipun simulasi antrian memberikan manfaat besar, ada beberapa tantangan, seperti pemilihan model yang tepat, penentuan parameter simulasi yang akurat, dan interpretasi hasil simulasi yang kompleks.

j) Kesimpulan:

Simulasi antrian dalam sistem adalah alat penting dalam analisis dan perancangan sistem. Dengan menggunakan struktur data antrian, kita dapat merepresentasikan dan menganalisis berbagai skenario antrian, memberikan pemahaman yang lebih baik tentang kinerja sistem dan membantu dalam pengambilan keputusan yang efektif.

4. Pemrosesan Data Streaming

Pemrosesan data streaming adalah paradigma pengolahan data yang berkaitan dengan aliran data yang kontinu, seperti data sensor, log peristiwa, atau aliran transaksi. Dalam konteks pemrosesan data streaming, antrian (queue) sering digunakan untuk mengelola dan mentransmisikan data secara efisien. Pada bagian ini, kita akan menjelaskan bagaimana antrian berperan dalam pemrosesan data streaming dan mengapa struktur data ini menjadi kunci dalam mengelola aliran data yang tidak pernah berhenti.

a) Konsep Dasar Pemrosesan Data Streaming:

   Pemrosesan data streaming berkaitan dengan pengolahan data secara real-time, di mana data dihasilkan dan diproses secara terus-menerus. Paradigma ini berbeda dengan pemrosesan data batch, di mana data diproses dalam kelompok tertentu.

b) Struktur Data Antrian dalam Streaming:

   Dalam pemrosesan data streaming, antrian digunakan untuk menyimpan dan mengelola data yang sedang mengalir. Antrian membantu dalam menyimpan data sementara sebelum data tersebut diproses atau disalurkan ke sistem atau aplikasi yang sesuai.

c) Penggunaan Antrian dalam Buffering Data:

   Antrian berfungsi sebagai alat buffering untuk menangani fluktuasi dalam laju produksi dan konsumsi data. Dengan adanya antrian, sistem dapat menangani sementara ketidakseimbangan antara sumber data dan sistem yang memproses data.

d) Operasi Enqueue dan Dequeue:

   Proses enqueuing (penambahan data ke dalam antrian) dan dequeuing (pengambilan data dari antrian) menjadi kunci dalam mengelola aliran data streaming. Data yang baru dihasilkan oleh sumber dapat dienqueue ke dalam antrian, sementara sistem secara bersamaan dapat melakukan dequeuing untuk memproses data.

e) Waktu-Nyata dan Latensi Rendah:

   Dalam pemrosesan data streaming, seringkali diperlukan respons waktu nyata dan latensi rendah. Antrian membantu mengoptimalkan waktu respons dengan menyediakan cara untuk menyimpan dan mengakses data secara efisien.

f) Pemodelan Aliran Data:

   Antrian digunakan untuk memodelkan aliran data yang dapat berubah secara dinamis. Ketika data baru tiba, itu dienqueue, dan data yang sudah diolah dapat didequeued. Ini menciptakan aliran data yang terorganisir.

g) Sinkronisasi dan Koordinasi:

Dalam sistem yang melibatkan beberapa sumber data atau modul pemrosesan, antrian membantu dalam sinkronisasi dan koordinasi. Sumber data dapat menyampaikan data ke antrian tanpa harus menunggu proses pemrosesan selesai.

h) Model Pub/Sub (Publisher-Subscriber):

Antrian sering digunakan dalam model publikasi-langganan, di mana sumber data (publisher) mengirimkan data ke antrian, dan sistem atau aplikasi (subscriber) dapat berlangganan untuk menerima dan memproses data tersebut.

i) Aplikasi dalam Internet of Things (IoT):

Pemrosesan data streaming sangat relevan dalam konteks Internet of Things (IoT), di mana sensor-sensor menghasilkan data secara kontinu. Antrian memainkan peran penting dalam menyimpan dan mengarahkan data sensor ke sistem atau aplikasi yang tepat.

j) Integrasi dengan Sistem Big Data:

