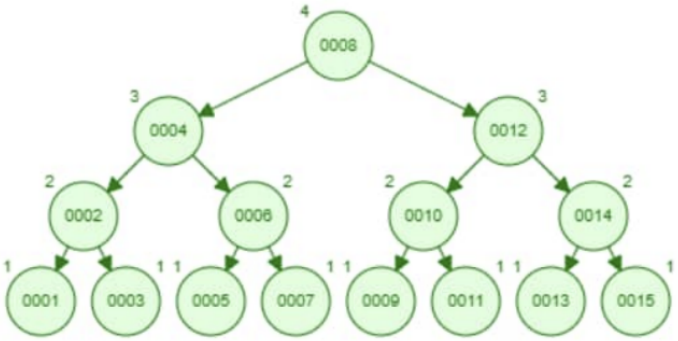
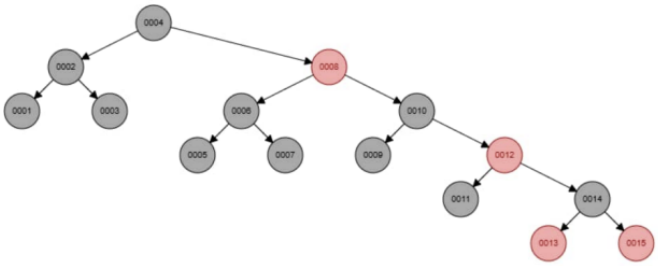
سوال ۱.۱

اعداد ۱ تا ۱۵ را به ترتیب در درخت قرمز سیاه و درخت AVL درج کنید و تمامی مراحل را قدم به قدم مشخص کنید.

پاسخ:





سوال ۲

فرض کنید یک مهمانی برگزار می شود که در آن n نفر با شماره های ۰ تا n-۱ شرکت می کنند. تعداد نامحدودی صندلی از ۰ تا بی نهایت برای این مهمانی در نظر گرفته شده است. هر مهمان به محض ورود، روی اولین صندلی خالی (با کمترین شماره) می نشیند. در صورتی که مهمانی محل را ترک کند، صندلی او بلافاصله آزاد می شود و نفر بعدی می تواند روی آن بنشیند.
به شما یک آرایه دو بعدی از اعداد صحیح داده می شود. شماره اندیس هر عنصر نشان دهنده شماره ورود یک مهمان است و هر عنصر از آرایه شامل دو عدد است که به ترتیب نشان دهنده زمان ورود و خروج یک مهمان هستند. همچنین، یک عدد صحیح به عنوان targetFriend به شما داده می شود که نشان دهنده شماره مهمانی است که می خواهیم شماره صندلی او را بیابیم. توجه داشته باشید که زمان های ورود مهمانان همگی متفاوت هستند. الگوریتم خود را برای حل این مساله ارایه دهید و پیچیدگی زمانی آن را محاسبه کنید.

مثال

ورودی:
$\text{times} = [[1,4],[2,3],[4,6]]$
$\text{targetFriend} = 1$
خروجی:
1

پاسخ:

در ابتدا داده ها را بر اساس زمان ورودشان مرتب می کنیم (البته باید توجه کرد که شماره هر مهمان ذخیره شود.) سپس شماره صندلی های خالی را در یک minheap با عنوان صندلی در دسترس ذخیره می کنیم. داده های مرتب شده را به ترتیب پیمایش می کنیم و در هر مرحله ریشه ی minheap (صندلی های در دسترس) حذف می شود و به عنوان صندلی رزرو شده در minheap دیگری با عنوان صندلی های رزرو شده بر اساس زمان خروج هر مهمان درج می کنیم. و اگر ریشه ی این minheap (صندلی رزرو شده) از زمان ورود مهمان بعدی کوچک تر یا مساوی بود. ریشه را حذف می کنیم و شماره صندلی در minheap (صندلی در دسترس) صندلی ای است که این مهمان می تواند روی آن بنشیند. پیچیدگی زمانی این الگوریتم از $\mathcal{O}(n \log n)$ می باشد.

```
۱       func (times,targetfriends)
۲       info=Array()
۳       j=0
۴       for i in times:
۵               info[j]=Friends(i[0],i[1],index)
۶       info.sort(lambda x:x.arrive)
۷       used_chairs=MinHeap()
۸       available_chairs=MinHeap()
۹       for i in info:
۱۰              a=info[]i].arrive
۱۱              l=info[i].leave
۱۲              while used_chairs and used_chairs.root.leave <= a:
```

```
                    leave,chair=used_chairs.delMin()
                    available_chairs.insert(chair)
            chair=available_chairs.delMin()
            used_chars.insert((l,chair))

    if i== targetfriends:
            return chair
```

سوال ۳.۱

عبارت بالا را به prefix و postfix تبدیل کنید.

