ساختمان داده‌ها و الگوریتم‌ها

فصل سوم (جلسه دهم)

S.Najjar.G@Gmail.com

صمد نجارقابل

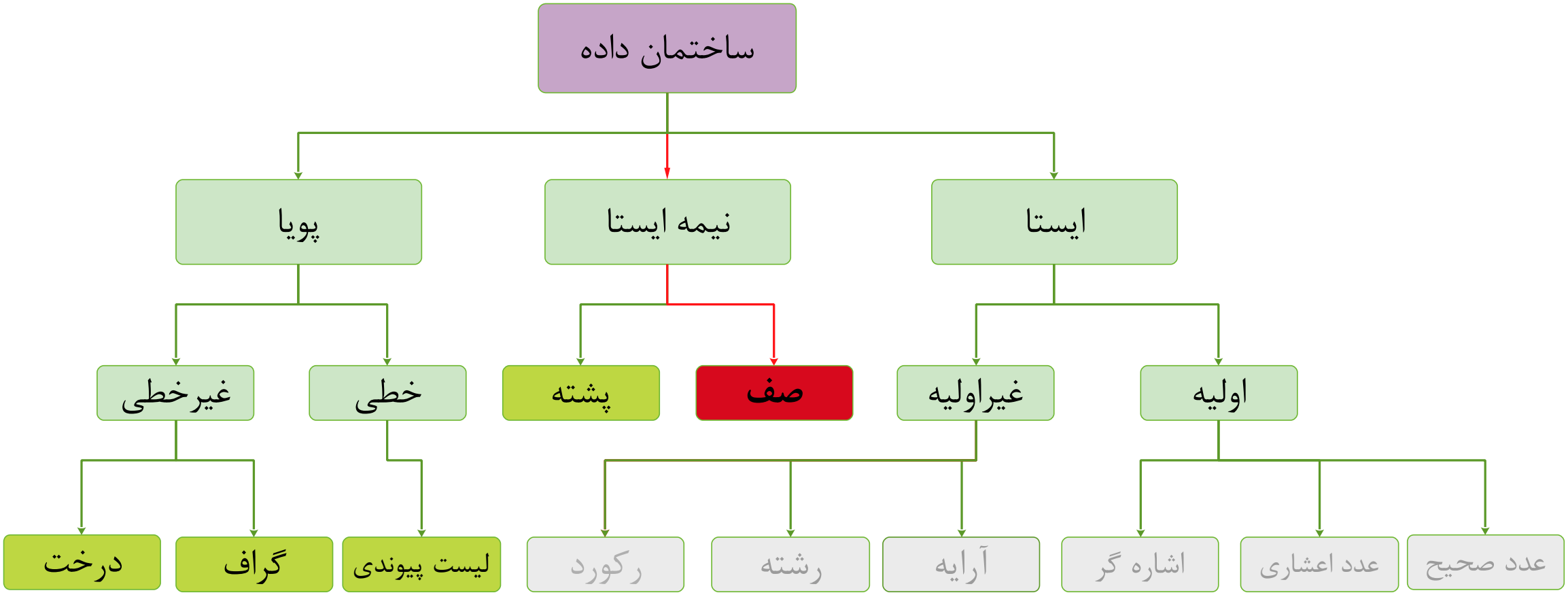
S.Najjar.G@Gmail.com

فهرست مطالب

- مقدمه‌ای بر الگوریتم‌ها و مفاهیم پایه
- معرفی پیچیدگی زمانی و حافظه‌ای و روشهای تحلیل مسائل
- **معرفی ساختمان داده‌های مقدماتی و الگوریتم‌های وابسته به آنها**
  - آرایه
  - **صف**
  - پشته
  - لیست پیوندی
- تئوری درخت و گراف و الگوریتم‌های مرتبط
- الگوریتم‌های مرتب‌سازی و تحلیل پیچیدگی مربوط به آنها
- مباحث تکمیلی در ساختمان داده‌ها

