جزوه جلسه هفتم داده ساختارها و الگوریتم

۲۰ مهر ۱۴۰۰

فهرست مطالب

۱ درخت (Tree)

یک درخت، گرافی است که شامل چندین راس و یال است به نحوی که دور در گراف وجود ندارد و همچنین گراف همبند است. درخت ها به دو گروه تقسیم میشوند:

۱. درخت ریشه دار: درختی که یک راس به عنوان ریشه انتخاب شده و بقیه رئوس نسبت به آن اولویت پیدا میکنند. به رئوس موجود در آخرین سطح، برگ گفته میشود.

۲. درخت بدون ریشه: درختی که تمام رئوس در یک سطح هستند و هیچ راسی نسبت به راس دیگر اولویت ندارد.

در یک درخته ریشه دار، فرزندان یک راس (رئوس متصل به آن راس) میتوانند مرتب یا غیر مرتب باشند. در یک درخت مرتب تمامی عناصر در یک سطح به ترتیب از چپ به راست تکمیل هستند.

۱.۱ برخی تعاریف مهم درخت

زیر درخت یک راس: یک درخت مستقل به ریشه آن راس و سلسله مراتب فرزندانش

ارتفاع یک راس: طول بلندترین مسیر (به سمت پایین) موجود از آن راس به یک برگ

عمق یک راس: طول مسیر آن راس تا ریشه درخت

۲.۱ درخت دودویی

درختی که در آن هر راس حداکثر به دو راس دیگر متصل باشند (حداکثر تعداد فرزندان هر راس برابر دو است)

درخت دودویی کامل: درختی که در آن تمام رئوس سطوح مختلف (به جر سطح آخر یا برگ ها) به صورت مرتب پر شده اند.

درخت دودویی تقریبا کامل: درختی که در آن در سطح آخر، عناصر از ابتدا تا یک جا پر هستند.

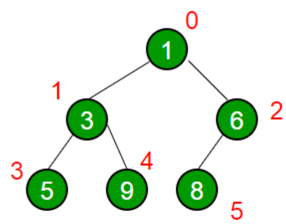
حال میتوان هرم دودویی یا Binary Heap را معرفی کرد.

۳.۱ هرم دودویی یا Binary Heap

هرم دودویی داده ساختاری است که از درخت دودویی شکل گرفته به طوری که یک درخت دودویی نسبتا کامل است و بسته به بیشینه یا کمینه بودن هر راس از فرزندان خود بزرگتر یا کوچکتر است. به کمک هرم دودویی میتوان صف اولویت را به نحوی پیدا کرد که مرتبه زمانی عملیات صف اولویت به شکل زیر باشند:

1. len(): $O(1)$
2. insert(k, v): $O(logn)$
3. find_max(): $O(1)$
4. delete_max(): $O(logn)$

برای پیاده سازی صف اولویت با هرم دودویی کافی است اعضای هیپ را مانند شکل زیر در یک آرایه ذخیره سازی کرد.



1	3	6	5	9	8
0	1	2	3	4	5

شکل ۱: هرم دودویی و شیوه ذخیره سازی آن در آرایه

هنگامی که به شیوه بالا ذخیره سازی انجام میشود دسترسی به پدر یک راس و فرزندانش به راحتی انجام میشود:

parent(i) $= \lfloor \frac{i-1}{2} \rfloor$
left_child(i) $= 2i + 1$
right_child(i) $= 2i + 2$

هر هرم دودویی، دو خاصیت دارد که میتوان اثبات کرد معادل یکدیگر هستند:

۱. هر راس از رئوس موجود در زیر درخت خود بزرگتر است.
۲. هر راس از فرزندان خود بزرگتر است.

۴.۱ مرتب سازی با صف اولویت

اگر Q یک صف اولویت باشد به کمک آن میتوان یک آرایه را به شکل زیر مرتب کرد:

```
def max_pq _sort(A):
    n = len(A)
    Q = <>
    for v in A:
        Q.insert(v)
    for i in range(n)
        A[n-1-i] = Q.delete _max()
```

در کد فوق nبار عمل insert و سپس nبار عمل delete انجام گرفته. پس مرتبه زمانی آن برابر با $O(nlogn)$ میباشد.

۵.۱ پیاده سازی عملیات تعریف شده برای صف اولویت

۱.۵.۱ len()

```
def len():
    return len(Q)
```

۲.۵.۱ find _max()

```
def find _max():
    return Q[0]
```

۳.۵.۱ insert(Q, v)

```
def insert(Q, v):
    Q.append(v)
    max _heapify _up(Q, len(Q) - 1)

def max _heapify _up(Q, i):
    if i > 0 and Q[i] > Q[parent(i):
    Q[0], Q[parent(i)] = Q[parent(i)], Q[0]
    max _heapify _up(Q, parent(i)
```

نکته: تا ارتفاع h از یک هرم دودویی حداکثر $2^{h+1} - 1$ و حداقل 2^h راس وجود دارد. همچنین عمق راس iام برابر است با $\lfloor log((i+1)/n) \rfloor$.

۴.۵.۱ **delete _max()**

```
def delete _max():
   Q[0], Q[len(Q) - 1], Q[len(Q) - 1], Q[0]
   result = Q.pop()
   max _heapify _down(Q, 0)
   return result

def max _heapify _down(Q, i):
   best = max(i, right _child(i), left _child(i))
   if best ≠ i:
       Q[i], Q[best] = Q[best], Q[i]
       max _heapify _down(Q, best)
```

برای حذف یک عنصر دلخواه نیز آنرا با آخرین برگ swap کرده و سپس برای قرار دادن عناصر در جایگاه درستشان از max _heapify _down استفاده میکنیم.

۶.۱ راه های ساخت یک **Binary Heap**

۱.۶.۱ درج تک تک عناصر

عضو جدید را به عنوان یک برگ وارد میکنیم و با max _heapify _up آنرا در جایگاه خود قرار میدهیم. پس کل عملیات در زمان $O(nlogn)$ انجام میشود.

۲.۶.۱ صدا کردن تابع **max _heapify _down** برای تمام عناصر

با فرض انجام این عملیات برای یک عنصر و پایین اوردن ان در زمان ثابت c با جمع زدن زمان برای هر راس در بدترین حالت داریم:

$$n/2 * c * 0 + n/4 * c * 1 + n/8 * c * 2 + ... \approx cn = O(n)$$

اولین جمله مربوط به برگ ها، جمله بعدی مربوط به راس ها با ارتفاع ۱ و ...