PRAKTIKUM 24
PRIORITY QUEUE

A. TUJUAN

Mahasiswadiharapkanmampu :

1. Memahamikonsep Priority Queue
2. Memahamiimplementasidari Priority Queue
3. MemahamiRepresentasidan alternative dari model penyimpanantrian

B. DASAR TEORI

Suatukontainer yang berisibeberapa item dimanasetiap item memilikiprioritas. Kemungkinansetiap item mempunyaiwaktutenggat yang berbeda-beda. Item denganprioritastertinggimerupakan item yang akandiprosesataudilayaniselanjutnya . Seringkali, penunggudalamantrianharusdiaturmenurutprioritas.Antriandisimpanmenurutprioritas.Aturan priority queue bukanlagi FIFO murni.Implementasi priority queue dalam sorted table.

Sorted Table

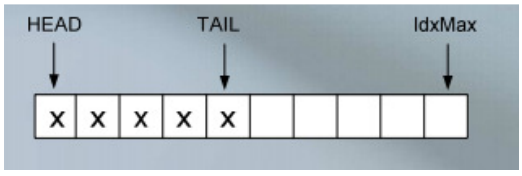
PengambilanelemenuntukdilayaniselaludariHEAD .Penambahanelemendilakukansesuaiprioritas. Elemendalamantrianselaludalamprioritas, denganprioritaslebihtinggiuntukdilayaniselalulebih "dekat" ke HEAD**.**

RepresentasiPenyimpananAntrian

Queue lebihtepatdirepresentasikansebagai table. Terdapattigaalternatifpenyimpananantrian

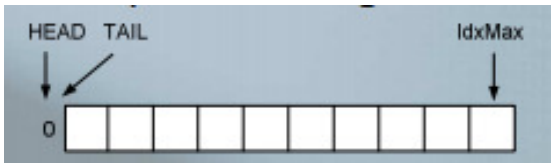
Alternatif I

Tabeldenganhanyarepresentasi TAIL adalahindekselementerakhir. HEAD selaludiset =1 jika queue tidakkosong. Jikaqueue kosong, maka HEAD=0. Berikutiniilustrasi queue tidakkosongdengan 5 elemen:



Gambar 24.1 Queue denganjumlah data 5 (Alternatif 1)

Ilustrasi queue kosong



Gambar 24.2Queue Kosong (Alternatif 1)

Algoritma paling sederhanauntukpenambahanelemen

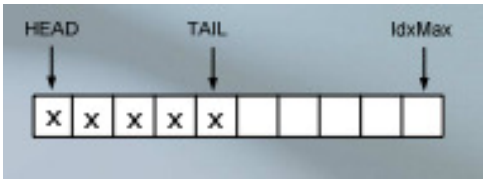
- Jikamasihadatempatadalahdengan “memajukan” TAIL.

Kasuskhususuntuk queue kosongkarena HEADharusdisetnilainyamenjadi 1.

Kasuskhususuntuk queue dengakeadaanawalberelemen 1, yaitumenyesuaikan HEAD & TAIL dengan DEFINISI. Algoritmainimencerminkanpergeseran orang yang sedangmengantri di dunianyatatapitidak EFISIEN.

Alternatif II

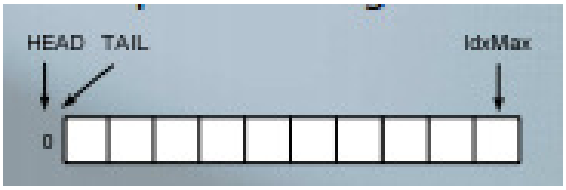
Tabeldenganrepresentasi HEAD & TAIL.HEAD bergerakketikasebuahelemendihapusjika queue tidakkosong.Jika queue kosongmaka HEAD=0. Ilustrasi queue tidakkosong, dengan 5 elemen, kemungkinanpertama HEAD sedang di posisiawal:



Gambar 24.3Queue denganjumlah data 5 (Alternatif 2)

Ilustrasi queue tidakkosong, dengan 5 elemen, kemungkinanpertama HEAD tidakberada di posisiawal. Hal initerjadiakibatalgoritmapenghapusan yang dilakukan.

