

## نمونه سوالات مربوط به بخش ۱.۱

۱.۱.۱ با توجه به الگوریتم های مرتب سازی حبابی و مرتب سازی انتخابی و تحلیل این الگوریتم ها مزایای مرتب سازی درجی را نسبت به این دو روش مرتب سازی بیان کنید.

راه حل : به طور معمول بسته به درجهٔ مرتب بودن آرایه ، مرتب سازی مقایسه های کمتری نسبت به مرتب سازی انتخابی انجام می دهد . در حالی که مرتب سازی انتخابی باید قسمت باقی ماندهٔ آرایه را هنگام قرار دادن عنصر جدید جستجو کند . مرتب سازی درجی تنها عناصر را به تعداد لازم جست و جو میکند . این یعنی زمانی که آرایه مرتب شده است یا تا حد خوبی نزدیک به مرتب شده است . مرتب سازی درجی رشد زمانی متناسب با  $n$  خواهد داشت . در رابطه با زمان میانگین مرتب سازی درجی در مقابل مرتب سازی انتخابی ، این نکته مهم است که در مرتب سازی انتخابی تعداد نوشتن ها (جابجایی ها) متناسب با  $n$  است درحالی که در مرتب سازی درجی این تعداد متناسب با  $n^2$  است . در برخی شرایط به ازای حد  $n$  به سمت بی نهایت نسبت زمان اجرای هر دو مرتب سازی گفته شده نسبت به مرتب سازی درجی واگرا خواهد بود و به بی نهایت میل خواهد کرد . البته در بدترین حالت هر سه ضریبی از  $n^2$  خواهد بود و نسبتشان در حد گفته شده همگرا خواهد بود از طرفی تغییر نکردن زمان تناسب زمان این دو مرتب سازی اخیر با  $n^2$  در بهترین حالت علاوه بر عوامل دیگر باعث می شوند که به ازای  $n$  های کوچک مرتب سازی درجی بر این دو روش ترجیح داده شود .

---

۱.۱.۲ در برنامه‌ی مرتب سازی درجی که پیاده سازی شد ، در هر مرحله ، پیدا کردن محل قرار گیری یک عنصر به کمک حلقه while به دست آمد . این قسمت را به نحوی تغییر دهید که جای عنصر جدید را با جست و جوی دودویی پیدا کند تحلیل زمانی برنامه را با این تغییر برای آرایهٔ دلخواه ورودی با اندازهٔ  $n$  انجام دهید .

راه حل : در مرتب سازی درجی معمولی از جست و جوی خطی برای یافتن محل قرار گیری عنصر استفاده می شد که در بدترین حالت تعداد مقایسه ها  $O(n)$  بود ولی اکنون که از جست و جوی دو دویی استفاده کردیم برای پیدا کردن محل قرار گرفتن عنصر حداقل  $O(\log n)$  هزینه باید بپردازیم بنابر این اگر هزینهٔ مقایسه ها مهم باشد مرتب سازی درجی با استفاده از جست و جوی دودویی برای ما ارجح است . هزینه کلی مرتب سازی هم  $O(n \log n)$  می شود .

```
intbinarySearch(inta[], intitem, intlow, inthigh)
{
    if(high <= low)
        return(item > a[low])? (low + 1): low;

    intmid = (low + high)/2;

    if(item == a[mid])
        returnmid+1;

    if(item > a[mid])
        returnbinarySearch(a, item, mid+1, high);
    returnbinarySearch(a, item, low, mid-1);
}
```

```

int i, loc, j, k, selected;

for(i = 1; i < n; ++i)
{
    j = i - 1;
    selected = a[i];

    // find location where selected should be inserted
    loc = binarySearch(a, selected, 0, j);

    // Move all elements after location to create space
    while(j >= loc)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = selected;
}

```

---

## نمونه سوالات مربوط به بخش ۱,۲

۱,۲,۱ الگوریتمی با هزینه  $O(n \log n)$  طراحی کنید که آرایه  $S$  با عناصر ثابت و عدد  $x$  را بگیرد و ببیند دو عددی که مجموعشان  $x$  می شود در مجموعه پیدا می شود یا خیر؟

راه حل : می توان ابتدا با **merge sort** آرایه را مرتب کرد و سپس به ازای هر عنصر  $x-a[i]$  ،  $a[i]$  را محاسبه کرد و اگر حاصل بزرگتر از صفر بود ، با استفاده از **binary search** فهمید آیا  $x-a[i]$  در مجموعه وجود دارد یا خیر . در کل  $n \log n$  هزینه ای مرتب کردن داریم و در بدترین حالت  $n$  بار هزینه ای **binary search** داریم . در کل هزینه ای کلی  $n \log n$  می شود .

