Data Structures and algorithms
Instructor : **Dr.Amin Eskandari**
Head-Ta's : Hamid Namjoo manesh , Amir Hossein Hemati , Ali Mojahed
Ta's : Amir Mohammad Asadjoo , Padina razmjooi , Armin Janfaza , Alireza Heydari , Mahdi Torabi ,
Maryam Ghorbani , Fatemeh Keshavarz , MohammadReza Fasli , Abolfazl Tadrisi

Quiz 03 Answers

پاسخنامه آزمون شماره ۰۳

پاسخ سوال اول :

بخش ۱ – صف بی‌قانون

الف)از **صف دوطرفه‌ی دوری (Circular Deque)** با آرایه استفاده می‌کنیم. دو متغیر اصلی:

- first: اندیس ابتدای صف
- num: تعداد عناصر موجود

اضافه‌کردن به جلو:

اول
first = (first - 1) % max_size

سپس q[first] = x

حذف از عقب:
فقط num = num - 1 (عنصر انتهایی به طور ضمنی حذف می‌شود.)

📚 منبع: چیت شیت فصل 4 (اَی یادتون باشه، شب قبل از کوییز گفتیم "چیت شیت ها رو هم بخونید 😁" برای چیت شیت ها خیلی زحمت کشیده شده و بخشی از آپدیت جزوه‌ای ترم(4042) بود که با تیمی از عزیزان (اسمشون رو در اول جزوه و چیت شیت ها میبینید) حدود 2 ماه کل پروژه‌ی آپدیت به طول انجامید.)

ب) برای Undo یک **Stack** (پشته) با آرایه (مطابق جزوه) نگهداری می‌کنیم. هر بار که یک عمل push/pop روی صف انجام می‌شود، یک رکورد شامل (نوع_عملیات, مقدار) در پشته push می‌کنیم.

هنگام ()undo:

- رکورد بالای پشته را pop کن.
- اگر عملیات push_front بود، یک pop_front روی صف انجام بده.
- اگر pop_back بود، مقدار را دوباره با push_back برگردان. (و مشابه برای بقیه).
  پیچیدگی آن O(1) می‌باشد.

📚 **منبع: چیت شیت فصل 4**

ج)

مرحله	عملیات	صف (اول ←دوم)
1	push_back(10)	[10]
2	push_back(20)	[10, 20]
3	push_front(5)	[5, 10, 20]
4	pop_front()	[10, 20]
5	undo() ← reverse pop_front → push_front(5)	[5, 10, 20]
6	push_back(30)	[5, 10, 20, 30]
7	push_front(99)	[99, 5, 10, 20, 30]
8	pop_back()	[99, 5, 10, 20]
9	undo() ← reverse pop_back → push_back(30)	[99, 5, 10, 20, 30]

محتوای نهایی: [99, 5, 10, 20, 30]

بخش ۲ – لیست VIP با Trie

الف) درج: "lulu", "luka", "yak", "yakut"

root

├── l → u → l → u* (پایان lulu)

│ └── k → a* (پایان luka)

└── y → a → k* (پایان yak)

└── u → t* (پایان yakut)

گره‌های mark شده با * نشان داده شده‌اند.

📚 **منبع:** بخش 3 فصل 5 (Trie)

ب)برای پشتیبانی از پیشوند، مطابق مثال انتهای جزوه(3-5)، **هنگام درج، تمام گره‌های مسیر** **mark = True** **را می‌کنیم** (نه فقط گره پایانی). آن‌گاه هر گره‌ی mark شده نشان‌دهنده‌ی یک پیشوند معتبر است.

تابع starts_with(prefix):

node = root

for ch in prefix:

idx = ord(ch) - ord('a')

if node.edges[idx] is None: return False

```
    node = node.edges[idx]
return node.mark
```

پیچیدگی: O(|prefix|)

📚 **منبع: فصل 5 بخش 3**

ج)

- find("yaku"):

اگر از درج معمولی (فقط علامت‌گذاری انتهای کلمه) استفاده کنیم، گره‌ی u در مسیر y-a-k-u-t مارک **نشده**، بنابراین **False**عه!

- starts_with("lu"):

اگر از روش علامت‌گذاری **تمام مسیر** استفاده شود، گره‌ی l-u مارک است بنابراین True هس!

