دانشکده مهندسی کامپیوتر

جزوه درس
ساختمان‌های داده

استاد درس: سید صالح اعتمادی

پاییز ۱۳۹۸

جلسه ۱۷

صف اولویت دار

حسن صبور – ۱۳۹۸/۸/۲۵

جزوه جلسه ۱۷ام مورخ ۱۳۹۸/۸/۲۵ درس ساختمان‌های داده تهیه شده توسط حسن صبور. در جهت مستند کردن مطالب درس ساختمان های داده، بر آن شدیم که از دانشجویان جهت مکتوب کردن مطالب کمک بگیریم. هر دانشجو می‌تواند برای مکتوب کردن یک جلسه داوطلب شده و با توجه به کیفیت جزوه از لحاظ کامل بودن مطالب، کیفیت نوشتار و استفاده از اشکال و منابع کمک آموزشی، حداکثر یک نمره مثبت از بیست نمره دریافت کند. خواهش‌مند است نام و نام خانوادگی خود، عنوان درس، شماره و تاریخ جلسه در ابتدای این فایل را با دقت پر کنید.

۱.۱۷ تعریف

صف اولویت‌دار (یا صف اولویتی – Priority Queue) از جمله ساختمان داده‌های بسیار پرکاربرد است. در صف عادی از تکنیک FIFO – مخفف First In First Out – استفاده می‌شود. در این تکنیک، مثل یک صف نانوایی، داده‌ها به ترتیب ورود پشت سر هم در صف قرار می‌گیرند. بنابراین اولین داده‌ی ورودی، اولین داده‌ی خروجی نیز خواهد بود. اما در صف اولویت‌دار برای هر داده، اولویتی – نه لزوما منحصربفرد – مشخص می‌شود. صف اولویت را می‌توان به اورژانس یک بیمارستان تشبیه کرد که هر بیمار با شدت بیماری بیشتر اولویت بیشتری برای رسیدگی دارد. سیستم عامل کامپیوتر هم برای مدیریت پردازش‌ها از صف‌های اولویت‌دار استفاده می‌کند.

به عنوان مثال، فرض کنید پردازش‌های زیر در انتظار اختصاص CPU به خود هستند:

شماره پردازش	۱	۲	۳	٤	۵	٦
اولویت	٤	۲	۱	۳	۵	٤

صف انتظار CPU یک صف اولویت‌دار است. در نتیجه CPU در اولین فرصت ممکن ابتدا پردازش شماره‌ی ۳ را انجام می‌دهد. سپس پردازش شماره‌ی ۲ و ...

تذکر: روش‌های زمان‌بندی CPU جهت انجام پردازش‌های مختلف یکی از بحث‌های جذاب و در عین حال مهم مبحث سیستم عامل است. بررسی تمامی روش‌های زمان‌بندی و مزایا و معایب آنها خارج از بحث

فعلی ما است.
برای پیاده‌سازی صف اولویتی عموما از آرایه استفاده می‌شود. من در اینجا سه روش مختلف را شرح می‌دهم.

۲.۱۷ پیاده‌سازی با استفاده از آرایه‌ی نامرتب

در این روش زمانی که داده‌ای وارد صف می‌شود، همچون صف عادی در انتهای آن قرار می‌گیرد. به عنوان نمونه، داده‌های مثال زمان‌بندی CPU به صورت زیر در آرایه قرار می‌گیرند:

	1	2	3	4	5	6
شماره پردازش	1	2	3	4	5	6
اولویت	4	2	1	3	5	4

هر عنصر آرایه ساختمانی مرکب از دو عنصر شماره‌ی پردازش و اولویت آن است. هر پردازش جدید به انتهای صف اضافه می‌شود که از مرتبه‌ی O(۱) است:

	1	2	3	4	5	6	7
شماره پردازش	1	2	3	4	5	6	7
اولویت	4	2	1	3	5	4	3

اما زمانی که قرار است پردازشی از آن خارج شود، باید تک تک عناصر بررسی شوند، تا پردازشی با بیشترین اولویت انتخاب شود. این فرآیند از مرتبه‌ی O(n) است.

۳.۱۷ پیاده‌سازی با استفاده از آرایه‌ی مرتب

در این روش بر خلاف روش قبل، آرایه بر اساس اولویت‌ها مرتب شده است.

