


دانشکده‌ی علوم ریاضی

داده‌ساختارها و الگوریتم‌ها ۲۴ آبان ۹۳

جلسه‌ی ۱۲: داده‌ساختارهای مقدماتی

مدرّس: دکتر شهرام خزائی *نگارنده: سینا اخوان، پرهام افتخار، شادی تقیان الموتی، شروین دهقانی، سردار فتوره بنابی، کامران کوپائی و خشایار میرمحمدصادق*

مقدمه

ساختمان داده[۱] روشی خاص برای مدیریت داده‌ها در کامپیوتر است. در واقع آن را می‌توان مجموعه‌ای پویا از داده‌ها تعریف کرد. از یک ساختمان داده اعمال مختلفی انتظار می‌رود اما معمولاً امکان پشتیبانی همه‌ی این اعمال توسط یک ساختمان داده‌ی خاص وجود ندارد. تعدادی از اعمال مورد انتظار از ساختمان داده به صورت زیر است.

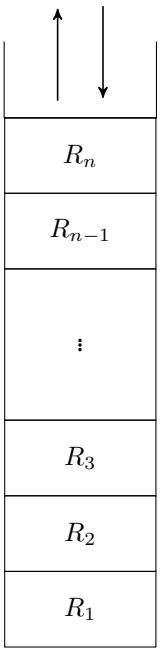
- درج یک عنصر جدید
- حذف یک عنصر
- جست‌جوی یک عنصر خاص
- کمینه‌ی عناصر موجود
- بیشینه‌ی عناصر موجود
- افزایش و کاهش کلید عناصر
- تشخیص عضو بعدی و قبلی یک عنصر خاص
- تشخیص تعداد عناصر موجود

در حالت ایده‌آل ما انتظار داریم که همه‌ی اعمال بالا در $\Theta(۱)$ انجام شود. اما $\Theta(\log n)$ نیز مطلوب است.

[۱] Data Structure

۱ پشته

پشته به مجموعه‌ای پویا گفته می‌شود که در آن عضوی که باید حذف شود از پیش مشخص شده است و آن، همان آخرین عضو درج شده است. روش عملکرد پشته را به اختصار [۲] می‌گویند.



در پشته به عمل درج PUSH و به عمل حذف POP گفته می‌شود.
پشته‌ای با حداکثر n عضو را می‌توان با استفاده از آرایه $S[1 \cdots n]$ پیاده‌سازی کنیم. اگر اندازه‌ی آرایه را با SIZE نمایش دهیم، پشته از اعضا $S[1 \cdots \text{SIZE}]$ تشکیل میشود، که $S[1]$ عضوی است که در ته پشته و $S[\text{SIZE}]$ عضوی است که در بالای پشته قرار دارد. هر یک از اعمال پشته می‌توانند با چند خط کد پیاده سازی شوند.

Algorithm 1 STACK.ISEMPTY

```
function IsEmpty(S)
    if S.size = 0 then
        return True
    else
        return False
```

Algorithm 2 STACK.ISFULL

```
function IsFull(S)
    if S.size = n then
        return True
    else
        return False
```

[۲] Last In First Out

Algorithm 3 STACK.PUSH

```
function PUSH(S, k)
    if S is not full then
        S.SIZE ← S.SIZE + 1
        S[S.SIZE] ← k
```

Algorithm 4 STACK.POP

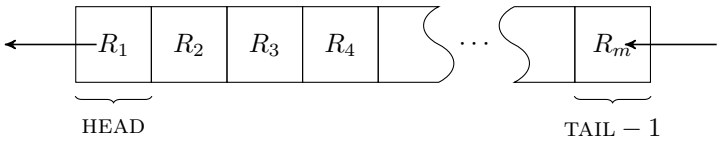
```
function POP(S)
    if S is not empty then
        x ← S[S.SIZE]
        S.SIZE ← S.SIZE − 1
        return x
```

Algorithm 5 STACK.POP

```
function POP(S)
    if S is not empty then
        S.SIZE ← S.SIZE − 1
        return S[S.SIZE + 1]
```