📚 **منبع: فصل 5 قسمت 3**

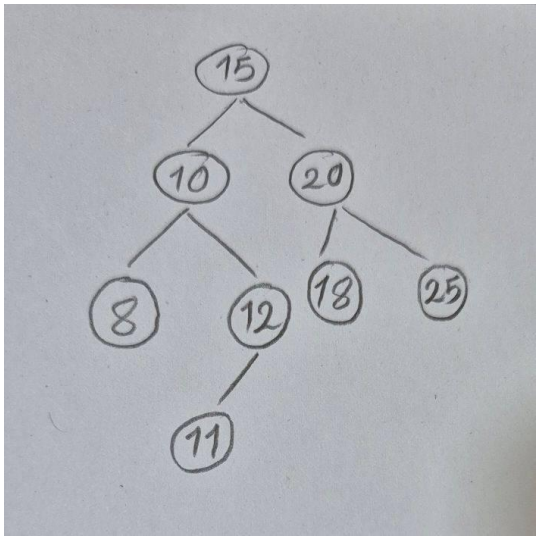
بخش ۳ – انبار پیچ با BST

الف)

- آرایه نامرتب: درج O(1) ، جستجو O(n)
- آرایه مرتب: جستجوی دودویی O(logn) ، درج O(n)
- BST: در حالت میانگین هر دو O(logn) هست، پس بین سرعت جستجو و درج مصالحه (بالانس) ایجاد می‌کند.

📚 **منبع: فصل 4 (ایضاً مثال پیچ و مهره‌ی جزوه)**

ب) درخت:



حذف: 10 (گره با دو فرزند)

طبق جزوه، جانشین inorder (کوچکترین گره‌ی زیردرخت راست) را می‌یابیم: 11

مقدار 10 را با 11 جایگزین کرده، سپس گره 11 (که برگ یا یک فرزند دارد) را حذف می‌کنیم.

📚 **منبع: فصل 5 بخش 2**

پ) LCA در BST: گره‌ای که یکی از دو مقدار در چپ و دیگری در راست آن قرار بگیرد (یا خود یکی از مقادیر باشد).

شبه‌کد:

```
lca(root, x, y):
    if root == null: return null
    if x < root.label and y < root.label:
        return lca(root.left, x, y)
```

```
else if x > root.label and y > root.label:
    return lca(root.right, x, y)
else:
    return root
```

- LCA(11,18):

از ۱۵ شروع: ۱۵>۱۱ ولی ۱۵<۱۸ ← در دو سمت مختلف، LCA = 15

- LCA(8,25):

۱۵>۸ و ۱۵<۲۵ ← باز هم میشه ۱۵.

📚**منبع: چیت شیت 2-5 و جزوه اصلی 2-5**

ت) اگر داده‌ها صعودی درج شوند، BST به یک **لیست خطی** (درخت اُریب) تبدیل می‌شود. ارتفاع n و جستجو O(n)

در دنیای واقعی با **AVL Tree** یا **Red-Black Tree** (درخت متوازن) رفع می‌شود.

📚**منبع: فصل 5 قسمت 2 (+چیت شیتش)**

بخش ۴ – Max-Heap

الف) آرایه از اندیس **1**: [18, 41, 63, 27, 50, 14, 32, _] (اندیس ۰ گفتیم که خالیه)

ساخت هرم با Bubble-Down از n//2 = 3 به سمت ۱:

- i=3(مقدار 50): فرزندان 41 و 18 – بزرگترین 50 است، تغییری نمی‌کند.

- (14): i=2 فرزندان 27 و 63 – 63 ← 14 <جابجایی 14 و 63 → آرایه [_, 32, 63, 50, 27, 14, 41, 18]. حال 14 در 5، فرزندی ندارد.
- (32): i=1 فرزندان 63 و 50 ← 63 ← 32 <جابجایی 32 و 63 → [_, 63, 32, 50, 27, 14, 41, 18]. حال 32 در 2، فرزندان 27 و 14 – تغییری نمی‌کند.

هیپ نهایی: [63, 32, 50, 27, 14, 41, 18] (به صورت درخت سطح‌بندی‌شده)

📚 **منبع: چیت شیت 4-5**

ب)

- extract_max: اول 63 را با 18 (آخرین) جابه‌جا، حذف 63، سپس 18 در ریشه Bubble-Down میشه و با 50 جابه‌جا در نهایت داریم: [50, 32, 41, 27, 14, 18]
- extract_max: دوم 50 با 18 جابه‌جا، حذف 50، Bubble-Down 18 با 41 جابه‌جا و در نهایت داریم: [41, 32, 18, 27, 14]

📚 **منبع: چیت شیت 4-5**

ج) در روش پایین به بالا، **نیمی از گره‌ها برگ هستند** و عملاً ۰ جابه‌جایی دارند. یک‌چهارم گره‌ها نهایتاً ۱ جابه‌جایی، یک‌هشتم ۲ جابه‌جایی و ... مجموع هزینه:

$$n/4 * 1 + n/8 * 2 + \ldots = O(n)$$

📚 **منبع: چیت شیت 4-5**

بخش ۵ – تاریخچه دما

الف) ساختار مناسب **لیست پیوندی دوطرفه‌ی حلقوی با گره نگهبان (Sentinel)** است.