Dalam ekosistem big data, pemrosesan data streaming dapat diintegrasikan dengan sistem penyimpanan data besar seperti Apache Kafka atau Apache Flink. Antrian membantu menyusun dan mentransmisikan data antar komponen sistem ini.

k) Pengelolaan Beban Kerja (Workload):

Antrian membantu dalam pengelolaan beban kerja dengan menyediakan mekanisme untuk mengatur dan mendistribusikan data secara efisien. Hal ini memungkinkan sistem untuk memproses data sesuai dengan kapasitasnya.

l) Keamanan dan Tahan Kesalahan:

Antrian juga dapat berperan dalam mengatasi masalah keamanan dan tahan kesalahan dengan menyediakan lapisan buffer untuk menangani sementara gangguan atau kegagalan sistem.

m) Kesimpulan:

Pemrosesan data streaming menjadi semakin penting dalam era di mana data dihasilkan secara kontinu. Dengan memanfaatkan struktur data antrian, sistem dapat mengelola aliran data dengan efisien, mengoptimalkan latensi, dan menyediakan respons

waktu nyata untuk aplikasi yang membutuhkan pemrosesan data dalam waktu singkat. Antrian menjadi fondasi yang krusial dalam membangun sistem pemrosesan data streaming yang efektif dan handal.

5. Algoritma Shortest Job First (SJF)

Shortest Job First (SJF) adalah salah satu algoritma penjadwalan proses yang diterapkan dalam sistem operasi. Tujuannya adalah untuk mengurutkan dan menjadwalkan proses berdasarkan waktu eksekusi yang paling singkat. Dalam tulisan ini, kita akan menjelaskan secara rinci konsep dan implementasi algoritma SJF serta penerapannya dalam penjadwalan proses.

Algoritma Shortest Job First (SJF) merupakan salah satu pendekatan yang digunakan dalam penjadwalan proses untuk meminimalkan waktu tunggu dan waktu respons sistem. Dalam SJF, proses dengan waktu eksekusi terpendek diberikan prioritas lebih tinggi.

Konsep dasar SJF adalah memberikan prioritas eksekusi kepada proses dengan waktu eksekusi yang lebih pendek. Ide utamanya adalah untuk menjalankan proses yang membutuhkan waktu paling singkat terlebih dahulu, sehingga dapat meningkatkan throughput sistem dan mengurangi waktu rata-rata tunggu.

a) Implementasi Non-Preemptive:
   SJF dapat diimplementasikan dalam mode preemptive (memotong eksekusi proses yang sedang berjalan) atau non-preemptive (menunggu proses selesai sebelum menjalankan yang berikutnya). Fokus utama pada bagian ini adalah SJF non-preemptive.
b) Penjadwalan Proses Non-Preemptive:
   Dalam SJF non-preemptive, ketika suatu proses dimulai, ia akan dijalankan hingga selesai sebelum proses berikutnya dimulai. Proses diperlakukan seperti antrian, dan proses dengan waktu eksekusi terpendek akan diambil dari depan antrian.
c) Waktu Tunggu dan Waktu Pelayanan:
   SJF bertujuan untuk meminimalkan waktu tunggu, yang merupakan total waktu yang dihabiskan oleh proses di dalam antrian sebelum dapat dieksekusi. Waktu pelayanan adalah total waktu yang diperlukan untuk mengeksekusi suatu proses.

d) Implementasi dengan Algoritma Penjadwalan:

   Untuk mengimplementasikan SJF, sistem perlu memiliki algoritma penjadwalan yang dapat memilih proses dengan waktu eksekusi terpendek. Ini melibatkan pemantauan waktu eksekusi setiap proses dan pengambilan keputusan yang cerdas.

e) Algoritma Penjadwalan SJF:

   Algoritma SJF dapat dijelaskan dengan langkah-langkah sebagai berikut:

   - Sort semua proses berdasarkan waktu eksekusi.
   - Ambil proses dengan waktu eksekusi terpendek dari depan antrian.
   - Jalankan proses tersebut.
   - Ulangi langkah-langkah di atas hingga semua proses selesai.

