



ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان ۹۹]

سوال ۶: انتخاب سریع هرمی

نگارنده: آئیریا محمدی

الف) الگوریتم زیر را ارائه می‌دهیم

FIND K'TH BIGGEST(M: MAXHEAP, K: INT)

```
1 let m : MaxHeap
2 m.add(M.root)
3 let count = 0
4 let node : Node
5 while count < k
6     node = m.pop() // extract max
7     count = count + 1
8     m.add(node.leftchild)
9     m.add(node.rightchild)
10 return node.value
```

توضیح به این شکل عمل کردم که یک ماکس هیپ ساختم که مقداری که گره‌هایش نگهداری می‌کنند خود اشاره‌گری به گره‌های مکس هیپ اصلی است. و از ریشه شروع کردیم و هر بار آمادیم ماکسیمم این هیپ کمکی را پاپ کردیم (که می‌شود اشاره‌گر به گره با بیشترین مقدار در هیپ اصلی) و بچه‌های آن گره را به هیپ اضافه می‌کنیم برای ادامه پیمایش.

شبه‌اثبات می‌دانیم در هیپ همیشه گره پدر از فرزندانش بزرگ‌تر است. در هر مرحله وقتی بزرگ‌ترین گره را از هیپ در می‌آوریم بچه‌هایش می‌توانند پس از آن گزینه خوبی باشند برای پیمایش. البته اگر گره‌های قبلاً اضافه شده دیگر گزینه خوبی باشند طبیعتاً پیمایش از آن ادامه می‌یابد. به عبارتی بچه‌های این گره بهترین کاندیدا برای اضافه شدن هستند چون تا وقتی گره‌های دیگر دیده نشده اند فرزندانشان کاندیدای خوبی نیستند (چون پدرشان از خودشان بزرگ‌تر و در نتیجه گزینه بهتری است).

مرتب‌ه زمانی در هر دور از حلقه یک گره از هیپ کمکی کم می‌شود و دو گره فرزند آن (از هیپ اصلی) به آن اضافه می‌شود. به عبارتی تا مرحله k ام اندازه هیپ k می‌شود و در تمام این k مرحله اندازه آن کمتر مساوی k است و در نتیجه تمام عملیات‌های حذف و اضافه‌ای که روی آن انجام می‌شود از مرتبه $\log k$ می‌باشد. و با k مرحله اجرای حلقه مرتبه کل $O(k \log k)$ می‌شود.

ب) الگوریتم زیر را ارائه می‌دهیم

```

VISIT(N: NODE, K: INT, X: INT, &COUNTER: INT)
1  // notice that counter is changed globally because of reference
2  // we only wish to find k bigger nodes than x
3  if n.val >= x
4      return
5  // one more match
6  ++counter
7  if counter >= k
8      return
9  visit(n.left, k, x, ptr to counter)
10 visit(n.right, k, x, ptr to counter)

```

```

IS-BIGGER(H: HEAP, K: INT, X: INT)
1  counter : int = 0
2  visit (h.root, k, x, ptr to counter)
3  return counter == k

```

این الگوریتم زمانی پایان میابد یا k عدد بزرگتر از x پیدا شود/ یا دیگر در این پیمایش ها در تمام شاخه ها با عددی کوچکتر از x مواجه شویم. از نظر زمانی در هر شاخه یا k پیشروی می کند یا درجا متوقف می شود. به عبارتی هر گرهی که فرزندانش از x کوچک تر باشند با خودشان (که از x بزرگتر بود) سرشکن می شود و حتما به ازای هر دو گره الکی یک گره بزرگتر می بینیم و اگر گرهی از x کوچک تر باشد تمام فرزندانش نیز کوچکتر است پس نیازی به بررسی فرزندانش نیست.

از طرف دیگر اگر k عضو بزرگتر از x پیدا کنیم حتما k امین عضو بزرگتر نیز از x بزرگتر خواهد بود.

در نتیجه طی $O(k)$ عملیات می توان به پاسخ رسید.