	1	2	3	4	5	6
شماره پردازش	5	6	1	4	2	3
اولویت	5	4	4	3	2	1

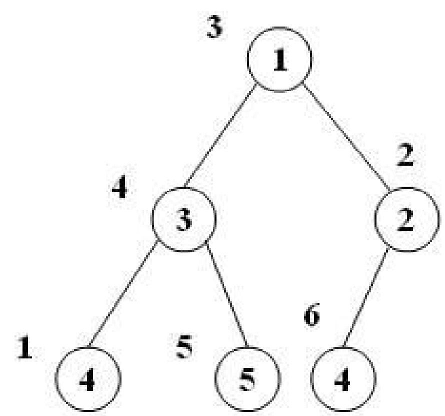
زمانی که داده‌ای وارد صف می‌شود، بر اساس اولویت خود در محل مناسب قرار می‌گیرد:

	1	2	3	4	5	6	7
شماره پردازش	5	6	1	7	4	2	3
اولویت	5	4	4	3	3	2	1

در این حالت پردازش با بیشترین اولویت همواره در انتهای صف قرار دارد و هزینه‌ی استخراج آن O(۱) است. این مسئله در مقایسه با آرایه‌ی نامرتب یک برتری است. اما در این روش هزینه‌ی درج O(n) است که در مقایسه با روش قبلی بدتر است.
در کل می‌توان گفت روش آرایه‌ی مرتب و نامرتب هم‌ارز یکدیگر بوده و از لحاظ عملکرد تفاوت چندانی با هم ندارند.

۴.۱۷ پیاده‌سازی با استفاده از آرایه‌ی نیمه‌مرتب

در این روش داده‌ها بر اساس اولویت آنها در یک درخت min–heap وارد می‌شوند:



اعداد داخل گره‌ها اولویت و اعداد خارجی شماره‌ی پردازش هستند. درخت فوق در نمایش آرایه‌ای به این صورت خواهد شد:

	1	2	3	4	5	6
شماره پردازش	3	4	2	1	5	6
اولویت	1	3	2	4	5	4

در یک درخت min–heap عنصری با کوچکترین کلید همواره در ریشه قرار دارد. در نتیجه عمل حذف گره ریشه از درخت min–heap، کوچکترین عنصر آن را به ما می‌دهد. این عمل بر اساس بحث‌های پیشین از مرتبه‌ی O(logn) است. عمل درج نیز در min–heap از همین مرتبه است.

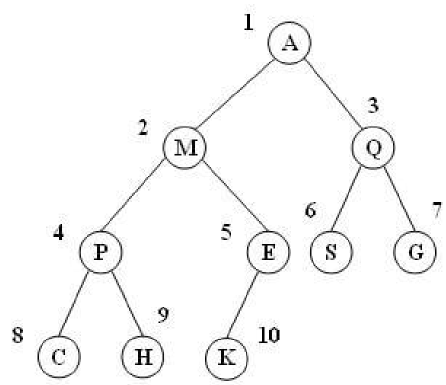
عملیات درج و حذف روی یک صف اولویتی که با استفاده از آرایه‌ی مرتب یا نامرتب ساخته شده باشد، روی هم رفته از مرتبه‌ی اجرایی n هستند. اما در روش آرایه‌ی نیمه‌مرتب این مرتبه به logn کاهش می‌یابد. پس می‌توان گفت که روش درخت هیپ برای پیاده‌سازی صف اولویتی کارآیی بسیار بهتری دارد.

۵.۱۷ heap

۱.۵.۱۷ تعریف

یک درخت دودویی کامل است، هرگاه تمامی سطوح درخت به غیر از احتمالا آخرین سطح پر بوده و برگ‌های سطح آخر از سمت چپ قرار گرفته باشند.

به یک مثال دقت کنید:



همانطور که مشاهده می‌کنید، تمامی سطوح درخت به غیر از آخرین سطح به طور کامل پر و همه‌ی برگ‌های سطح آخر نیز در سمت چپ درخت هستند. در واقع تمامی برگ‌های درخت دودویی کامل در دو سطح آخر آن قرار دارند.

۲.۵.۱۷ نمایش درخت دودویی کامل

نمایش با استفاده از لیست پیوندی و آرایه دو شکل مشهور نمایش درخت دودویی در ساختمان داده‌ها است. در حالت عادی انتخاب یکی از این دو روش برای نمایش بهینه و با مصرف حافظه‌ی کمتر بسته به چیدمان عناصر درخت دارد. به عنوان مثال، در درخت‌های مورب روش نمایش با آرایه بدترین بازدهی و بیشترین مصرف حافظه را دارد. اما در درخت دودویی کامل این روش در مقایسه با روش لیست پیوندی بسیار بهینه‌تر است.