f) Keuntungan Algoritma SJF:

   Keuntungan utama SJF adalah efisiensi dalam meminimalkan waktu eksekusi. Jika proses dengan waktu eksekusi terpendek dijalankan terlebih dahulu, ini dapat mengurangi waktu rata-rata tunggu dan meningkatkan produktivitas sistem.

g) Tantangan dan Kelemahan SJF:

   Tantangan utama SJF adalah sulitnya memprediksi waktu eksekusi sebenarnya dari suatu proses. Jika estimasi waktu eksekusi tidak akurat, proses mungkin harus menunggu lebih lama dari yang diperkirakan.

h) Penerapan dalam Lingkungan yang Dinamis:

   SJF dapat diterapkan dalam lingkungan yang dinamis, di mana proses masuk dan keluar secara terus-menerus. Sistem perlu mengadaptasi algoritma penjadwalan untuk mengakomodasi perubahan dinamis ini.

i) Penerapan dalam Sistem Operasi:

   Banyak sistem operasi modern menggunakan variasi dari SJF, seperti Shortest Remaining Time First (SRTF), yang merupakan versi preemptive dari SJF.

j) Kesimpulan:

Algoritma Shortest Job First (SJF) adalah pendekatan efisien dalam penjadwalan proses dengan memprioritaskan eksekusi proses dengan waktu terpendek. Meskipun memiliki tantangan dalam estimasi waktu eksekusi, SJF tetap menjadi pilihan yang baik untuk mengoptimalkan waktu tunggu dan meningkatkan produktivitas dalam berbagai lingkungan sistem operasi.

H. Macam-Macam Queue dan Repsesentasinya

1. Queue dengan Linear Array

Linear array adalah suatu array yang dibuat seakan-akan merupakan suatu garis lurus dengan satu pintu masuk dan satu pintu keluar. Berikut ini diberikan deklarasi kelas QueueLinear sebagai implementasi dari Queue menggunakan linear array. Dalam prakteknya, kita dapat menggantinya sesuai dengan kebutuhan kita. Data dikses dengan field data, sedangkan indeks item pertama dan terakhir disimpan dalam variabel Head dan Tail. Konstruktor akan menginisialisasi nilai Head dan Tail dengan -1 untuk menunjukkan bahwa antrian masih kosong dan mengalokasikan data sebanyak MAX_QUEUE yang ditunjuk oleh data. Destruktor akan mengosongkan antrian kembali dan mendealokasikan memori yang digunakan oleh antrian.

hapus elemen 1 2 …….. ke – n sisip elemen

head tail

Di bawah ini diperlihatkan suatu queue yang akan menempati N elemen array memori, serta cara pengurangan (delete) dan penambahan (added) elemen pada queue tersebut.

REMOVE (Q)

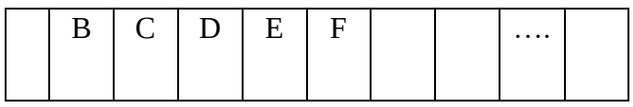
	B	C	D					….	

Head : 2

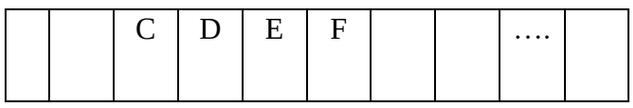
Tail : 4

1 2 3 4 5 6 7 ..... N

INSERT(INSERT(E),F)



REMOVE(Q)



Dapat dilihat bahwa setiap terjadi penghapusan elemen pada queue nilai (index) dari Head bertambah satu (1) ; dapat ditulis HEAD = HEAD+1. Begitu pula bila terjadi penambahan elemen pada queue nilai (index) Tail bertambah satu (1); dapat ditulis TAIL = TAIL + 1.

Akan terjadi ketidakefisienan bila penambahan elemen sudah pada posisi index N (Tail = N) maka tidak dapat lagi dilakukan penambahan elemen, sedangkan dilokasi memori yang lain (nilai di bawah N) masih terdapat memori array yang kosong. Untuk mengatasi hal tersebut maka kita bayangkan bahwa memori dari queue tersebut berbentuk melingkar dimana kedua ujungnya saling bertemu atau disebut juga dengan Circular Queue.