S.Najjar.G@Gmail.com

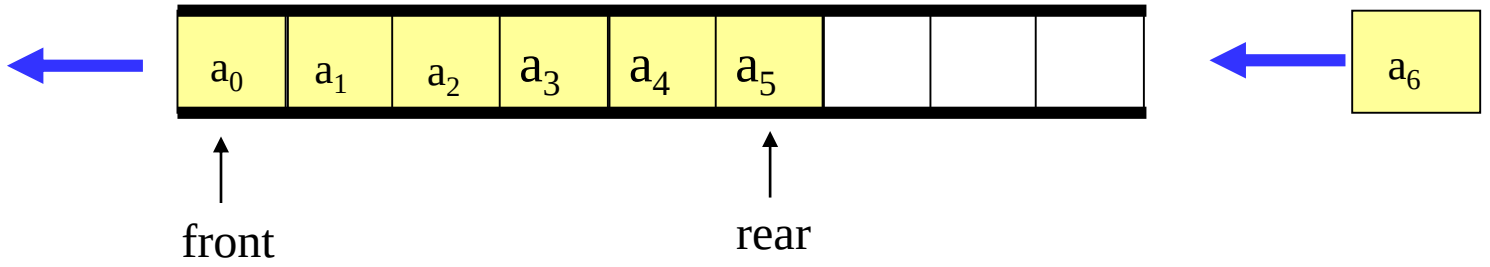
دسته بندي ساختمان داده‌ها



S.Najjar.G@Gmail.com

Queue (صف)

- A queue is an ordered list in which all insertions take place at one end and all deletions take place at the opposite end. It is also known as First-In-First-Out (FIFO) lists.



Abstract Data Type of queue

S.Najjar.G@Gmail.com

structure *Queue* is
objects: a finite ordered list with zero or more elements.
functions:
for all *queue* ∈ *Queue*, *item* ∈ *element*,
max_ queue_ size ∈ positive integer
Queue CreateQ(*max_queue_size*) ::=
create an empty queue whose maximum size is
max_queue_size
Boolean IsFullQ(*queue*, *max_queue_size*) ::=
if(number of elements in *queue* == *max_queue_size*)
return *TRUE*
else return *FALSE*
Queue AddQ(*queue*, *item*) ::=
if (IsFullQ(*queue*)) *queue_full*
else insert *item* at rear of *queue* and return *queue*

S.Najjar.G@Gmail.com

Abstract Data Type of queue (Cont.)

Boolean IsEmptyQ(*queue*) ::=
    if (*queue* ==CreateQ(*max_queue_size*))
    return *TRUE*
    else return *FALSE*
Element DeleteQ(*queue*) ::=
    if (IsEmptyQ(*queue*)) **return**
    else remove and return the *item* at front of queue.

Implementation 1: using array

S.Najjar.G@Gmail.com

```
Queue CreateQ(max_queue_size) ::=
# define MAX_QUEUE_SIZE 100/* Maximum queue size */
typedef struct element{
            int key;
            /* other fields */
            };
element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;
Boolean IsEmpty(queue) ::= front == rear
Boolean IsFullQ(queue) ::= rear == MAX_QUEUE_SIZE-1
```

S.Najjar.G@Gmail.com

Implementation 1: using array (Cont.)

Add to a queue

```
void addq(int rear, element item)
{
/* add an item to the queue */
    if (rear == MAX_QUEUE_SIZE-1) {
       queue_full( );
       return;
    }
   queue [++rear] = item;
}
```

S.Najjar.G@Gmail.com

Implementation 1: using array (Cont.)

Delete from a queue

```
element deleteq(int front, int rear)
{
/* remove element at the front of the queue */
   if ( front == rear)
       return queue_empty( );      /* return an error key */
   return queue [++ front];
}
```

problem: there may be available space when IsFullQ is true I.E. movement is required.