---

## ۱,۲,۲ رفتار تابع $F$ به صورت زیر است :

$$F(i,0) = F(i+1,0) + F(i+1,1) \quad \text{if } i < n$$

$$F(i,1) = 2*F(i+1,0) + F(i+1,1) \quad \text{if } i < n$$

$$F(n,0) = 1, F(n,1) = 0$$

اگر بخواهیم مقدار  $F(1,1)$  را به طور مستقیم به دست آوریم رفتار مجانبی تعداد عمل جمع (+) را به دست آورید.

راه حل :

$$F(n,0) + F(n,1) = F(n-1,0)$$

$$2F(n,0) + F(n,1) = F(n-1,1)$$

$$F(n-1,0) + F(n-1,1) = F(n-2,0)$$

$$2F(n-1,0) + F(n-1,1) = F(n-2,1)$$

...

$$F(2,0) + F(2,1) = F(1,0)$$

$$2F(2,0) + F(2,1) = F(1,1)$$

تعداد عمل های جمع برابر  $(n-1)^*2^m$  می باشد که  $O(n)$  می باشد.

۱,۲,۳ مسئله‌ی یافتن تعداد وارونگی (نابجایی) : فرض کنید آرایه‌ای دلخواه از اعداد مانند A با n عنصر یکتا داریم ، می گوییم در آرایه نابجایی رخداده است اگر  $j < i$  و  $a[i] > a[j]$  باشد الف) الگوریتمی ارائه دهید که تعداد این وارونگی (نابجایی) را بشمارد. ب) اگر قرار باشد از یک خط لیست شروع کنیم و از آن خط تا آخر لیست و بعد از اول لیست تا آن جا به ترتیب حرکت کنیم ، الگوریتمی ارائه دهید که حداقل تعداد نابجایی‌ها بین تمام حالات را از روی قسمت الف محاسبه کند (یعنی ما تعداد نابجایی‌ها را در قسمت الف برای آرایه پیدا کردیم حال اگر از عنصر دیگر آرایه شروع به پیمایش کنیم تعداد نابجایی جدید را چگونه می‌توان از عدد قسمت الف به دست آورد). ج) حداکثر تعداد نابجایی‌ها در مجموعه‌ی  $\{1, 2, 3, \dots, n\}$  در چه شرایطی اتفاق می‌افتد و تعدادش چه قدر است ؟

راه حل :

الف ) فرض کنیم تعداد نابجایی‌ها در نیمه‌ی اول  $inv_1$  و در نیمه‌ی دوم  $inv_2$  باشد در این صورت تنها حالاتی که محاسبه نشده اند حالاتی اند که بین دو نیمه‌ی اول و دوم رخ می‌دهند یعنی نابجایی‌هایی که باید در مرحله‌ی ادغام شمرده شوند. در مرحله‌ی ادغام (i) اندیس مربوط به زیر آرایه‌ی سمت چپ و (j) اندیس مربوط به زیر آرایه‌ی سمت راست است. اگر  $a[i] > a[j]$  باشد و اندیس میانی mid داشته باشیم آنگاه mid-i نابجایی خواهیم داشت چون همه‌ی عناصر سمت راست  $a[j], a[i], \dots, a[1]$  از  $a[i]$  بزرگتر خواهند بود (مرتب شده اند) بنابراین با استفاده از الگوریتم مرتب سازی ادغامی و پرداختن هزینه‌ی  $\log n$  می‌توان تعداد نابجایی‌ها را محاسبه کرد .