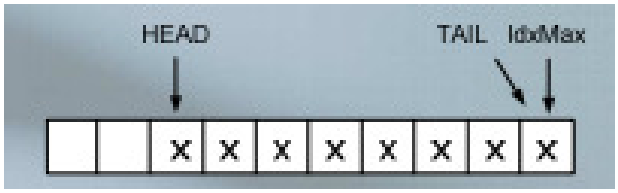
Ilustrasi queue kosong



Gambar 24.4Queue Kosong (Alternatif2)

Algoritmauntukpenghapusanelemenjika queue tidakkosong: ambilnilaielemen HEAD, kemudia HEAD maju. Kasuskhususuntuk queue dengankeadaanawalberelemen 1, yaitumenyesuaikan HEAD& TAIL dengan DEFINISI. Algoritmainitidakmencerminkanpergeseran orang yang sedangmengantri di dunianyata, tapiefisien.

Namunsuatusaatterjadikeadaan queue penuhtetapi "semu" sebagaiberikut:



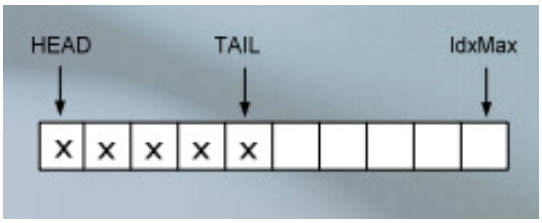
Gambar 24.5Queue Kosong di indekdepan

Jikakeadaaniniterjadi,
haruslahdilakukanaksimenggeserelemenuntukmenciptakanruangankosong.
Pergeserandilakukanjikadanhanyajika.

Tabeldenganrepresentasi HEAD & TAIL "berputar" mengelilingiindekstabeldariawalsampaiakhir, kemudiankembalikeawal.Jika queue kosongmaka HEAD=0. Representasimemungkinkantidakperlulagiadapergeseran yang harusdilakukansepertipadaalternatif II padasaatpenambahanelemen.

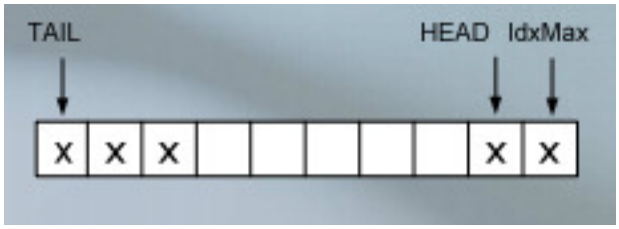
Alternatif III

Tabeldenganrepresentasi HEAD & TAIL "berputar" mengelilingiindekstabeldariawalsampaiakhir, kemudiankembalikeawal. Jika queue kosongmaka HEAD=0. Representasimemungkinkantidakperlulagiadapergeseran yang harusdilakukansepertipadaalternatif II padasaatpenambahanelemen
Ilustrasi queue tidakkosongdengan 5 elemendan HEAD berada di posisiawal:



Gambar 24.6Queue denganjumlah data 5 (Alternatif 3)

Ilustrasi queue tidakkosongdengan 5 elemen, HEAD tidakberada di posisiawal, tetapi “>” atau “sesudah” TAIL.



Gambar 24.7Queue denganjumlah data 5, posisi head > tail (Alternatif 3)

Algoritmauntukpenambahanelemen

- Jikamasihadatempatadalahdenganmemajukan TAIL.
- Jika TAIL sudahmencapaiIdxMaxmakasuksesordariIdxMaxadalah 1 sehingga TAIL yang baru=1.
- Jika TAIL belummencapaiIdxMaxmakaalgoritmapenambahanelemen = altern II.

Kasuskhususuntuk queue kosongkarena HEADharusdisetnilainyamenjadi 1. Algoritmauntukpenghapusanelemen :

- Jika queue tidakkosongambilnilaielemen HEAD, kemudian HEAD maju.

Penentuansuksesordariindeksyang sudahdiubah/”maju” dibuatsepertialgoritmapenambahanelemen.

- Jika HEAD mencapaiIdxMax, makasuksesordari HEAD =1

Kasuskhususuntuk queue dengankeadaanawalberelemen 1, yaitumenyesuaikan HEAD & TAIL dengan DEFINISI.Algoritmaini EFISIEN karenatidakperlupergeserandanseringkalistrategipemakaiantabelsemacaminidisebutsebagai “circular buffer”, dimanatabelpenyimpanelemendianggapsebagai “buffer”.