- Head: گره‌ای با داده None که next و prev آن **به خودش** اشاره می‌کنند.

- مزیت: درج/حذف در ابتدا O(1)، برخلاف آرایه که نیاز به جابجایی O(n)دارد.

📚 **منبع: فصل 4 (+ کمی دانش خارج از جزوه)**

ب) درج در ابتدا (بعد از sentinel):

```
new_node = Node(data)
new_node.prev = head
new_node.next = head.next
head.next.prev = new_node
head.next = new_node
```

حذف گره x:

```
x.prev.next = x.next
x.next.prev = x.prev
```

پیچیدگی هر دو O(1)

📚 **منبع: و باز هم نهایتا همه چیز به فصل 4 ختم میشه** 😁

پاسخ سوال دوم :

بیایید اول به سراغ بخش الف برویم. شاید بپرسید چطور می‌توان فهمید که چند درخت دودویی میتوانیم بکشیم؟

پاسخ ساده است. با داشتن یک پیمایش با n نود و دانستن اعداد کاتالان!

اما اعداد کاتالان چیستند؟

اعداد کاتالان (Catalan Numbers) دنباله‌ای از اعداد صحیح مثبت هستند که در بسیاری از مسائل شمارش مبحث ترکیب ظاهر می‌شوند. این اعداد انواع خاصی از مسیرهای مشبک، درخت های باینری و بسیاری از اشیاء ترکیبی دیگر را شمارش می‌کنند. اعداد کاتالان در یک رابطه بازگشتی پایه صدق می‌کنند و یک فرمول بسته برحسب ضرایب دوجمله‌ای دارند.

این اعداد با فرمول زیر تعریف می شوند:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

که در آن n عدد صحیح نامنفی است.

در زمینه درختان دودویی، عدد کاتالان C_n تعداد درختان دودویی متمایز با n گره را نشان می دهد.

در ابتدا گفته شد که نیاز به داشتن یک پیمایش با n گره هستیم تا بتوانیم با استفاده از اعداد کاتالان تعداد درخت های متمایز را محاسبه کنیم.

با نگاه به هر یک از پیمایش های داده شده متوجه می شویم که ۱۲ نود داریم. حالا بیاید با استفاده از فرمول اعداد کاتالان تعداد درخت های متمایز را محاسبه کنیم :

$$C_{12} = \frac{1}{12+1}\binom{2 \times 12}{12}$$

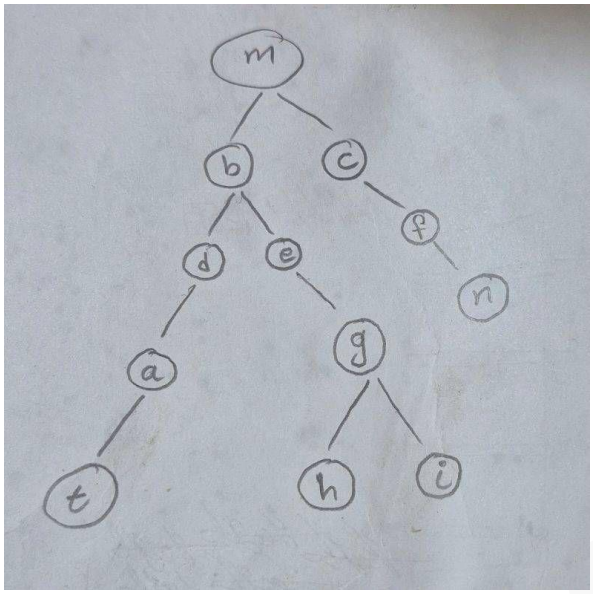
$$C_{12} = \frac{1}{13} \times \frac{(24)!}{(12)!\,12!} = \frac{1}{13} \times 2{,}704{,}156 = 208{,}012$$

بنابراین، **۲۰۸,۰۱۲**درخت دودویی متمایز با ۱۲ گره وجود دارد.