در روش استفاده از آرایه برای نمایش درخت دودویی، گره‌های درخت مطابق شکل فوق با شروع از ریشه و در هر سطح از چپ به راست به ترتیب شماره‌گذاری شده و مقدار هر کدام از گره‌ها با توجه به شماره‌ی آن در یکی از خانه‌های آرایه قرار می‌گیرد. برای درخت فوق داریم:

1	2	3	4	5	6	7	8	9	10
A	M	Q	P	E	S	G	C	H	K

در آرایه‌ی متناظر درخت دودویی کامل، از همه‌ی عناصر به صورت کامل استفاده شده و هیچ حافظه‌ی هرزی وجود ندارد (چرا؟). به همین خاطر این روش نمایش برای درخت کامل مناسب است.

فرض کنیم توابع Parent Left، و Right شماره‌ی یک گره را گرفته و به ترتیب شماره‌ی گره والد، فرزند چپ و فرزند راست را برگرداند. در این صورت با توجه به شکل فوق:

Right(i)=۲i+۱ ، Left(i)=۲i ، Parent(i)=[i/۲]

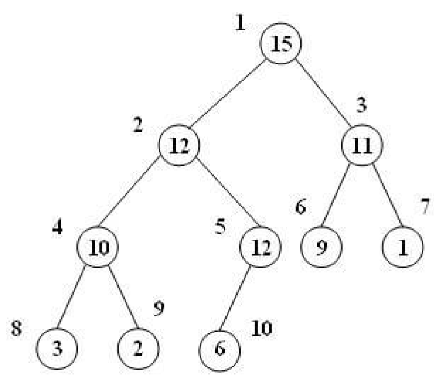
که منظور از [] جزء صحیح (کف) عدد است.

به عنوان مثال، در مورد گره شماره‌ی ۳ می‌توان نوشت:

Parent(۳)=[۳/۲]=۱،Left(۳)=۲×۳=۶،Right(۳)=۲×۳+۱=۷

۳.۵.۱۷ max heap

درخت دودویی کاملی است که مقدار هر گره بیشتر یا مساوی فرزندان خود است.

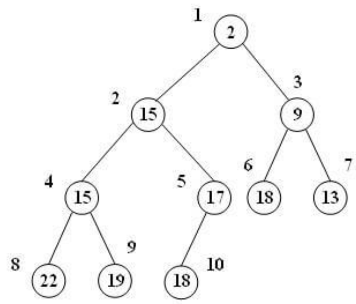


و نمایش آرایه‌ای:

	1	2	3	4	5	6	7	8	9	10
Max Heap	15	12	11	10	12	9	1	3	2	6

۴.۵.۱۷ min heap

درخت دودویی کاملی است که مقدار هر گره کمتر یا مساوی فرزندان خود است.



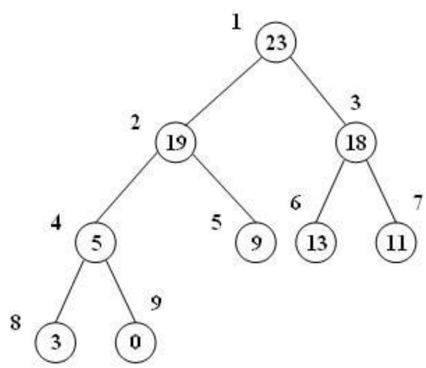
۵.۵.۱۷ ساختن heap

ساختن یک درخت heap در واقع وارد کردن متوالی گره‌ها در آن است. برای وارد کردن یک گره به درخت heap، طی دو مرحله به صورت زیر عمل می‌کنیم:

۱- گره مفروض را در محلی از درخت که شرط کامل بودن آن به هم نخورد (بدون در نظر گرفتن شرط max-heap یا min-heap بودن) درج می‌کنیم.

۲- اگر گره مذکور بر اساس موقعیت خود در درخت، شرط max-heap یا min-heap بودن را نقض نکند، نیاز به انجام کاری نیست و عملیات درج تمام شده است. در غیر اینصورت، با جابجا کردن گره با والد خود، درخت جدیدی حاصل می‌شود که باید مرحله‌ی ۲ در مورد آن تکرار شود.