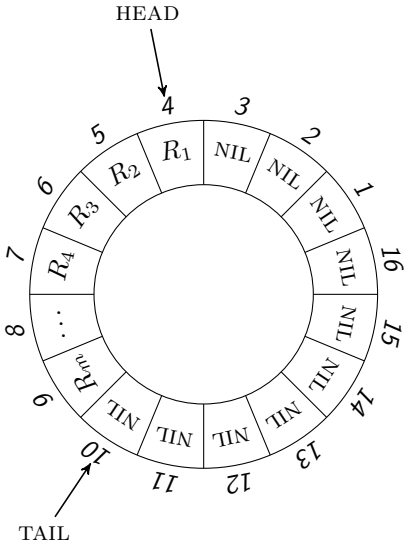
۲ صف

صف به مجموعه‌ای پویا گفته می‌شود که در آن عضوی که باید حذف شود از پیش مشخص شده است و آن، همان اولین عضو درج شده است. روش عملکرد صف را به اختصار [۳] می‌گویند.
در صف به عمل درج ENQUEUE و به عمل حذف DEQUEUE می‌گوییم.



صفی با حداکثر $n-1$ عضو را می‌توان با استفاده از آرایه $Q[1\cdots n]$ پیاده‌سازی کنیم. برای این منظور تصور می‌کنیم این آرایه مانند شکل زیر به صورت حلقوی است که درایه اول آرایه به درایه آخر آرایه متصل است.

[۳] First In First Out



صف دارای نشانگر HEAD است که ابتدای آن را مشخص می‌کند یا به آن اشاره می‌کند. نشانگر TAIL موقعیت بعد از ته صف را مشخص می‌کند که عضو تازه از راه رسیده، در آن موقعیت صف درج خواهد شد. اعضای صف در مکان‌های $\text{HEAD}, \text{HEAD}+۱, \cdots, \text{TAIL}-۱$ قرار دارند، که اگر روی صف حرکت کنیم موقعیت ۱ بلافاصله بعد از موقعیت n در ساختار حلقوی قرار می‌گیرد. وقتی $\text{HEAD} = \text{TAIL}$ صف خالی است. در ابتدا داریم $\text{HEAD} = \text{TAIL} = ۱$ و وقتی $\text{HEAD} = \text{TAIL} + ۱ \pmod n$ صف پر است. هر یک از اعمال صف می‌توانند با چند خط کد پیاده سازی شوند.

Algorithm 6 ENQUEUE

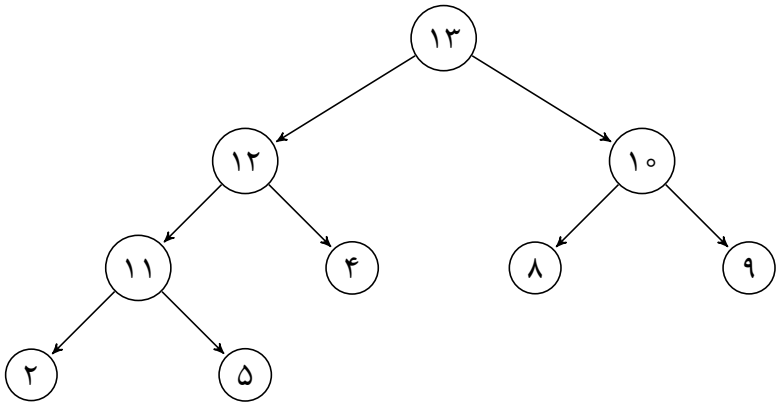
```
function ENQUEUE(Q, x)
    if Q is not full then
        Q[TAIL] ← x
        if TAIL = n then
            TAIL ← 1
        else
            TAIL ← TAIL + 1
```

Algorithm 7 DEQUEUE

```
function DEQUEUE(D, x)
    if Q is not empty then
        x ← Q[HEAD]
        if HEAD = SIZE then
            HEAD ← 1
        else
            HEAD ← HEAD + 1
        return x
```

۳ صف اولویت

مرتب‌سازی هرمی الگوریتم خوبی برای مرتب‌سازی است، ولی در عمل معمولا از مرتب‌سازی سریع استفاده می‌شود. با این وجود خود هرم کاربرد زیادی دارد و استفاده از آن به عنوان یک صف اولویت کارامد است. همانند هرم‌ها دو نوع صف اولویت وجود دارد: صف اولویت بیشینه و صف اولویت کمینه. در اینجا به بررسی صف اولویت بیشینه می‌پردازیم.