ب) بازسازی درخت:

- گره ریشه، گره m می باشد زیرا : (اول Pre و آخر Post)
- فرزند چپ m، b است و فرزند راست c هست . با توجه به Post که (... b ... c m)
- ادامه به صورت بازگشتی

درخت نهایی:



تعداد:

- **برگ‌ها** ← t, h, i, n : ۴ برگ.
- **دوفرزندی‌ها** ← m, b, g : ۳ گره.
- **تک‌فرزندی‌ها** ← d, a, e, c, f : ۵ گره (مجموعاً ۱۲ گره).

ج) برگ‌ها : t, h, i, n

دوفرزندی‌ها : m, b, g

📚 **منبع:** فصل 1-5 (+چیت شیتش)

پاسخ سوال سوم :

اول به سراغ مورد ۱ می رویم :

با نگاه به ساختار این رابطه متوجه می شویم که برای بدست آوردن مرتبه آن می توانیم از قضیه اصلی استفاده کنیم :

$$a = 16\,, b = 2\,, f(n) = 3n^3 - 5 \rightarrow n^{log_2^{16}} = n^{log_2^{2^4}} = n^4\ ?\ n^3 \rightarrow T(n) \in \theta(n^4)$$

پس مورد اول با توجه به حالت اول قضیه اصلی حل می شود.

به سراغ مورد دوم سوال می رویم.

این مورد نیز با استفاده از قضیه اصلی قابل حل می باشد.

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3 \log n \rightarrow a = 4\,, b = 2\,, f(n) = n^3 \log n \rightarrow n^{log_2^4} = n^2 ? n^3 \rightarrow T(n) \in \theta(n^3 log n)$$

و در نهایت برای مورد آخر نیز می توان از قضیه اصلی استفاده کرد :

$$a = 9\,, b = 3\,, f(n) \epsilon \theta(n^2) \rightarrow n^{log_3^9} = n^2 ? n^2 \rightarrow T(n) \in \theta(n^2 \log n)$$

پاسخ سوال چهارم :

از آنجایی که رشته های ما همگی متشکل از اعداد 0 و 1 هستند میتوانیم بگوییم که اگر عدد 1 آمد بروید به سمت راست و اگر 0 آمد به سمت چپ می رویم.

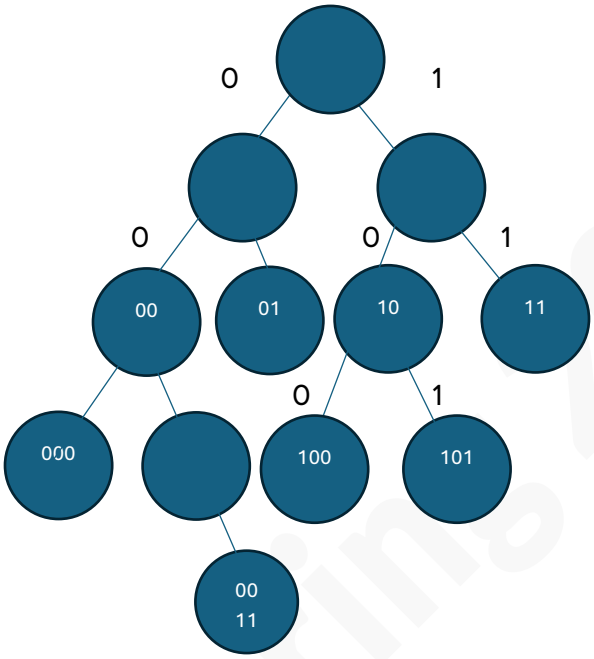
حال بیایید ابتدا درخت را رسم کنیم و از عدد 100 شروع می کنیم.

ابتدا یک گره را رسم می کنیم و از آنجایی که عدد 100 با 1 شروع می شود ابتدا به سمت راست سپس چپ و سپس چپ می رویم .

حالا به سراغ رشته 101 می رویم.پس ابتدا به سمت راست سپس چپ و پس از آن به سمت راست می رویم.

برای رشته 01 ابتدا به سمت چپ سپس راست.

به همین ترتیب به سراغ دیگر گره ها می رویم تا درخت پایانی را رسم کنیم :



نکته ای که باید در نظر داشت این است که اگر درخت Trie را به صورت Pre-order پیمایش کنید. به صورتی که فقط آن گره هایی که درونشان داده هست را چاپ کنیم. گویی Dictionary sort را انجام داده ایم ت که مرتبه یافتن آن $\theta(n)$ است که n مجموع طول همه رشته ها می باشد.