Operasi-Operasi Queue dengan Linear Array :

a) IsEmpty

   Fungsi IsEmpty berguna untuk mengecek apakah queue sudah terisi atau masih kosong. hal ini dilakukan dengan mengecek apakah tail bernilai -1 atau tidak. Nilai -1 menandakan bahwa queue masih kosong.

b) IsFull Fungsi

   IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah nilai tail sudah sama dengan jumlah maksimal queue. Jika nilai keduanya sama, berarti antrian sudah penuh.

c) EnQueue

Fungsi EnQueue berguna untuk memasukkan sebuah elemen dalam sebuah antrian.

d) DeQueue

Fungsi DeQueue berfungsi untuk mengambil sebuah elemen dari antrian. Operasi ini sering disebut juga serve. Hal ini dilakukan dengan cara memindahkan sejauh satu langkah ke posisi di depannya sehingga otomatis elemen yang paling depan akan tertimpa dengan elemen yang terletak di belakangnya.

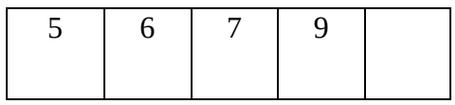
e) Clear Fungsi

Clear berguna untuk menghapus semua lemen dalam queue dengan jalan mengeluarkan semua elemen tersebut satu per satu hingga queue kosong dengan memanfaatkan fungsi DeQueue.

1. Queue dengan Circular Array

Circular array adalah suatu array yang dibuat seakan-akan merupakan sebuah lingkaran dengan titik awal (head) dan titik akhir (tail) saling bersebelahan jika array tersebut masih kosong. Jika menggunakan array untuk queue seperti di atas, maka ketika ada proses pengambilan (dequeue) ada proses pergeseran data. Proses pergeseran data ini pasti memerlukan waktu apalagi jika elemen queue-nya banyak. Oleh karena itu solusi agar proses pergeseran dihilangkan adalah dengan metode circular array. Kelebihan jenis ini adalah alokasi penyimpanan data yang optimal dan dinamis. Hal ini disebabkan penambahan maupun pengurangan data/ item antrian yang baru selalu menempati pos kosong yang disediakan sistem.

Queue dengan circular array dapat dibayangkan sebagai berikut:



Front = 1

Tail = 4

1 2 3 4 5

Aturan-aturan dalam queue yang menggunakan circular array adalah:

a. Proses penghapusan dilakukan dengan cara nilai depan (front) ditambah 1: front=front+1.

b. Proses penambahan elemen sama dengan linear array yaitu nilai belakang ditambah 1: tail=tail + 1.

c. Jika front = maks dan ada elemen yang akan dihapus, maka nilai front = 1.

d. Jika tail = maks dan front tidak 1 maka jika ada elemen yang akan ditambahkan, nilai tail = 1.

e. Jika hanya tinggal 1 elemen di queue (front = tail), dan akan dihapus maka front diisi 0 dan tail diisi dengan 0 (queue kosong).

Front dan Tail akan bergerak maju, jika:

- Untuk penambahan.

  Tail sudah mencapai elemen terakhir array akan memakai elemen pertama array yang telah dihapus.

- Untuk pngahapusan.

  Front telah mencapai elemen terakhir array, maka akan menuju elemen pertama jika antrian masih berisi elemen.

Circular Queue menggunakan fungsi Counter. Counter merupakan jumlah pengantri yang ada dalam antrian.

