

Queue dengan Struktur Berkait

IF1210 – Algoritma dan Pemrograman 1
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Queue



Queue adalah sederetan elemen yang:

- dikenali elemen pertama (HEAD) dan elemen terakhirnya (TAIL).
- aturan penambahan dan penghapusan elemennya didefinisikan sebagai berikut:
Penambahan selalu dilakukan setelah **elemen terakhir**,
Penghapusan selalu dilakukan pada **elemen pertama**.

Definisi operasi

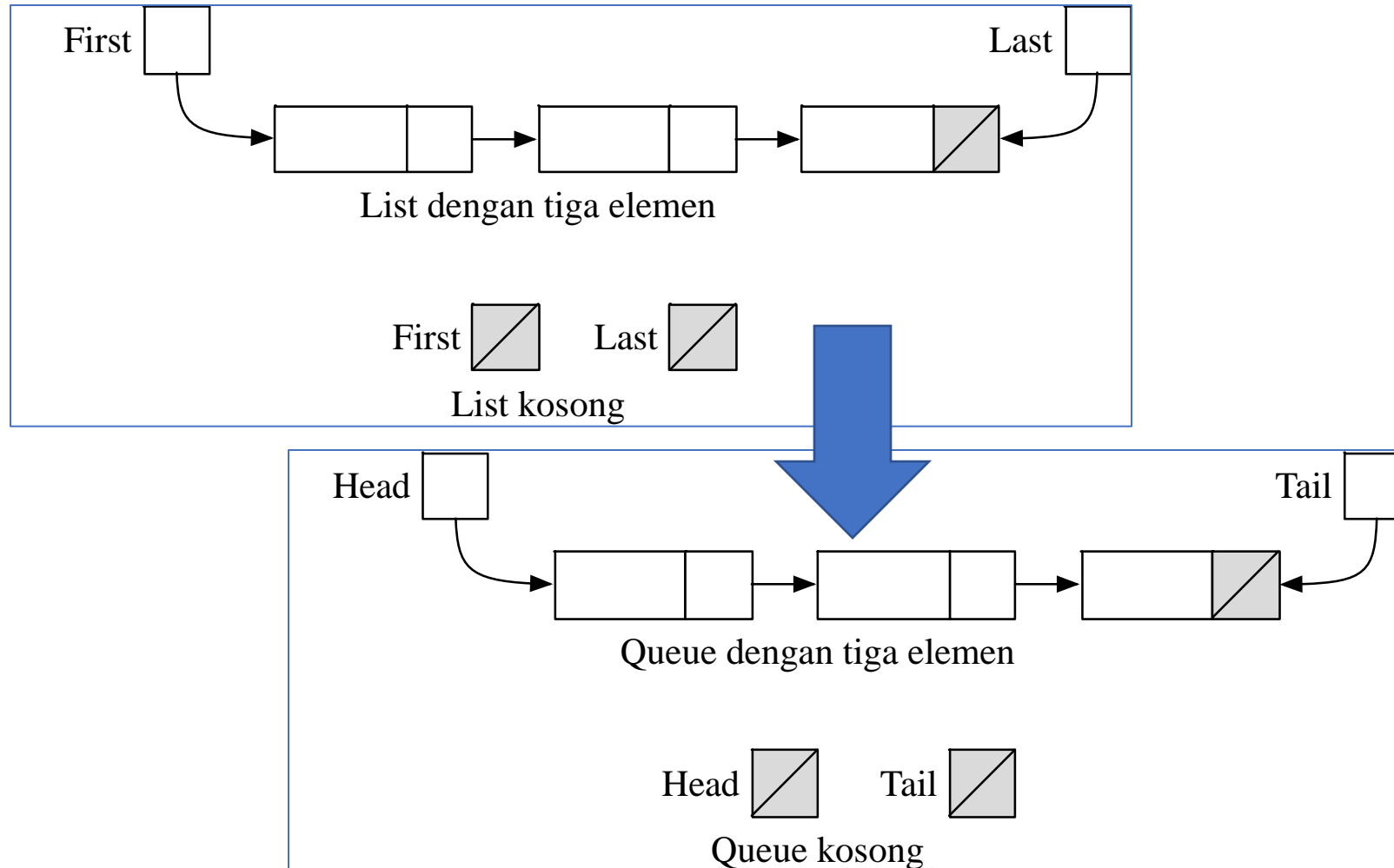
Jika diberikan Q adalah Queue dengan elemen $ElmtQ$

$CreateQueue: \rightarrow Q$	{ Membuat sebuah antrian kosong }
$head: Q \rightarrow ElmtQ$	{ Mengirimkan elemen terdepan Q saat ini }
$length: Q \rightarrow \underline{integer}$	{ Mengirimkan banyaknya elemen Q saat ini }
$enqueue: ElmtQ \times Q \rightarrow Q$	{ Menambahkankan sebuah elemen setelah elemen paling belakang Queue }
$dequeue: Q \rightarrow Q \times ElmtQ$	{ Menghapus kepala Queue, mungkin Q menjadi kosong }
$isEmpty: Q \rightarrow \underline{boolean}$	{ Tes terhadap Q : true jika Q kosong, false jika Q tidak kosong }

Representasi berkait seperti apa yang paling cocok untuk queue?