ب) می‌توان با هزینه‌ی  $O(n)$  مینیمم تعداد نابجایی در تمام حالات را به دست آورد به طوری که اگر تعداد نابجایی‌ها در مرحله

قبل count باشد (حالت اول هم از قسمت الف به دست آمد) به ازای هر عنصر  $i$   $a[i] < i$  تعداد نابجایی‌های جدید برابر با

$$count = count - ((a[i-1]-1)-(size of array - a[i-1]))$$

و در هر مرحله هم مینیمم گیری انجام می‌دهیم تا در نهایت جواب مورد نظر به دست آید .

ج ) حداکثر تعداد نابجایی‌ها زمانی اتفاق می‌افتد که آرایه مرتب نزولی باشد و تعداد آن  $\binom{n}{2}$  می‌باشد .

### نمونه سوالات مربوط به بخش ۱,۳

۱,۳,۱ توابع زیر را به ترتیب درجهٔ رشد مرتب کنید.

$$\ln \ln n, \log^* n, n^{2^n}, n^{\log \log n}, \ln n, \sqrt{\log n}, n^2, (\log n)!$$

راه حل :

$$\log^* n < \ln \ln n < \sqrt{\log n} < \ln n < n^2 < (\log n)! < n^{\log \log n} < n^{2^n}$$


---

۱,۳,۲ توابع زیر را بر حسب پیچیدگی با ذکر دلیل مرتب کنید.

$$\sum_{i=1}^n i 2^i, n^n, \log n, \sqrt[10]{n}, n!, \sum_{i=0}^{\frac{n}{2}} n - 2i, \log^* n, \sum_{i=0}^n i^3, \log(n!), (\log n)!, 2^n$$

راه حل :

$$\sum_{i=0}^{\frac{n}{2}} n - 2i = 2 + 4 + \dots + n = 2 * \frac{\frac{n}{2} * \frac{n+2}{2}}{2} = O(n^2)$$

$$\begin{aligned} \sum_{i=1}^n i 2^i &= 1 * 2 + 2 * 2^2 + 3 * 2^3 + 4 * 2^4 + \dots + n * 2^n \\ &= (2 + 2^2 + \dots + 2^n) + (2^2 + 2^3 + \dots + 2^n) + \dots + 2^n \\ &= 2^{n+1} - 2^1 - 1 + 2^{n+1} - 2^2 - 1 + 2^{n+1} - 2^3 - 1 + \dots + 2^{n+1} - 2^n - 1 \\ &= n * 2^{n+1} - 2^{n+1} - n = O(n * 2^n) \end{aligned}$$

$$\sum_{i=0}^n i^3 = \left(\frac{n * (n+1)}{2}\right)^2 = O(n^4)$$

$$O(n!) = O(n^n) > \sum_{i=1}^n i 2^i > 2^n > (\log n)! > \sum_{i=0}^n i^3 > \sum_{i=0}^{\frac{n}{2}} n - 2i > \log n! > \sqrt[10]{n} > \log n > \log^* n$$

۱,۳,۳ توابع مجانبی را برای توابع زیر به دست آورید.

$$T(n) = \sum_{i=1}^{\log n} \frac{(i+1)(n+1)}{2^{i+1}} \quad \text{الف)$$

$$T(n) = \sum_{i=1}^n \log_2 \left( \frac{n}{i} \right) \quad \text{ب)$$

راه حل:

(الف)

$$T(n) \leq \sum_{i=1}^{\lfloor \log n \rfloor} \left( \frac{n+1}{2^{i+1}} \right) (i+1) = (n+1) \sum_{i=1}^{\lfloor \log n \rfloor} \frac{i+1}{2^{i+1}} = (n+1) \sum_{i=2}^{\lfloor \log n \rfloor + 1} \frac{i}{2^i} < (n+1) \sum_{i=2}^{\infty} \frac{i}{2^i}$$

$$\text{Since } \left( \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \dots \right) = 2 \quad \rightarrow T(n) \approx 2(n+1)$$

(ب)

$$\sum_{i=1}^n \log_2 \left( \frac{n}{i} \right) < \int_0^n \log_2 \left( \frac{n}{i} \right) di = n \log n - n(\log n - 1) = n$$

نمونه سوالات مربوط به بخش ۱,۴

۱,۴,۱ برای تابع

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + 1 & \text{if } n \text{ is even} \\ 2 * T\left(\frac{n-1}{2}\right) & \text{if } n \text{ is odd} \end{cases}$$