پاسخ:

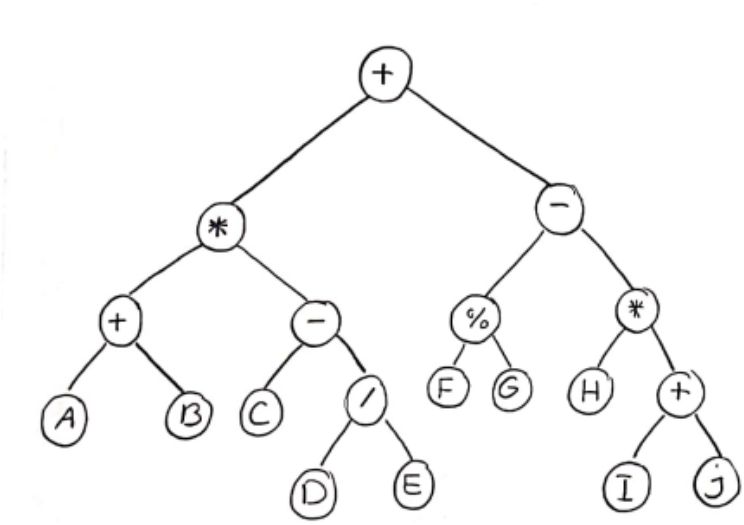
prefix : +*+AB-C/DE-%FG*H+IJ

postfix : AB+CDE/-*FG%HIJ+*-+

سوال ۳.۲

برای عبارت بالا یک درخت عبارت رسم کنید.

پاسخ:



سوال ۴.۱

یک درخت دودویی به شما داده شده است، بررسی کنید آیا این درخت یک max heap است یا خیر. الگوریتم خود را به همراه شبه کد ارایه کنید و هزینه زمانی آن را محاسبه کنید.

پاسخ:

برای بررسی MaxHeap بودن یک درخت دودویی باید ۲ شرط زیر را بررسی کرد. ۱) درخت کامل باشد و ۲) هر گره از فرزندانش بزرگتر باشد. با استفاده از پیمایش میان ترتیب در هر سطح گره را با فرزندانش مقایسه می کنیم. با استفاده از متغیر flag بودن یا نبودن فرزندان هر گره را بررسی می کنیم. پیچیدگی زمانی $\mathcal{O}(n)$

```
is_heap(root)
        q=Queue()
        q.enqueue(root)
        flag=False
        while q is not empty:
                node=q.dequeue()
                if node.left:
                        if flag or node.left.data > node.data:
                                return False
                        q.enqueue(node)
                else:
                        flag =True
                        if node.right:
                                if flag or node.right.data > node.
                                    ↪ data:
                                        return False
                                q.enqueue(temp.right)
                        else:
                                flag=True
        return True
```

سوال ۴.۲

چگونه می توان یک max heap را تبدیل به min heap کرد، الگوریتم خود را به همراه شبه کد ارایه دهید و هزینه زمانی آن را محاسبه کنید.

پاسخ:

برای تبدیل یک MaxHeap به MinHeap چندین روش وجود دارد. اما بهینه ترین روش ساخت یک MinHeap جدید با استفاده از الگوریتم ساختن Heap است. که پیچیدگی زمانی این الگوریتم از $\mathcal{O}(n)$ است.

```
build_heap(A):
        for i= length(A)/2 down to 1:
                Min_Heapify(A,i)
```

```
interative_postorder(node):
        postorder=Array()
        i=0
        if node is none:
                return postorder
        S1=Stack()
        S2=Stack()
        while S1 is not empty:
                node=S1.pop()
                S2.push(node)
                if root.left:
                        S1.push(node.left)
                if root.right:
                        S1.push(node.right)
                while S2 is not empty:
                        postorder[i]=S2.top
                        i++
                        S2.pop()

        return postordder
```

```
interative_preorder(node):
        preorder=Stack()
        if root is none:
                return preorder
        S=Stack()
        S.push(node)
        while S is not empty:
                node=S.pop()
                preorder.push(node.data)
        if node.right:
                S.push(node.right)
        if node.left:
                S.push(node.left)
        return preorder
```

سوال ۵

شما یک آرایه از وظایف CPU داده شده است که هر کدام با یک حرف از A تا Z برچسب گذاری شده اند. هر فاصله زمانی، CPU می تواند بیکار باشد یا اجاره تکمیل یک وظیفه را بدهد. وظایف را می توان به هر ترتیبی تکمیل کرد. اما یک محدودیت وجود دارد، باید حداقل n فاصله زمانی بین دو وظیفه با برچسب یکسان وجود داشته باشد. الگوریتمی برای به دست آوردن حداقل تعداد فاصله های زمانی CPU مورد نیاز برای تکمیل همه وظایف ارایه دهید. پیچیدگی زمانی الگوریتم خود را محاسبه کنید.