Queue dengan Struktur berkait

List Linier yang dicatat first dan last \approx queue



Operasi-operasi dasar pada Queue

CreateQueue \asymp CreateList

enqueue \asymp insertLast

dequeue \asymp deleteFirst

length \asymp length

isEmpty \asymp isEmpty

ADT Queue dengan Rep. List

```
/* File: queue_linked.h */
#ifndef QUEUE_LINKED_H
#define QUEUE_LINKED_H
#include "boolean.h"
#include <stdlib.h>

#define NIL NULL
/* Deklarasi infotype */
typedef int ElType;
/* Queue dengan representasi berkait dengan pointer */
typedef struct node* Address;
typedef struct node {
    ElType info;
    address next;
} Node;

/* Type queue dengan ciri HEAD dan TAIL: */
typedef struct {
    address addrHead; /* alamat penghapusan */
    address addrTail; /* alamat penambahan */
} Queue;
```

ADT Queue dengan Rep. List

```
/* Selektor */
#define NEXT(p) (p)->next
#define INFO(p) (p)->info

#define ADDR_HEAD(q) (q).addrHead
#define ADDR_TAIL(q) (q).addrTail
#define HEAD(q) (q).addrHead->info

/* Prototype manajemen memori */
Address newNode(ElType x);
/* Mengembalikan alamat sebuah Node hasil alokasi dengan info = x,
   atau NIL jika alokasi gagal */
```


ADT Queue dengan Rep. List

```
boolean isEmpty(Queue q);
/* Mengirim true jika q kosong: ADDR_HEAD(q)=NIL and ADDR_TAIL(q)=NIL */
int length(Queue q);
/* Mengirimkan banyaknya elemen queue. Mengirimkan 0 jika q kosong */

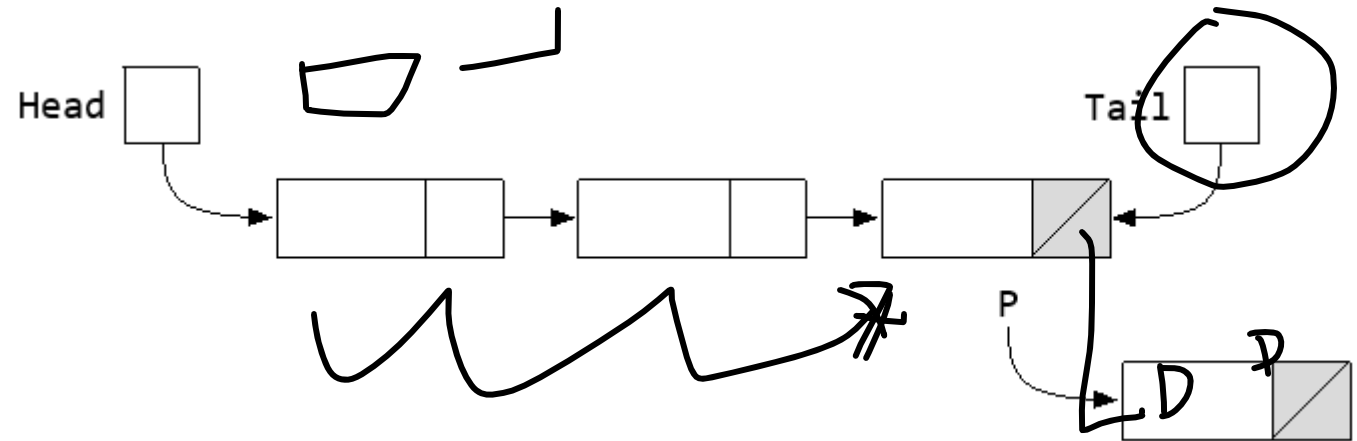
/**/ Kreator /**/
void CreateQueue(Queue *q);
/* I.S. sembarang */
/* F.S. Sebuah q kosong terbentuk */

/**/ Primitif Enqueue/Dequeue /**/
void enqueue(Queue *q, ElType x);
/* Proses: Mengalokasi x dan menambahkan x pada bagian Tail dari q
    jika alokasi berhasil; jika alokasi gagal q tetap */
/* Pada dasarnya adalah proses insertLast */
/* I.S. q mungkin kosong */
/* F.S. x menjadi Tail, Tail "maju" */
void dequeue(Queue *q, ElType *x);
/* Proses: Menghapus x pada bagian HEAD dari q dan mendealokasi elemen HEAD */
/* Pada dasarnya operasi deleteFirst */
/* I.S. q tidak mungkin kosong */
/* F.S. x = nilai elemen HEAD pd I.S., HEAD "mundur" */
#endif
```