الف) نشان دهید به تعداد نامتناهی  $n$  وجود دارد به طوری که  $T(n) = \Omega(n)$

ب) نشان دهید به تعداد نامتناهی  $n$  وجود دارد به طوری که  $T(n) = O(\log n)$

راه حل:

(الف)

$$n=2^k-1$$

$$T(n)=T(2^k-1)=2*T(2^k-2/2)=2*T(2^{k-1}-1)$$

$$S(k)=S(k-1) \quad S(2)=1$$

$$S(k)=2^{k-1} \quad T(2^k-1)=2^k-1$$

$$T(n)=\Omega(n)$$

(ب)

$$n=2^k$$

$$T(n)=T(2^k)=T(2^{k-1})+1$$

$$S(k)=S(k-1)+1 \quad S(0)=1$$

$$S(k)=k+1 \quad T(2^k)=k+1$$

$$T(n)=O(\log n)$$

۱،۴،۲ یک مثال نقض برای هر کدام از ادعا ها بیابید .

الف) اگر  $f(n)-g(n)=O(s(n)-r(n))$  آنگاه  $g(n)=O(r(n))$  و  $f(n)=O(s(n))$

ب) اگر  $\frac{f(n)}{g(n)}=O\left(\frac{s(n)}{r(n)}\right)$  آنگاه  $g(n)=O(r(n))$  و  $f(n)=O(s(n))$

راه حل :

الف)

$$f(n)=2n, g(n)=n \rightarrow f(n)-g(n)=2n-n=n \neq O(n-n)=O(1)$$

(ب)

$$f(n)=n^3 \rightarrow O(n^3), g(n)=n^2=O(n^3) \rightarrow \frac{f(n)}{g(n)}=n \neq O(1)$$

۱،۴،۳ توابع  $f(n)$  و  $g(n)$  را چنان مثال بزنید که  $f(n) \neq O(g(n))$  و  $g(n) \neq O(f(n))$

راه حل :

$$f(1)=g(1)=1$$

$$f(n)=\begin{cases} g(n-1)^2 + f(n-1) & \text{if } n \text{ is even} \\ f(n-1) + 1 & \text{if } n \text{ is odd} \end{cases}, g(n)=\begin{cases} g(n-1) + 1 & \text{if } n \text{ is even} \\ f(n-1)^2 + g(n-1) & \text{if } n \text{ is odd} \end{cases}$$

برای  $n$  های زوج ،  $f(n)^2 = O(g(n)^2)$  و برای  $n$  های فرد  $f(n) = O(g(n)^2)$  است .

۱،۴،۴ با فرض اینکه  $f(n)$  و  $g(n)$  توابعی نامنفی باشند ، رابطه‌ی زیر را ثابت کنید :

$$\max(f(n), g(n)) = \Theta(f(n)+g(n))$$

راه حل :

$$\forall n > \text{Max}\{f(n), g(n)\} \leq f(n) + g(n)$$

$$\forall n > 2 * \text{Max}\{f(n), g(n)\} \geq f(n) + g(n) \quad \text{Max}\{f(n), g(n)\} > \frac{1}{2}(f(n)+g(n))$$

$$\forall n > \frac{1}{2}(f(n)+g(n)) < \text{Max}\{f(n), g(n)\} < f(n) + g(n) \rightarrow n_0 = 1, c_0 = \frac{1}{2}, c_1 = 1$$

---

اگر  $f(n) = \Omega(\log n)$ ,  $g(n) = O(n)$ ,  $f(n) = \omega(n)$ ,  $f(n) = O(n^2)$  محدوده عبارات زیر را بیابید.

$$\begin{array}{lcl} f(n) \times g(n^2) & \bullet \\ \frac{f(\log n)}{g(n)} & \bullet \\ f(\sqrt{n}) \times g(2^n) & \bullet \end{array}$$