مثال

ورودی:
$\text{tasks} = [\text{"A"},\text{"A"},\text{"A"},\text{"B"},\text{"B"},\text{"B"}]$
$n=2$
خروجی:
8 توضیحات:
A -> B -> idle -> A -> B -> idle -> A -> B

پاسخ:

در ابتدا تعداد هر کاراکتر را شمارش و ذخیره می کنیم. سپس داده ها را در یک max-heap درج می کنیم. در هر مرحله ریشه max-heap حذف می شود و مقدارش را یک واحد کم می کنیم. اگر مقدارش صفر نشد آن را در یک صف درج می کنیم. داده ای که در صف درج شده است باید حداقل تا n فاصله ی زمانی صبر کند و سپس از آن صف حذف شده و دوباره در max-heap درج شود. این روند را تا زمانی که صف خالی نیست ادامه می دهیم.

سوال ۶

اعداد ۱ تا ۱۷ را به ترتیب در یک درخت جستجوی دودویی درج کنید. پس از درج میخواهیم این درخت را به یک درخت جستجوی دودویی متوازن تبدیل کنیم. ارتفاع درخت در حالت متوازن چقدر است؟ به چند چرخش نیاز است تا درخت متوازن شود؟ تمامی چرخش ها را مرحله به مرحله نمایش دهید و الگوریتم خود را با جزییات شرح دهید.

پاسخ:

درخت داده شده را با پیمایش میان ترتیب، پیمایش می کنیم که برای ما دنباله ای مرتب شده از داده ها را تولید می کند. سپس به صورت بازگشتی عنصر وسط دنباله را پیدا می کنیم و درخت جدید را درست می کنیم. هزینه زمانی از $\mathcal{O}(n)$ است.

سوال ۷

یک کامپیوتر مورد حملات DDos قرار گرفته است. این حمله از طرف کامپیوتر های دیگر در اینترنت صورت می گیرد. هر کامپیوتر در شبکه جهانی اینترنت با استفاده از یک آدرس IP شناخته می شود. این آدرس شامل چهار عدد مثبت ۰ تا ۲۵۵ است که در نمایش با نقطه از یکدیگر جدا شده اند.(به عنوان مثال 113.30.185.12)

شما به عنوان متخصص ساختمان داده باید به یک مهندس امنیت کمک کنید تا برای مسدود کردن کامپیوتر های دیگر یک ساختمان داده کارآمد طراحی کند.

این ساختمان داده باید دو عملیات زیر را تعریف کند.

- بلاک کردن یک IP آدرس: یک آدرس IP به شما داده می شود و باید آن را به مجموعه آدرس های بلاک شده اضافه کنید.
- بررسی بلاک بودن IP آدرس: بررسی می کند که این آدرس IP قبلا بلاک شده است یا خیر.

پاسخ:

در این پیاده سازی، یک درخت Trie به کار می رود که در آن هر گره نمایانگر یک بخش از آدرس IP است . برای هر آدرس IP که مسدود می شود، یک گره جدید در درخت ایجاد می شود تا آدرس مربوطه در آن ذخیره شود. سپس در گره نهایی، فیلدی به نام is_blocked قرار می دهیم که نشان می دهد این آدرس مسدود شده است یا خیر. زمانی که یک آدرس IP مسدود می شود، ابتدا آن را به بخش های جداگانه تقسیم می کنیم. سپس در درخت Trie به طور مرحله به مرحله جرکت می کنیم تا هر بخش از آدرس را به گره مربوطه اظافه کنیم. در نهایت، در گره مربوط به آخرین بخش، is_blocked را برابر با True قرار می دهیم. برای بررسی مسدود بودن از ریشه ی Trie شروع می کنیم و با پیمایش گره ها، به دنبال این آدرس می گردیم. اگر به گره ای برسیم که نشان دهنده ی is_blocked برابر با True باشد یعنی آدرس IP مسدود شده است.