
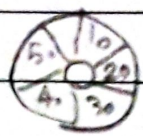



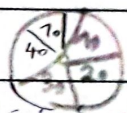
40230172089

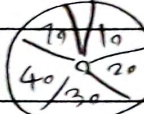
زینب عباسی

enqueue(50) [10, 20, 30, 40, 50]  ①

enqueue(60) [10, 20, 30, 40, 50, 60]  overflow Error

dequeue() [20, 30, 40, 50, 60]  بدون 10

enqueue(70) [20, 30, 40, 50, 60, 70]  آفرین انگاه

enqueue(80) Error_max=6  overflow error

② در صف فعلی و وقت عملیات dequeue صورت گیرد و صف از نظر

ظاهر کما میتر می شود اما در واقع فضای خالی در ابتدای صف موجود

است و باعث مدر رفتن حافظه می شود. چون دیگر قابلیت اضافه کردن عناصر به ابتدای صف را نداریم

راهی که در صف فعلی برای جلوگیری از این اتفاق وجود دارد کیفیت دادن

عناصر به انتهای صف با $T(n) = O(n)$ یا استفاده از صف میرفتی است.

در صف میرفتی معماً معضله را dequeue می کنیم

راه اولین عنصر لیست اشاره می دهیم که به سبب زنجاری این عملیات

$O(1)$ است. در مورد مرتبه زمانی dequeue $O(1)$

enqueue در مورد مرتبه زمانی $O(1)$ است

۵) طراحی صف در لیست آرایه ای راحت تر است و میتوانیم به تمام

راحت تر دسترسی داشته باشیم (به دلیل اندیس ها) در scale

کوید استفاده از این صف از نظر زمانی بهینه تر است

اما اندازه ثابت آرایه محدودیت حافظه (در صورتی که لیست درمیان

صف موردنیاز باشد) از مطالب صف آرایه ای می باشد.

در صف پیوندی، اندازه صف را با یک لیست و مدیریت حافظه

نداریم. اما طراحی این صف به دلیل پیوندهای پیچیده آرایه ای

نیست به آرایه ای دارد و برای ذخیره پیوندها نیز به فضای

حافظه بیشتری نیاز پیدا می کنیم

همچنین برای یابی از حالت پیوندی به پیوندهای

آرایه محدود و نفوذ از اندازه لیست پیوندی است و صحت آن

این است که نیاز به فضای بیشتری دارد چون اشاره گر ها را

دارد و به هر یک پیاده سازی آن بیشتر است

Elements(Q: queue) integer {

(4)

if (front == -1) {

return 0

}

else if (front < rear) {

return rear - front + 1

}

else {

return size - (front - rear + 1) \Leftrightarrow (max + (rear - front))

mod max

}

1 size(Q: queue) integer {

return (max + (Q.R - Q.F)) % max

}

Delete Even (Queue) queue

⑤

```
while (!is Empty()) {
```

```
    if (Q.elements [Q, R] / 2 == 0) {
```

```
        dequeue(Q)
```

```
    }
```

```
    Q.R--;
```

```
}
```

```
Return Q
```

```
}
```

```
remove-even (Q & Queue) {
```

```
    Result: Queue
```

```
    for (i in range Q.size) {
```

```
        temp = dequeue(Q)
```

```
        if (temp % 2 != 0) enqueue(Result, temp)
```

```
    return Result;
```

```
}
```


reverse(Q: queue) queue {

⑥

new Q = create();

while (!is Empty()) {

tempElement = Q.Elements[Q.R];

dequeue(Q);

Q.R--;

new Q.enqueue(tempElement);

}

return new Q;

}

def getNode(Q, index)

current, l = Q.first, 0

while i < Q.size():

if i == index: return current

current = current.next

l += 1

return None

def getReversed(Q)

rQ = Queue()

for i in range(Q.size-1, 0, -1):

rQ.enqueue(Q.getNode(i))

return rQ.

2000