به عنوان مثال، فرض کنید یک درخت max-heap به فرم زیر داریم:

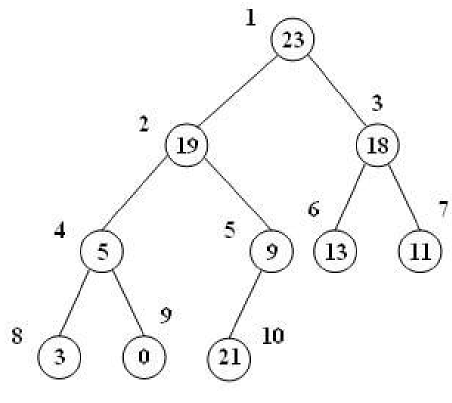


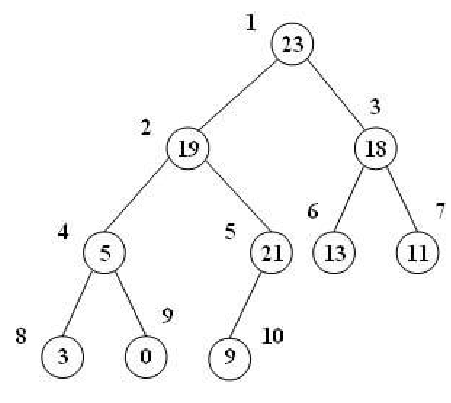
حال می‌خواهیم گرهی با مقدار ۲۱ را به درخت اضافه کنیم. برای اینکار در مرحله‌ی اول گره مذکور را به محلی که شرط کامل بودن درخت نقض نشود وارد می‌کنیم. این محل سمت چپ‌ترین فضای آزاد آخرین سطح درخت است:

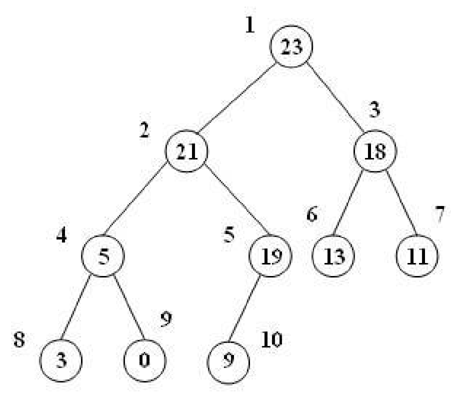
با درج این گره، شرط max-heap بودن نقض می‌شود. چرا که مقدار گره شماره‌ی ۱۰ از والد خود یعنی گره شماره‌ی ۵ بیشتر است. پس با توجه به دستورالعمل مرحله‌ی دوم، مقدار دو گره را جابجا می‌کنیم:

با این عمل، باز هم شرط max-heap بودن برآورده نمی‌شود. گره‌های شماره‌ی ۵ و ۲ این شرط را نقض کرده‌اند. پس باز هم با تکرار مرحله‌ی دوم مقدار این دو گره را با هم جابجا می‌کنیم:

حال شرط max-heap بودن برقرار بوده و عملیات درج گره تمام می‌شود.







با توجه به این مثال می‌توان مرحله‌ی دوم عملیات درج را اینگونه بیان کرد:

۲- گره درج شده را با والدهای خود تا جایی که شرط max-heap یا min-heap بودن برقرار شود جابجا می‌کنیم.

۶.۵.۱۷ برنامه‌نویسی درج گره در درخت heap

در اینجا کد مربوط به درج گره در درخت max-heap را می‌آورم که با یک تغییر جزئی همین کد برای درخت min-heap هم قابل استفاده است.

همانطور که بحث شد، بهترین روش نمایش درخت heap استفاده از آرایه است. در مورد درخت max-heap اولیه فوق داریم:

	1	2	3	4	5	6	7	8	9
Max Heap	23	19	18	5	9	13	11	3	0

با اضافه کردن گره ۲۱ و طی کردن مراحل دوگانه درج گره:

	1	2	3	4	5	6	7	8	9	10
Max Heap	23	19	18	5	9	13	11	3	0	21

	1	2	3	4	5	6	7	8	9	10
Max Heap	23	19	18	5	21	13	11	3	0	9

	1	2	3	4	5	6	7	8	9	10
Max Heap	23	21	18	5	19	13	11	3	0	9

در قسمت قبلی رابطه‌ی ریاضی بین اندیس‌های والد و فرزند بیان شده است. بر اساس این رابطه و توضیحات فوق، تابع درج گره با مقدار v در یک درخت max-heap که در حال حاضر n عنصر (گره) دارد در زبان C++ به این صورت خواهد بود:

```
void push(int heap[], int &n, int v){
    int i, temp;
    heap[++n] = v;
    for(i = n ; i > 1 && heap[i] > heap[i / 2] ; i /= 2){
        temp = heap[i];
        heap[i] = heap[i / 2];
        heap[i / 2] = temp;
    }
}
```

تذکر ۱: اندیس آرایه‌ها در زبان برنامه‌نویسی C++ از صفر شروع می‌شود. اما در اینجا برای راحتی کار و هماهنگ شدن با روش شماره‌گذاری درخت دودویی کامل، از اولین خانه - یعنی خانه‌ی شماره‌ی صفر - برای نمایش درخت heap استفاده نشده است.