یک صف اولویت ساختمان داده‌ای برای نگه داشتن مجموعه‌ای از اعضایی است که هر یک دارای یک مقدار مربوطه به نام key است. یک صف اولویت بیشینه اعمال زیر را پشتیبانی می‌کند.

- درج: عضوی را در مجموعه درج می‌کند.
- پیدا کردن ماکزیمم: عضوی با بزرگترین کلید را بر می‌گرداند.
- پیدا کردن و حذف ماکزیمم: عضوی با بزرگترین کلید را پیدا و حذف کرده سپس آن را برمی‌گرداند.
- افزایش کلید: مقدار کلید عضوی را با مقدار جدید افزایش می‌دهد، که فرض شده مقدار جدید از مقدار فعلی کلید بیشتر است.

اکنون در مورد چگونگی پیاده سازی اعمال یک صف اولویت بیشینه با استفاده از هرم بحث می‌کنیم. هرم به صورت یک ارایه $A[۱...n]$ در نظر گرفته می‌شود که حداکثر n عنصر را می‌تواند ذخیره کند.

- تابع HEAP-MAXIMUM عمل پیدا کردن ماکزیمم را در زمان $\Theta(۱)$ پیاده‌سازی می‌کند.

Algorithm 8 HEAP-MAXIMUM

```
function HEAP-MAXIMUM(A)
    return A[1]
```

- تابع HEAP-MAX-EXTRACT عمل پیدا کردن و حذف ماکزیمم را در زمان $O(\log n)$ پیاده سازی می‌کند.

Algorithm 9 HEAP-MAX-EXTRACT

```
function HEAP-MAX-EXTRACT(A)
    if HEAPSIZE < 1 then
        error "heap underflow"
    max ← A[1]
    A[1] ← A[HEAPSIZE]
    HEAPSIZE ← HEAPSIZE − 1
    MAX-HEAPIFY(A, 1)
    return max
```

- تابع HEAP-INCREASE-KEY عمل افزایش کلید را در زمان $O(\log n)$ پیاده‌سازی می‌کند به طوری که در ابتدا کلید عضو را به مقدار جدیدش تغییر می‌دهد. چون افزایش کلید $A[i]$ ممکن است ویژگی هرم بیشینه را نقض کند آنگاه مسیری تا ریشه می‌پیماید تا جای مناسبی برای کلیدی که جدیداً افزایش یافته پیدا کند. در پی این پیمایش مکرراً یک عضو را با پدرش مقایسه می‌کند، اگر کلید عضو بزرگتر باشد، جای کلیدها را عوض کرده و ادامه می‌دهد. اگر کلید عضو کوچکتر باشد، پایان می‌یابد چون ویژگی هرم بیشینه برقرار می‌شود.

Algorithm 10 HEAP-INCREASE-KEY

```
function HEAP-INCREASE-KEY(A, i, key)
    if key < A[i] then
        error "new key is smaller than current key"
    A[i] ← key
    while i > 1 & A[PARENT(i)] < A[i] do
        A[i] ↔ A[PARENT(i)]
        i ← PARENT(i)
```

- تابع HEAP-INSERT عمل درج را در زمان $O(\log n)$ پیاده سازی می‌کند به طوری که کلید عنصر جدیدی که بایستی در هرم A درج شود را به عنوان ورودی می‌گیرد. در ابتدا با اضافه کردن یک برگ جدید به درخت که کلید آن $-\infty$ است، هرم را بسط می‌دهد. سپس HEAP-INCREASE-KEY را برای تنظیم کلید این گره جدید به مقدار صحیح خود و حفظ ویژگی هرم بیشینه فراخوانی می‌کند.

Algorithm 11 HEAP-INSERT

```
function HEAP-INSERT(A, key)
    if HEAPSIZE < n then
        HEAPSIZE ← HEAPSIZE + 1
        A[HEAPSIZE] ← −∞
        HEAP-INCREASE-KEY(A, HEAPSIZE, key)
```