ADT Queue dengan Rep. List

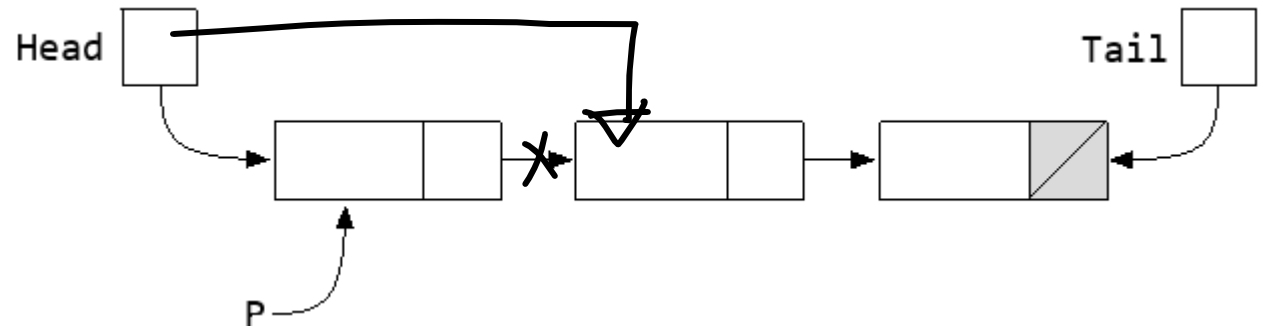
```
void enqueue(Queue *q, ElType x) {  
    /* Proses: Mengalokasi x dan menambahkan x pada bagian Tail dari q  
       jika alokasi berhasil; jika alokasi gagal q tetap */  
    /* Pada dasarnya adalah proses insertLast */  
    /* I.S. q mungkin kosong */  
    /* F.S. x menjadi Tail, Tail "maju" */
```

```
    /* Kamus Lokal */  
    address p;  
    /* Algoritma */  
    p = newNode(x);  
    if (p != NIL) {  
        if (isEmpty(*q)) {  
            ADDR_HEAD(*q) = p;  
        } else {  
            NEXT(ADDR_TAIL(*q)) = p;  
        }  
        ADDR_TAIL(*q) = p;  
    } /* else: alokasi gagal, q tetap */  
}
```



ADT Queue dengan Rep. List

```
void dequeue(Queue *q, ElType *x) {  
    /* Proses: Menghapus x pada bagian HEAD dari q dan mendealokasi  
       elemen HEAD */  
    /* Pada dasarnya operasi delete first */  
    /* I.S. q tidak mungkin kosong */  
    /* F.S. x = nilai elemen HEAD pd I.S., HEAD "mundur" */  
  
    /* Kamus Lokal */  
    address p;  
    /* Algoritma */  
    *x = HEAD(*q);  
    p = ADDR_HEAD(*q);  
    ADDR_HEAD(*q) = NEXT(ADDR_HEAD(*q));  
    if (ADDR_HEAD(*q) == NIL) {  
        ADDR_TAIL(*q) = NIL;  
    }  
    NEXT(p) = NIL;  
    free(p);  
}
```