راه حل :

$$\bullet \quad f(n) \times g(n^2) = O(n^4) = \omega(n \log n)$$

$$g(n^2) = O(n^2) = \Omega(\log(n^2)) = \Omega(2 \log n) = \Omega(\log n)$$

$$\bullet \quad \frac{f(\log n)}{g(n)} = O(\log n) = \omega\left(\frac{\log n}{n}\right)$$

$$\frac{1}{g(n)} = O\left(\frac{1}{\log n}\right) = \Omega\left(\frac{1}{n}\right)$$

$$\bullet \quad f(\sqrt{n}) \times g(2^n) = O(n 2^n) = \omega(n \sqrt{n})$$

## نمونه سوالات مربوط به بخش ۱,۵

۱,۵ زمان اجرای هریک از تکه کدهای زیر را بیابید.

a)

```
for (int i=1; i<n; i++)
    for (int j=1; j<n; j++)
        i*=2;
```

راه حل :

خط دوم : بار چک خواهد شد. پس از هر دفعه چک شدن (جز دفعه‌ی آخر) نه دوباره می‌شود. (یعنی  $n-1$  بار) بنابراین دفعه‌ی دومی که شرط  $i < n$  چک می‌شود  $n-1$  خواهد بود و برنامه خاتمه می‌یابد.  
زمان مورد نیاز در صورت برابر بودن زمان اجرای هر خط  $n+n-1+2=2n+1$  خواهد بود.

b)

```
int k = 0, j = 0;
for (int i=n; i>j; i--)
    for(j= 0; j<2i; j++)
        while ((j+k) %2==1)
            k++;
```

در این قسمت نیز محقق نشدن شرط خط سوم برقراری شرط خط دوم را بر هم می‌زند. پس خط اول یک بار و خط دوم تنها دوبار اجرا خواهد شد.

خط سوم نیز تنها یک بار تا دوباره نه خواهد رفت که در آن حالت  $i=n$  است پس  $2n+1$  بار خط سوم اجرا خواهد شد.  
شرط خط چهارم به ازای مقدار صفر  $j=0$  نادرست است. پس خط پنجم اجرا نمی‌شود و خط چهارم یکبار اجرا می‌شود ولی پس از آن با افزوده شدن یکی به  $j$ ، مجموع  $j$  و  $k$  فرد شده و شرط برقرار می‌شود و پس خط چهارم دوبار و خط پنجم یکبار اجرا می‌شود و دوباره مجموع  $j$  و  $k$  زوج می‌شود و این روند با هر بار زیاد شدن  $j$  تکرار می‌شود بنابراین خط چهارم  $2^{2n-1}$  بار اجرا می‌شود و خط پنجم  $2n-1$  بار تکرار خواهد شد.  
پس در مجموع زمان اجرای کل برنامه  $8n+2$  خواهد بود.

c)

```
for (int i=0; i<n; i++)
    for (int j=1; j<i*i; j*=2);
```

در این برنامه خط دوم به ازای هر  $i$ ، سقف  $\log_2 i^2 + 1$  بار انجام می‌شود (یافتن کوچکترین توان دوی بزرگتر مساوی  $i^2$  با شروع از صفر) پس زمان اجرای کل برنامه برابر است با :

$$n + 1 + 2 + \sum_{i=2}^{n-1} (\lceil \log_2 i^2 \rceil + 1) = n + 3 + n - 2 + \sum_{i=2}^{n-1} \lceil 2 \log_2 i \rceil \approx 2n + 1 + 2 \log_2 \left( \prod_{i=2}^{n-1} i \right)$$

$$= 2n + 1 + 2 \log_2 (n-1)! \xrightarrow{\text{استرلينگ}} 2n + 1 + 2n \log n$$

که هزینه‌ی کلی  $O(n \log n)$  می‌شود.

d)

```
for (int i=2*n; i>0; i--)
    for (int j=2*i; j>i/2; j--);
```

در این قطعه کد خط دوم نیز به تعداد

$$\sum_{i=2n}^1 (2i - \left\lceil \frac{i}{2} \right\rceil + 1) = 2n + \sum_{i=2n}^1 (3(\frac{i}{2}) + \{\frac{i}{2}\}) = 2n + \frac{3*2n*(2n+1)}{2*2} + \frac{n}{2} = 3n^2 + 4n$$

و خط اول به تعداد  $2n+1$  بار اجرا می شود ، بنابراین زمان اجرای کل الگوریتم  $3n^2+6n+1$  خواهد بود .