RepresentasiPenyimpanan Priority Queue

Tempatpenyimpananantriandapatdiadaptasidarisalahsatualternatif.Penomoranprioritasdapatditentukan, misal:

- Prioritasbernilai 0 s/d PrioMax, dengan 0 untukprioritas paling tinggi (ascending)
- Prioritasbernilai 0 s/d PrioMax, dengan 0 untukprioritas paling rendah (descending)
- Contohkeadaan queue denganprioritas 0 s/d 5, tersusun ascending

- Ilustrasi queue tidakkosong, dengan 5elemen, dengan HEAD tidakberada di posisiawal, tetapimasih "lebihkecil" atau "sebelum" TAIL. Angka(1,0) berartinilaiinformasi yang disimpanadalah 1 denganprioritas 0.

ADT Priority Queue (PQ)

Representasifisik:

- Tabelkontigu
- Berkait pointer

Primitif-primitif ADT PQ yang didefinisikan:

- Menentukanapakahsuatu PQ kosongatautidak (IsEmpty)
- Menentukanapakahsuatu PQ telahpenuhataubelum (IsFull)
- Mengirimkanbanyaknyaelemen PQ (NbElmt) Menginisialisasisebuah PQ kosongCreateEmpty)
- Mengembalikansemuamemori PQ, PQ kosong (dealokasi)
- Menambahkanelemen X pada PQ denganaturan PQ (add)
- Menghapuselemen X pada PQ denganaturan FIFO (del)

C. TUGAS PENDAHULUAN

Buatlah resume 1 halamanmengenai**PriorityQueue** danberikanpenjelasannya.!

D. PERCOBAAN

Dalamantrianberprioritas, setiapelemenn yang akanmsukdalamantriansudahditentukanlebih dahuluprioritasnya. Dalamhaliniberlakuduaketentuan, yaitu:

1. Elemen-elemen yang mempunyaiprioritaslebihtinggiakandiproseslebihdahulu.
2. Duaelemen yang mempunyaiprioritassamaakandikerjakansesuaidenganurutanpadasaatkeduaelemeninimasukdalamantrian.

Percobaan1 :MemahamiPenggunaandari class PriorityQueue

```
importjava.util.*;
```

```
public class PriorityQueueDemo {
    public static void main(String[] args) {
PriorityQueue<String>stringQueue;
stringQueue = new PriorityQueue<String>();
stringQueue.add("ab");
stringQueue.add("abcd");
stringQueue.add("abc");
stringQueue.add("a");
        //don't use iterator which may or may not
        //show the PriorityQueue's order
        while (stringQueue.size() > 0) {
System.out.println(stringQueue.remove());
        }
    }
}
```

Percobaan2 :MemahamiPenggunaandari class PriorityQueuedan data yang tersimpandalamobjekPriorityQueuemengimplementasikan interface Comparator.

```
import java.util.Comparator;
import java.util.PriorityQueue;

public class PQueueTest {
    public static void main(String[] args) {
PriorityQueue<Integer>pQueue    =    new    PriorityQueue<Integer>(10,    new
Comparator<Integer>() {
            public int compare(Integer int1, Integer int2) {
boolean flag1 = isPrime(int1);
boolean flag2 = isPrime(int2);
                if (flag1 == flag2) {
                    return int1.compareTo(int2);
                } else if (flag1) {
                    return -1;
                } else if (flag2) {
                    return 1;
                }
                return 0;
            }
        });
pQueue.add(1);
pQueue.add(5);
pQueue.add(6);
pQueue.add(4);
pQueue.add(2);
pQueue.add(9);
pQueue.add(7);
pQueue.add(8);
pQueue.add(10);
pQueue.add(3);
        while (true) {
            Integer head = pQueue.poll();
            if (head == null) {
                break;
            }
```

```
System.out.print(head + " <-- ");
        }
    }

    /**
     *
     * @param n
     * @return
     */
    public static booleanisPrime(int n) {
        if (n <= 1) {
            return false;
        }
        if (n == 2) {
            return true;
        }
        if (n % 2 == 0) {
            return false;
        }
        long m = (long) Math.sqrt(n);
        for (long i = 3; i <= m; i += 2) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Percobaan 3.