Solution: Circular Queue

S.Najjar.G@Gmail.com

Implementation 2: circular queue

To resolve the issue of moving elements in the queue, circular queue assigns next element to q[0] when

rear == MaxSize – 1.

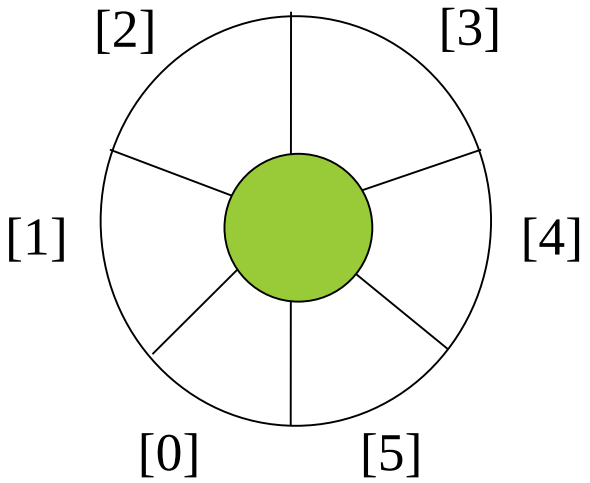
front: one position counterclockwise from the first element

rear: current end

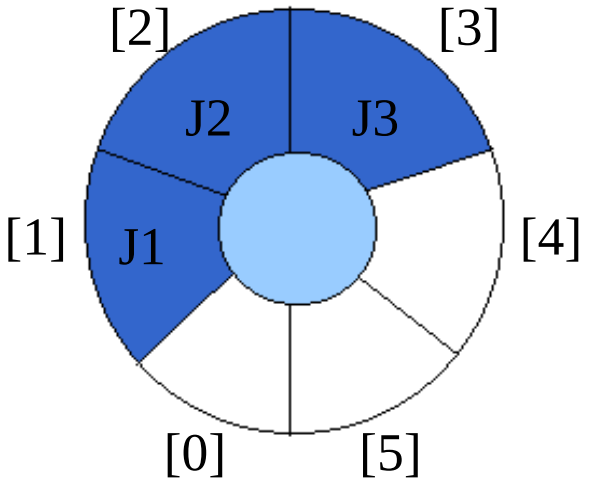
S.Najjar.G@Gmail.com

Implementation 2: circular queue (Cont.)