## نمونه سؤالات مربوط به بخش ۲

۱.۲ فرض کنید آرایه‌ی  $n$  بعدی  $W[u_1][u_2][u_3] \dots [u_n]$  را داریم. فرمولی برای محاسبه‌ی آدرس خانه‌ی  $w[i_1][i_2][i_3] \dots [in]$  ارایه دهید. (فرض کنید هر int یک بایت است و آدرس خانه‌ی اول آرایه ۰ است)

راه حل:

$$i_1(u_2 \times u_3 \times \dots \times u_n) + i_2(u_3 \times u_4 \times \dots \times u_n) + \dots + i_{n-1}(u_n) + i_n$$

۲.۲ فرض کنید یک ماتریس  $n \times n$  از اعداد صحیح داریم. که  $k$  عنصر آن صفر است.  
الف) روشی برای نگه داری این ماتریس در آرایه‌ی چند بعدی ارایه دهید که حافظه‌ای برای صفرها مصرف نشود.

ب) این روش در چه صورتی بهتر از استفاده از آرایه‌ی  $n \times n$  است؟

ج) روشی برای ترانهاده کردن ماتریس حالت الف ارایه دهید.

راه حل:

الف) از آرایه‌ی  $3 \times n$  بعدی استفاده میکنیم این‌طوری که شماره‌ی سطر و ستون و مقدار هر عدد غیر صفر را نگه داری میکنیم.

ب) اگر  $n \times n > 3 \times (n-k)$

ج) شماره‌ی سطر و ستون هر خانه را جایه‌جا میکنیم.

## نمونه سؤالات مربوط به بخش ۳

۱.۳ با استفاده از یک یا چند پشته‌ی جدیدی طراحی کنید که علاوه بر تابع‌های push و pop و top تابع دیگری (get\_min) داشته باشد که کوچکترین عنصر پشته را بر می‌گرداند. هزینه‌ی انجام تمام عملیات باید  $O(1)$  باشد. (توجه: فقط اجازه‌ی استفاده از استک را دارید)

راه حل:

از دو استک استفاده میکنیم. استک اصلی و استک کمکی. متدهای push و pop را بازسازی میکنیم.

برای push این‌طوری عمل میکنیم: Push

۱. عنصری که میخواهیم push کنیم ( $x$ ) را در استک اصلی push میکنیم.

۲.  $x$  را با top کمکی ( $y$ ) مقایسه میکنیم اگر  $y < x$  بود  $x$  را در استک کمکی push میکنیم اگر نه  $y$  را push میکنیم.

Pop: هر دو استک را pop میکنیم و عنصر اول استک اصلی را بر می‌گردانیم.

Top: عنصر اول استک اصلی را بر می‌گردانیم.

get\_min: عنصر اول استک کمکی را بر می‌گردانیم.

مثالاً برای دنباله‌ی ۱۸ ۱۹ ۴ ۸ ۱۰ ۱ ۹

استک اصلی می‌شود ۱۸ ۱۹ ۴ ۸ ۱۰ ۱ ۹

استک کمکی می‌شود ۱۸ ۱۸ ۴ ۴ ۴

۲.۳ با استفاده از تابع بازگشتنی عناصر یک پشته را مرتب کنید. (اجازه‌ی استفاده از حلقه را ندارید)

راه حل:

```
sort(stack s){  
    if (s.is_empty())  
        return;  
    int temp=s.pop();  
    sort(s);  
    insert(temp, s);  
}
```

```
insert(int element, stack s){
```

```

if(s.is_empty() or s.top()<element)
    s.push(element);
else {
    int temp=s.pop();
    insert(element, s);
}

```

**۳.۳ یک رشته(string)** از پرانتز باز و پرانتز بسته داده شده است. الگوریتمی بهینه ارایه دهد که طول بزرگترین زیررشته ای که پرانتز های مجاز دارد را پیدا کند. (راهنمایی هزینه (n) است)

#### راه حل:

یک پشته و یک متغیر result در نظر میگیریم.

۱. عدد -۱ را به عنوان base در استک push میکنیم.

۲. به ازای هر کاراکتر از رشته

الف) اگر کاراکتر پرانتز باز بود index آن را در پشته push میکنیم

ب) اگر کاراکتر پرانتز بسته بود استک pop میکنیم

ب.۱. اگر پشته خالی نبود اختلاف index کاراکتر حاضر با top استک طول زیررشته ی

مجاز است که این عدد اگر از result بیشتر بود result را به روز میکنیم.