تذکر ۲: در این تابع پارامتر n به صورت مرجع تعریف شده است که مختص زبان برنامه‌نویسی C++ بوده و در زبان C وجود ندارد. متغیرهای مرجع در یک نوشته به طور کامل توضیح داده شده است.

نکته: روش درج گره جدید در درخت heap در ظاهر شباهت‌هایی به درج گره جدید در درخت جستجوی دودویی (Binary Search Tree) دارد. اما این دو اختلاف‌های مشخصی دارند:

نکته: با توجه به قطعه کد بالا، مرتبه‌ی اجرایی عمل درج در درخت heap از مرتبه‌ی O(logn) است.

۷.۵.۱۷ حذف گره از درخت Heap

حذف گره از درخت هیپ عموما از ریشه‌ی آن صورت می‌گیرد. حذف گرهی غیر از گره ریشه، ممکن است هزینه‌ای معادل ساخت مجدد درخت تحمیل کند. چرا که با حذف یک گره غیر ریشه و جایگزین کردن گرهی دیگر با آن، نه تنها شرط heap بودن که شرط درخت کامل بودن هم ممکن است نقض شود. اکثر کاربردهای این نوع درخت نیز تنها با حذف گره از ریشه سر و کار دارند.

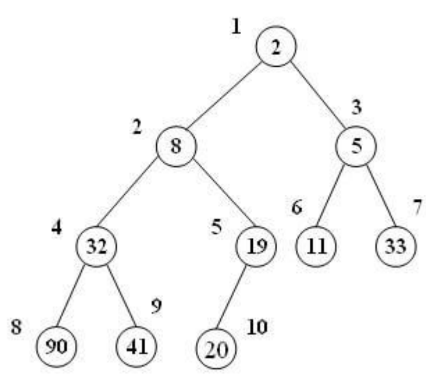
برای حذف گره ریشه‌ی درخت دو مرحله زیر را اجرا می‌کنیم:

۱- گره ریشه را حذف و سمت راست‌ترین برگ سطح آخر را جایگزین آن می‌کنیم.

۲- در صورتی که گره درج شده جدید شرط heap بودن را نقض نکند عملیات حذف تمام می‌شود. در غیر اینصورت این گره با فرزند مناسب جایگزین شده و این مرحله برای درخت جدید مجددا اجرا می‌شود.

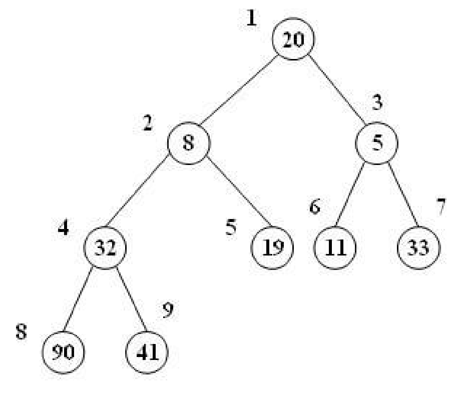
با اجرای مرحله‌ی اول و جایگزین کردن آخرین گره آخرین سطح درخت، شرط کامل بودن درخت پایدار می‌ماند. اما عموما شرط heap بودن نقض می‌شود. در مرحله‌ی دوم، گره تازه وارد را با یکی از فرزندان خود جایگزین می‌کنیم، تا به شرط heap بودن نزدیک شویم. اما کدام فرزند؟ پاسخ را با یک مثال مشخص می‌کنم.