EMPTY QUEUE



front = 0
rear = 0



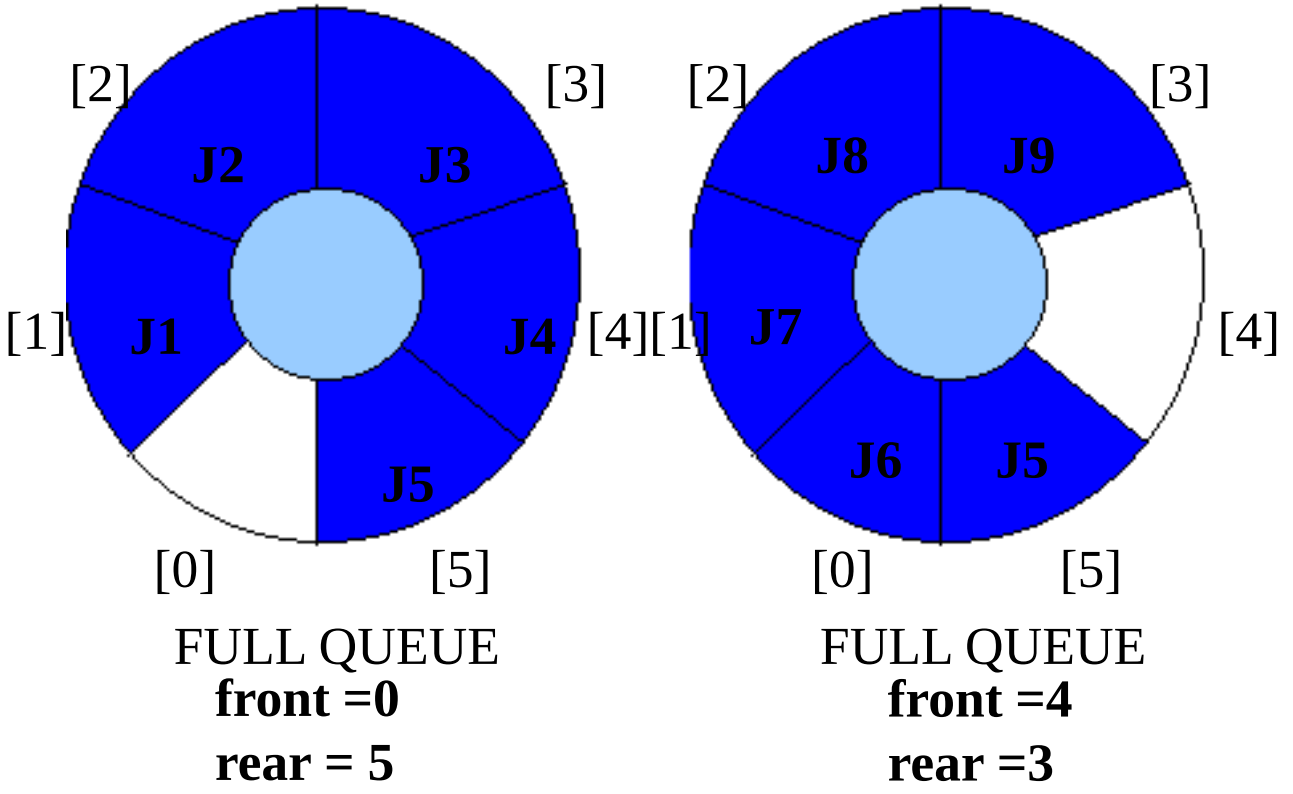
front = 0
rear = 3

S.Najjar.G@Gmail.com

Implementation 2: circular queue (Cont.)

Problem: Queue is empty when front == rear. But it is also true when queue is full. This will be a problem.

Solution:one space is left when queue is full



S.Najjar.G@Gmail.com

Implementation 2: circular queue (Cont.)

Add to a circular queue

```
void addq(int front, int rear, element item)
{
/* add an item to the queue */
     if (front == (rear +1) % MAX_QUEUE_SIZE)
      {
        queue_full();/* print error */
         return;
       }
     rear = (rear +1) % MAX_QUEUE_SIZE;
     queue[rear] = item;
}
```

S.Najjar.G@Gmail.com

Implementation 2: circular queue (Cont.)

Delete from a circular queue

```
element deleteq(int front, int rear)
{
   element item;
   /* remove front element from the queue and put it in item */
       if (front == rear)
          return queue_empty( );  /* queue_empty returns an error key */

      front = (front+1) % MAX_QUEUE_SIZE;
      return queue[front];
}
```

