ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان 99]

جلسه : هرم

نگارنده: علی شفیعی

در جلسه گذشته با داده ساختار هرم آشنا شدیم. در این جلسه الگوریتم ساختن هرم و تحلیل مرتبه زمانی آن را خواهیم دید و در آخر پیاده سازی هرم را خواهیم دید.

الگوریتم ساختن هرم

```
BuildMaxHeap(A)
A.heapSize = A.length
for i = ⌊A.length/2⌋ to 1
    MaxHeapify(A,i)
```

حال با یک تحلیل زمان ساده می‌توان نتیجه گرفت کران بالای زمان اجرای این الگوریتم $O(n\log n)$ است زیرا n بار الگوریتم $MaxHeapify(A,i)$ فراخوانی شده است و می‌دانیم زمان اجرای این الگوریتم از مرتبه $o(\log n)$ است. حال در ادامه تحلیل زمانی بهتری برای این الگوریتم بدست می‌آوریم:
می‌دانیم زمان اجرای $MaxHeapify$ بر روی یک گره به ارتفاع آن گره بستگی دارد و می‌دانیم ارتفاع اکثر گره‌ها کم است،بنابراین می‌توان انتظار داشت کران بالای بهتری برای زمان اجرا این الگوریتم بدست آوریم.همچنین می‌دانیم ارتفاع یک هرم با n عنصر برابر است با $\lfloor \log n \rfloor$ است .

لم

بیشینه تعداد گره‌های با ارتفاع h در یک هرم با n عنصر برابر است با $\lceil \frac{n}{2^{h+1}} \rceil$.
اثبات:
این لم را با استقرا بر روی h اثبات می‌کنیم. برای پایه استقرا، می‌دانیم که تعداد برگ‌های یک هرم برابر $\lceil \frac{n}{2} \rceil$ است،چون در نمایش آرایه‌ای، عناصر با اندیس i و $2i > n$ برگ هستند. فرض کنید در درخت T با n گره ، n_h تعداد گره‌ها در ارتفاع h و نیز حکم برای گره‌های با ارتفاع $h-1$ درست باشد. اگر T' با n' گره همان T باشد که برگ‌های آن را برداشته باشیم،داریم $n' = n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$ همچنین می‌دانیم که گره‌هایی که ارتفاعشان در T برابر h است، در T' در ارتفاع $h-1$ قرار دارند. بنابراین:

$$n_h = n'_{h-1} \le \lceil \frac{n'}{2^h} \rceil = \lceil \frac{\lfloor \frac{n}{2} \rfloor}{2^h} \rceil \le \lceil \frac{\frac{n}{2}}{2^h} \rceil = \lceil \frac{n}{2^{h+1}} \rceil$$

و حکم ثابت می‌شود.
حال می‌دانیم زمان مورد نیاز برای اجرای $MaxHeapify$ بر روی یک گره با ارتفاع h برابر است با $O(h)$ ، و بنابراین می‌توانیم کران بالای مرتبه زمانی این الگوریتم را به صورت زیر تعریف کنیم:

$$\sum_{h=0}^{\lfloor \log n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h) = O(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}) = O(n)$$

اثبات تساوی آخر نیز به این شکل است:

.

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$
$$\Rightarrow \frac{d}{dx} \sum_{n=0}^{\infty} x^n = \frac{1}{(1-x)^2}$$
$$\Rightarrow \sum_{n=0}^{\infty} n x^{n-1} = \frac{1}{(1-x)^2}$$
$$\Rightarrow \sum_{n=0}^{\infty} n x^{n} = \frac{x}{(1-x)^2}$$
$$\Rightarrow \sum_{h=0}^{\infty} h (\tfrac{1}{2})^{h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

بنابراین می‌توانیم در زمان خطی، یک آرایه نامرتب را به هرم بیشینه تبدیل کنیم.

حال در ادامه پیاده سازی هرم را در زبان جاوا می‌بینیم:

```java
public class Heap<T extends Comparable<T>> {

    ArrayList<T> list;

    public Heap() {
        list = new ArrayList<>();
    }

    public Heap(ArrayList<T> list) {
        this.list = list;
        heapify();
    }

    public void push(T value) {
        list.add(value);
        bubbleUp( key: list.size() - 1);
    }

    public T top() {
        return list.get(0);
    }

    public T pop() {
        T result = list.get(0);
        list.set(0, list.remove( index: list.size() - 1));
        bubbleDown( key: 0);
        return result;
    }
```

```java
private int getRightChild(int key) {
    return key * 2 + 2;
}
private int getLeftChild(int key) {
    return key * 2 + 1;
}

private int getParent(int key) {
    return (key-1)/2;
}

private boolean isRoot(int key) {
    return key == 0;
}

private boolean isLeaf(int key) {
    return !isInTree(getLeftChild(key));
}

private boolean isInTree(int key) {
    return key < list.size();
}

private void heapify() {
    for (int i = list.size() - 1; i >= 0; i--) {
        bubbleDown(i);
    }
}
```

```java
public void decreaseKey(int key, T newValue) {
    if (list.get(key).compareTo(newValue) > 0) {
        list.set(key, newValue);
        bubbleUp(key);
    }
}
```

```java
    private void bubbleUp(int key) {
        while (!isRoot(key)) {
            int parent = getParent(key);
            if (list.get(parent).compareTo(list.get(key)) < 0) {
                break;
            }
            T temp = list.get(key);
            list.set(key, list.get(parent));
            list.set(parent, temp);
            key = parent;
        }
    }

    private void bubbleDown(int key) {
        while (!isLeaf(key)) {
            int minChild = getLeftChild(key);
            if (isInTree(getRightChild(key)) &&
                    list.get(getRightChild(key)).compareTo(list.get(minChild) )< 0) {
                minChild = getRightChild(key);
            }
            if (list.get(key).compareTo(list.get(minChild))<0){
                break;
            }
            T temp=list.get(key);
            list.set(key,list.get(minChild));
            list.set(minChild,temp);
            key=minChild;
        }
    }


}
```