```
import java.util.Comparator;
import java.util.PriorityQueue;

enumProductQuality {
    High, Medium, Low
}

class Product implements Comparable<Product> {

    String name;
ProductQuality priority;

    Product(String str, ProductQualitypri) {
        name = str;
        priority = pri;
    }

    public intcompareTo(Product msg2) {
        return priority.compareTo(msg2.priority);
    }
}

class MessageComparator implements Comparator<Product> {

    public int compare(Product msg1, Product msg2) {
```

```
        return msg2.priority.compareTo(msg1.priority);
    }
}

public class Main {

    public static void main(String args[]) {
PriorityQueue<Product>pq = new PriorityQueue<Product>(3);
pq.add(new Product("A", ProductQuality.Low));
pq.add(new Product("B", ProductQuality.High));
pq.add(new Product("C", ProductQuality.Medium));
        Product m;
        while ((m = pq.poll()) != null) {
System.out.println(m.name + " Priority: " + m.priority);
        }
PriorityQueue<Product>pqRev     =     new     PriorityQueue<Product>(3,     new
MessageComparator());
pqRev.add(new Product("D", ProductQuality.Low));
pqRev.add(new Product("E", ProductQuality.High));
pqRev.add(new Product("F", ProductQuality.Medium));
        while ((m = pqRev.poll()) != null) {
System.out.println(m.name + " Priority: " + m.priority);
        }
    }
}
```

Percobaan4 : The program processes job requests with different employee statuses. Output lists the jobs by status along with their total time

```
import java.io.*;
import java.util.Scanner;
import ds.util.HeapPQueue;

public class Program15_3 {

    public static void main(String[] args)
            throws IOException {
        // handle job requests
HeapPQueue<JobRequest>jobPool = new HeapPQueue<JobRequest>();
        // job requests are read from file "job.dat"
        Scanner sc = new Scanner(new FileReader("job.dat"));
        // time spent working for each category
        // of employee
        // initial time 0 for each category
int[] jobServicesUse = {0, 0, 0, 0};
JobRequest job = null;
        // read file; insert each job into
        // priority queue
        while ((job = JobRequest.readJob(sc)) != null) {
jobPool.push(job);
        }
        // delete jobs from priority queue
        // and output information
System.out.println("Category Job ID" + " Job Time");
        while (!jobPool.isEmpty()) {
```

```
            // remove a job from the priority
            // queue and output it
            job = (JobRequest) jobPool.pop();
System.out.println(job);
            // accumulate job time for the
            // category of employee
jobServicesUse[job.getStatus().value()] +=
job.getJobTime();
        }
System.out.println();
writeJobSummary(jobServicesUse);
    }

    private static void writeJobSummary(
int[] jobServicesUse) {
System.out.println("Total Pool Usage");
System.out.println(" President "
                + jobServicesUse[3]);
System.out.println(" Director "
                + jobServicesUse[2]);
System.out.println(" Manager "
                + jobServicesUse[1]);
System.out.println(" Clerk "
                + jobServicesUse[0]);
    }
}
```

E. LATIHAN

1. MengimplementasikanPriorityQueue

Tabel 24.1 Interface PriorityQueue

Interface PQueue	
booleanisEmpty()	Mengembalikannilai true jika queue kosongdan false jika queue terdapat minimal satuelemen
T peek()	Mengembalikanelemendenganprioritastertinggi. Jika queue kosong melempar exception yaitu: throws NoSuchElementException.
T pop()	Menghapuselemen di depan queue danmengembalikannilaiberdasarkan prioritastertinggi. Jika queue kosongmakamelempar exception yaitu : NoSuchElementException.
void push(T item)	Menyisipkan item di queue
int size()	Mengembalikanjumlahelemendari queue

Membuat Class HeapPQueue ,denganmengurutkan data menggunakan Comparatoratau Comparable

```
import java.util.Collections;
```

```
import java.util.Comparator;
import java.util.LinkedList;

public class HeapPQueue<T extends Comparable<? super T>>implements
PQueue<T> {

    private LinkedList<T>qlist = null;
    private Comparator<T> comparator = null;

    public HeapPQueue() {
qlist = new LinkedList<T>();
    }

public HeapPQueue(Comparator comp) {
qlist = new LinkedList<T>();
        comparator = comp;
    }

    @Override
    public booleanisEmpty() {
throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public T peek() {
throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public T pop() {
throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void push(T item) {
throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public int size() {
throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

Selanjutnyalakukanpengujianterhadapclass**HeapPQueue .**

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.