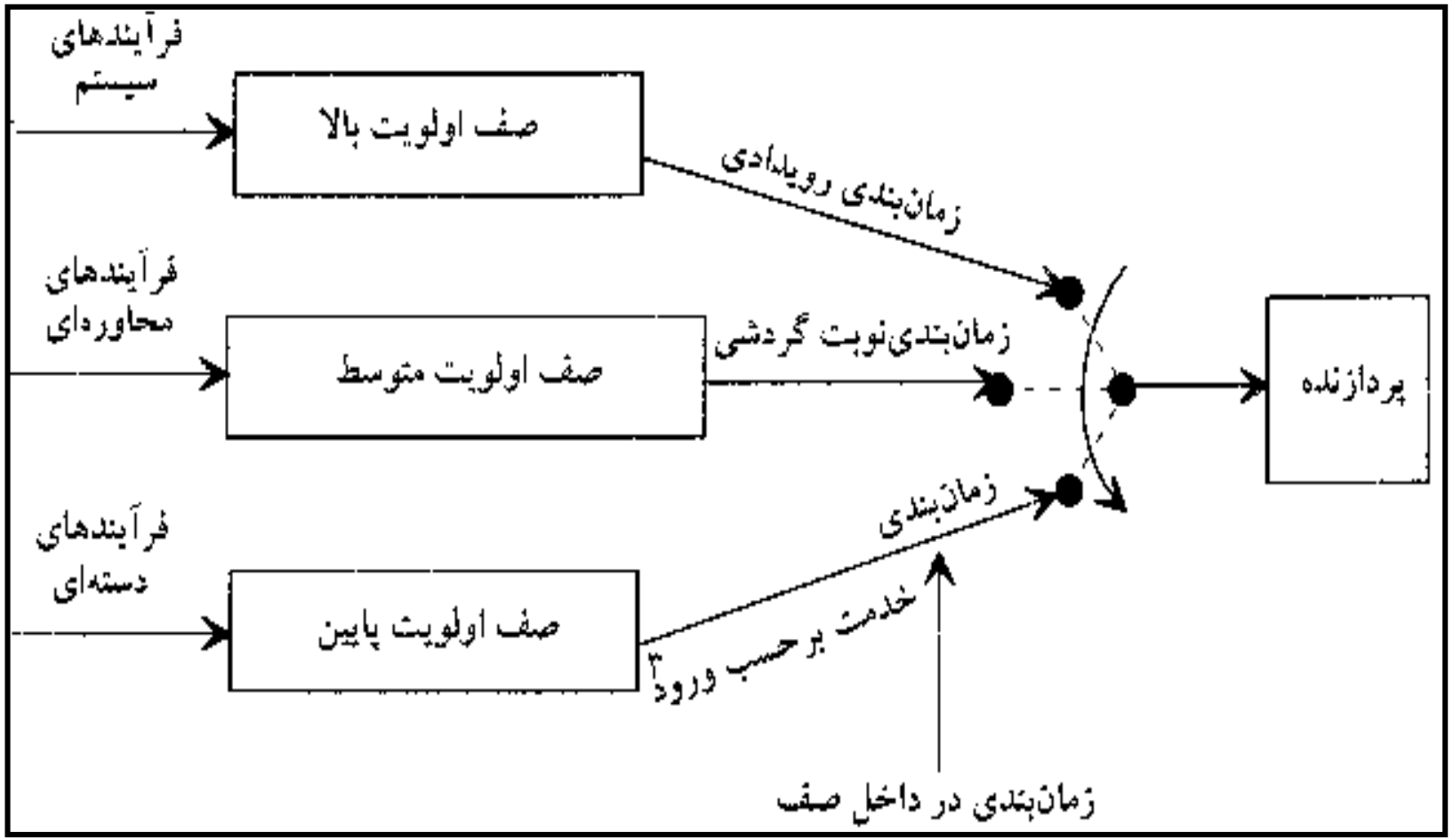
S.Najjar.G@Gmail.com

کاربردهای صف

- صف کاربردهای زیادی در علم کامپیوتر دارد.یکی از کاربردهای مهم صف در شبیه‌سازی است.
- کاربرد مهم دیگر صف، در پیاده سازی جنبه‌های مختلف سیستم عامل است. محیط چند برنامه ای، برای کنترل برنامه ها از چندین صف استفاده می‌کند.

S.Najjar.G@Gmail.com

کاربردهای صف



S.Najjar.G@Gmail.com

Home Work 5

- الگوریتم صف خطی را با استفاده از آرایه در یکی از زبانهای برنامه نویسی پیاده سازی و مثالی روی آن اجراء نمایید.

S.Najjar.G@Gmail.com

Home Work 6

- الگوریتم صف حلقوی را با استفاده از آرایه در یکی از زبانهای برنامه نوسی پیاده سازی نمایید و با اجراء آن مثالی را تست کنید.