- Pada proses awal, counter = 0
- Pada proses insert, counter++
- Pada proses delete, countr

Proses Circular Queue :

- AWAL (Inisialisasi)
- INSERT (Sisip, Masuk, Simpan, Tulis)
- DELETE (Hapus, Keluar, Ambil, Dilayani)

Kondisi Antrian :

	Kondisi	Ciri
a	Kosong	COUNTER = 0
b	Penuh	COUNTER = n
c	Bisa diisi	COUNTER < n
d	Ada isinya	COUNTER > 0

Gambar 1.19 Kondisi Antrian

Operasi-Operasi Queue dengan Circular Array

f) IsEmpty Fungsi

IsEmpty berguna untuk mengecek apakah queue sudah terisi atau masih kosong. Hal ini dilakukan dengan mengecek apakah tail lebih besar dari head dan tail masih terletak bersebelahan dengan head. Jika benar demikian, maka queue masih kosong.

g) IsFull Fungsi

IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah tempat yang masih kosong tinggal satu atau tidak (untuk membedakan dengan empty dimana semua tempat kosong). Jika benar berarti queue penuh.

h) EnQueue Fungsi

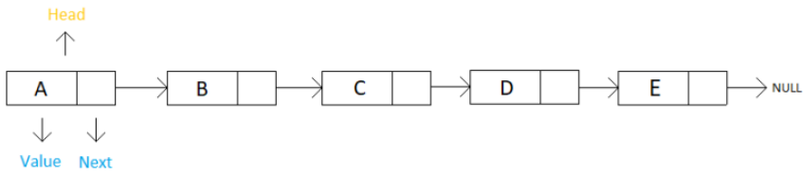
EnQueue berfungsi untuk memasukkan sebuah elemen ke dalam queue tail dan head awal bernilai nol (0).

i) DeQueue

DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara memindahkan posisi head satu posisi ke belakang.

2. Queue dengan Linked-List

Selain menggunakan array, queue juga dapat dibuat dengan linked list. Metode linked list yang digunakan adalah double linked list.



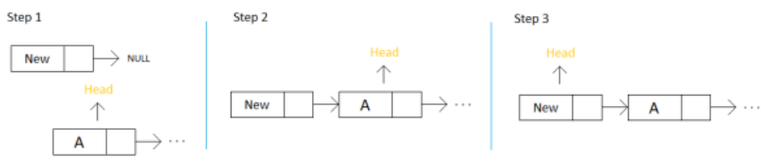
Gambar 1.20 Queue Linked List

Berikut adalah sebuah gambaran linked list sederhana :

- Head, adalah sebuah pointer atau penanda dari awal sebuah linked list.
- Value, adalah sebuah data yang disimpan dalam sebuah item.
- Next, adalah sebuah pointer yang menghubungkan dengan item selanjutnya.

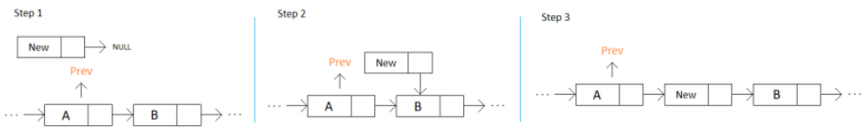
Kemudian berikut adalah ilustrasi dari operasi – operasi yang umum terdapat dalam linked list sederhana yaitu :

Add First:



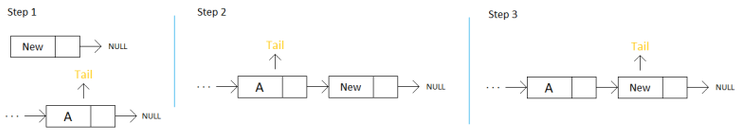
Gambar 1.21 Linked List Sederhana (add Firs).

Add After:



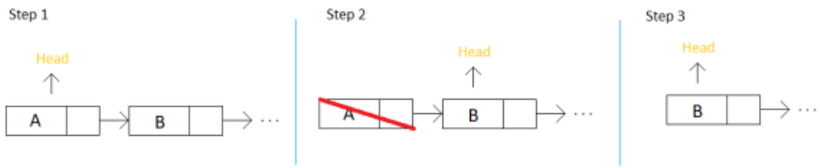
Gambar 1.22 Linked List Add After

Add Last:



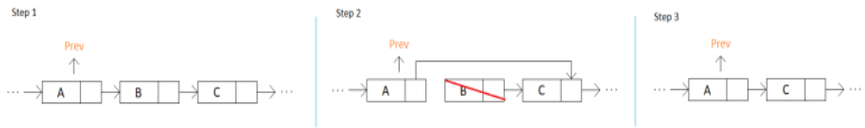
Gambar 1.23 Linked List Add List

Remove First:



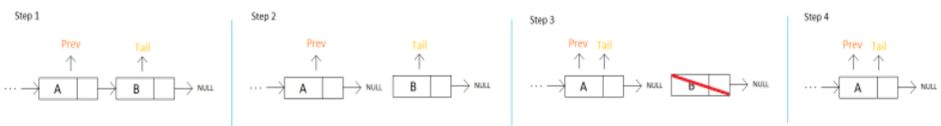
Gambar 1.24 Linked List Remove First

Remove After:



Gambar 1.25 Linked List Remove After

Remove Last:



Gambar 1.26 Linked List Remove Last

Operasi-operasi Queue dengan Double Linked List:

- IsEmpty

  Fungsi IsEmpty berguna untuk mengecek apakah queue masih kosong atau sudah berisi data. Hal ini dilakukan dengan mengecek apakah head masih menunjukkan pada Null atau tidak. Jika benar berarti queue masih kosong.

- IsFull

  Fungsi IsFull berguna untuk mengecek apakah queue sudah penuh atau masih bias menampung data dengan cara mengecek apakah Jumlah Queue sudah sama dengan MAX_QUEUE atau belum. Jika benar maka queue sudah penuh.

- EnQueue

  Fungsi EnQueue berguna untuk memasukkan sebuah elemen ke dalam queue (head dan tail mula-mula meunjukkan ke NULL).

- DeQueue

  Procedure DeQueue berguna untuk mengambil sebuah elemen dari queue. Hal ini dilakukan dengan cara menghapus satu simpul yang terletak paling depan (head).

BAB II

PENUTUP

KESIMPULAN

Queue adalah struktur data yang menerapkan prinsip "First In First Out" (FIFO), di mana elemen-elemen yang dimasukkan pertama kali menjadi yang pertama kali diambil. Ini mencerminkan cara antrian dalam kehidupan sehari-hari, di mana orang atau objek yang pertama kali datang adalah yang pertama kali dilayani. Dalam pengimplementasiannya, Queue dapat diwujudkan menggunakan berbagai struktur data dasar, seperti array atau linked list. Meskipun ada variasi implementasi, inti dari Queue adalah menyediakan dua operasi utama: enqueue, yang menambahkan elemen ke ujung belakang antrian, dan dequeue, yang menghapus elemen dari ujung depan antrian. Operasi enqueue dan dequeue sering memiliki kompleksitas waktu konstan, membuat Queue efisien untuk digunakan dalam situasi di mana pengolahan data perlu mengikuti urutan waktu masuk atau ketibaan. Contohnya termasuk sistem penjadwalan tugas di sistem operasi, antrian pesanan dalam bisnis e-commerce, atau pengelolaan tugas dalam program paralel. Dalam implementasi praktis, Queue seringkali digunakan bersamaan dengan struktur data lain untuk membangun solusi yang lebih kompleks. Misalnya, Queue dapat diimplementasikan menggunakan linked list untuk mendukung operasi enqueue dan dequeue yang efisien, atau digunakan sebagai bagian dari algoritma pencarian graf yang lebih kompleks. Dengan menggunakan Queue, program dapat memproses data secara berurutan dan memastikan bahwa elemen-elemen diproses sesuai dengan urutan kedatangan mereka. Kesederhanaan dan efisiensi operasional membuat Queue menjadi pilihan yang umum dalam berbagai aplikasi dan algoritma di dunia pemrograman.

DAFTAR PUSTAKA

https://www.belajarstatistik.com/blog/2022/02/14/contoh-progam-queue-dalam-bahasa-c/

https://catatulang.blogspot.com/2017/11/queue-dalam-bahasa-c.html

https://www.academia.edu/18129263/Konsep_QUEUE_Dalam_Bahasa_Pemrograman_C_