Priority Queue

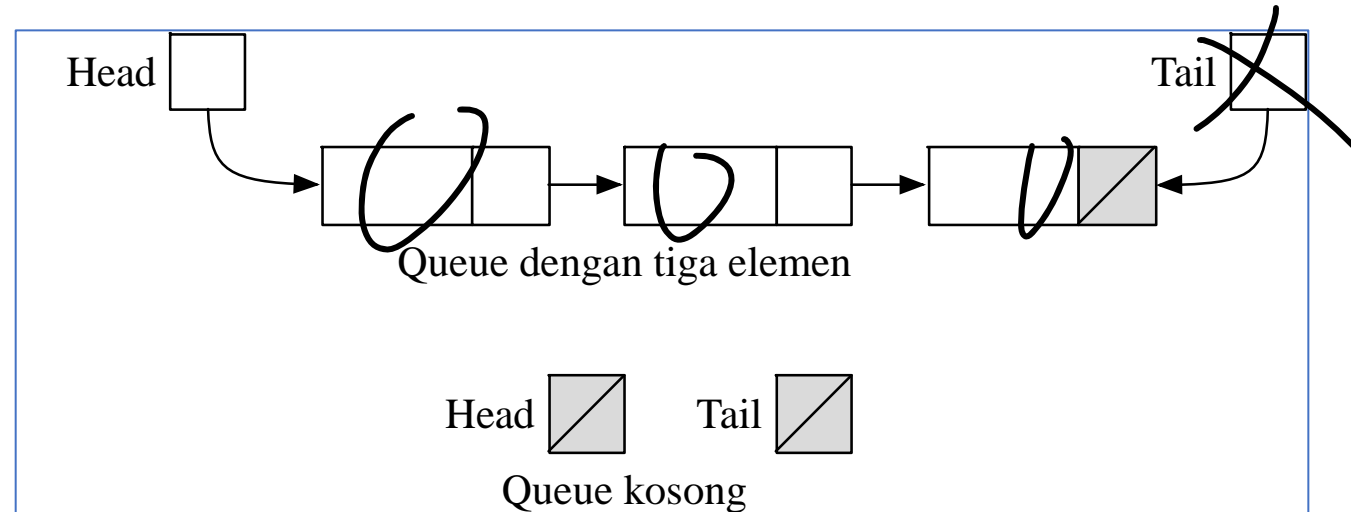
Priority queue:

- Elemen queue terurut menurut suatu prioritas tertentu
- Sering dianggap sebagai *modified queue*
- Menambahkan elemen berarti menambahkan elemen sesuai urutan prioritas
- Menghapus elemen adalah menghapus elemen dengan prioritas tertinggi/terendah (pada bagian Head)

Representasi berkait seperti apa yang paling cocok untuk priority queue?

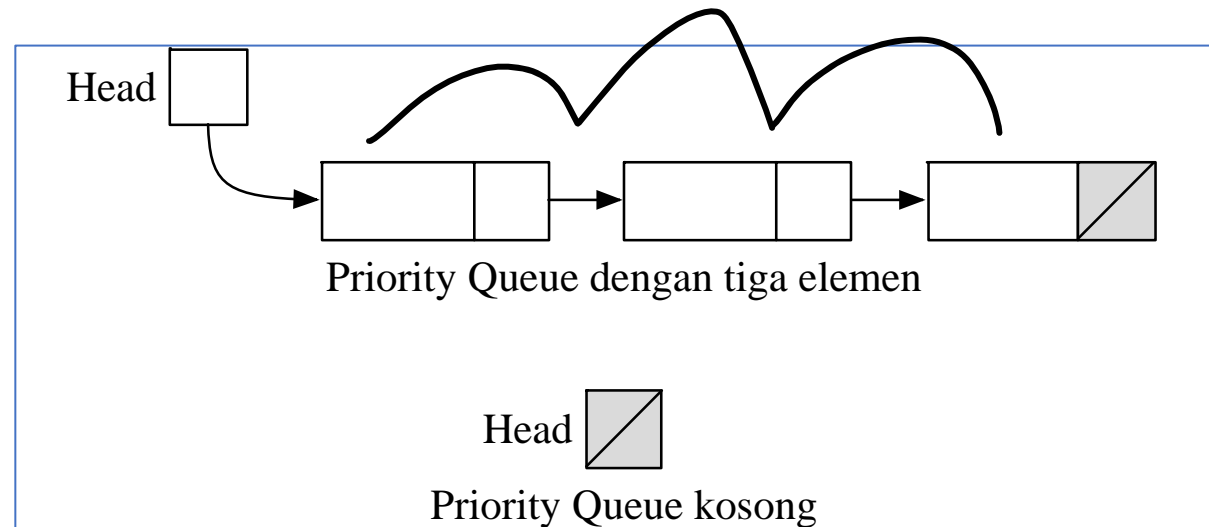
Priority Queue dengan Rep. List

List Linier yang dicatat first dan last \approx queue “biasa”



Priority Queue dengan Rep. List

List linier “biasa” ~ priority queue



Operasi-operasi dasar Priority Queue

Node elemen queue:

```
type Node: < info: ElType,  
                prio: integer,  
                next: Address >
```

- Enqueue → menambahkan elemen secara terurut mengikuti prioritas tertentu
- Dequeue → menghapus elemen dengan prioritas tertinggi (yaitu di Head)
 - Pada dasarnya sama saja dengan Dequeue pada queue/list biasa

Perhatikan representasi logik yang digunakan