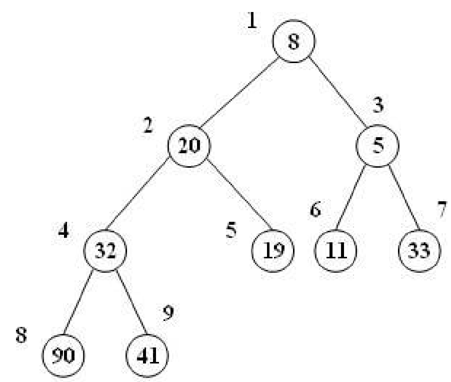
فرض کنید قصد داریم گره ریشه‌ی درخت min-heap زیر را حذف کنیم:



مرحله‌ی اول را اجرا کرده و گره شماره‌ی ۱۰ را جایگزین ریشه می‌کنیم: شرط درخت کامل بودن همچنان برقرار است. اما درخت فعلی min-heap نیست. چرا که ریشه از هر دو فرزند خود بزرگتر است. حال مطابق مرحله‌ی دوم باید یکی از فرزندان را با والد جابجا کنیم.

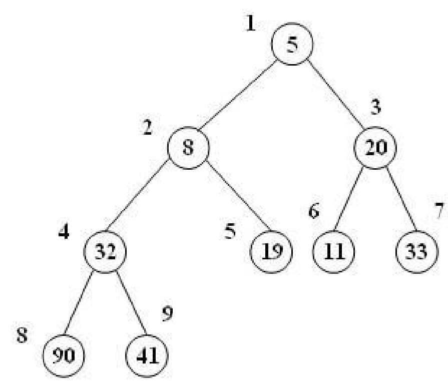
اگر فرزند چپ با مقدار ۸ را انتخاب کنیم:



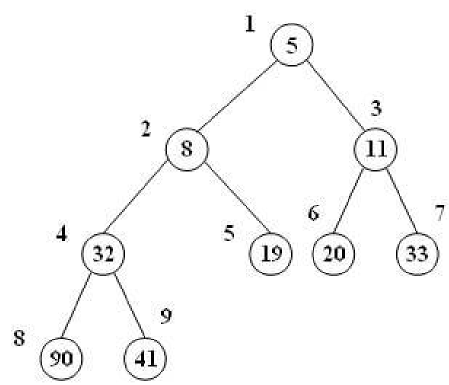


در این حالت، علاوه بر بحث مکان درست گره شماره‌ی ۲ با مقدار ۲۰، مشکل دیگری هم داریم: گره شماره‌ی ۱ و ۳ هم شرط min-heap را نقض می‌کنند.

اگر فرزند راست را انتخاب می‌کردیم:



در این حالت لااقل گره‌های شماره‌ی ۱ و ۲ مشکلی ندارند و تنها دغدغه‌ی ما محل درست گره شماره‌ی ۳ خواهد بود. پس نتیجه اینکه: در درخت min-heap، فرزندی را جایگزین والد می‌کنیم که مقدار کوچکتری داشته باشد. این مسئله در مورد max-heap به صورت عکس است. یعنی فرزندی را در درخت max-heap جایگزین می‌کنیم که مقدار بیشتری دارد.
اما هنوز گره شماره‌ی ۳ شرط min-heap بودن را نقض می‌کند. پس با تکرار مرحله‌ی دوم و با توجه به نتیجه‌گیری فوق، این گره را با گره شماره‌ی ۶ جابجا می‌کنیم:



به این ترتیب شرط min-heap بودن نیز برقرار شده و عملیات حذف گره به اتمام می‌رسد.

۸.۵.۱۷ برنامه‌نویسی حذف گره از درخت heap

این عملیات برای درخت فوق در نمایش آرایه‌ای به فرم زیر خواهد شد:

	1	2	3	4	5	6	7	8	9	10
Min Heap	2	8	5	32	19	11	33	90	41	20

	1	2	3	4	5	6	7	8	9
Min Heap	20	8	5	32	19	11	33	90	41

	1	2	3	4	5	6	7	8	9
Min Heap	5	8	20	32	19	11	33	90	41

	1	2	3	4	5	6	7	8	9
Min Heap	5	8	11	32	19	20	33	90	41

بر اساس روابط ریاضی بین شماره‌ی اندیس گره‌های والد و فرزند، تابع pop برای حذف گره ریشه به این ترتیب خواهد بود:

```
int pop(int heap[], int &n){
    int i = 1, result, temp, min;
    result = heap[1];
    heap[1] = heap[n--];
    while(2 * i <= n){
        min = 2 * i;
        if(min + 1 <= n && heap[min + 1] < heap[min])
            min++;
        if(heap[i] <= heap[min])
            break;
        temp = heap[i];
        heap[i] = heap[min];
        heap[min] = temp;
        i = min;
    }
    return result;
}
```

؟؟

Bibliography