ب.۲. اگر پشته خالی بود (یعنی base را pop کرده‌ایم) کاراکتر جدید را به

عنوان base جدید در پشته push میکنیم.

برای مثال: رشته ی زیر را در نظر بگیرید:

```

)((()))
stack: [-1]
result=0

```

حالت ب.۱)

stack: [0]

result=0

حالت الف)

stack: [0,1,2,3]

result=0

حالت ب.۲)

(طول رشته ی مجاز = ۴-۲)

result=2

حالت الف)

stack: [0,1,2,5]

result=2

حالت ب.۱)

(طول رشته ی مجاز = ۶-۲)

result=4

حالت ب.۱)

(طول رشته ی مجاز = ۷-۱)

result=6

حالت الف)

stack: [0,1,8,9]

result=6

حالت ب (۲.۰) index=10

(طول رشته‌ی مجاز = ۱۰-۲)(۳=۰,۱,۸) stack: [0,1,8]

result=6

۴.۳ یک رشته (string) از پرانتز باز و پرانتز بسته داده شده است. الگوریتمی بهینه ارایه دهید که حداقل تعداد چرخش‌های لازم در پرانتز‌ها را پیدا کند تا چینش پرانتز‌ها مجاز شود. (چرخش یعنی تبدیل پرانتز باز به بسته یا بر عکس)

راه حل:

از استک استفاده می‌کنیم.

۱. اگر کاراکتر پرانتز باز بود آن را در استک push می‌کنیم

۲. اگر کاراکتر پرانتز بسته بود

۲.الف اگر top استک پرانتز باز بود استک را pop می‌کنیم

۳. ب اگر top استک پرانتز بسته بود یا استک خالی بود پرانتز بسته را در استک push می‌کنیم.

با این کار تمام زیر رشته‌های مجاز از رشته حذف می‌شوند و رشته‌ی باقی‌مانده در استک به این شکل در می‌آید: ...)))))))

فرض کنید تعداد پرانتز بسته ها  $n$  و تعداد پرانتز بازها  $m$  است.

تعداد چرخش‌های لازم برابر است با  $\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil$

## نمونه سؤالات مربوط به بخش ۲.۴

۱.۲.۴ الگوریتمی بنویسید که با هزینه‌ی زمانی  $O(n^5)$  اعداد باینری ۱ تا  $n$  را چاپ کند. (فرض کنید تابع چاپ کردن فقط string به عنوان ورودی می‌گیرد یعنی اجازه‌ی چاپ کردن int, ... را ندارید)

راه حل:

اول "۱" را در یک صف enqueue می‌کنیم.

بعد به تعداد  $n$  بار این کار را تکرار می‌کنیم

۱. صف را dequeue می‌کنیم و عنصر را چاپ می‌کنیم

۲. به ته عنصر dequeue شده اضافه می‌کنیم و حاصل را در صف enqueue می‌کنیم

۳. به ته عنصر dequeue ۱ اضافه می‌کنیم و حاصل را در صف enqueue می‌کنیم

۲.۲.۴ فرض کنید  $n$  ایستگاه اتوبوس داریم. اتوبوس از هر ایستگاه فقط به ایستگاه بعدی می‌رود. (و از ایستگاه  $n$  به ایستگاه ۱ می‌رود). برای هر ایستگاه مقدار سوختی که در آن است و فاصله‌ی ایستگاه تا ایستگاه بعدی را داریم. (فرض کنید با  $k$  لیتر سوخت  $k$  متر فاصله میتوان طی کرد). فرض کنید اتوبوس اول خالی است.

الف) الگوریتمی بهینه ارایه دهید که تعیین کند اتوبوس از کدام ایستگاه شروع به حرکت کند تا در راه نماند.

ب) با توجه به قسمت الف بگویید که اگر مسیری وجود نداشته باشد الگوریتم چطور متوجه می‌شود؟

راه حل:

الف) برای تمام ایستگاه‌ها مقدار سوخت موجود منهای مقدار سوخت مورد نیاز تا ایستگاه بعد را محاسبه می‌کنیم. این مقدار (که ممکن است منفی باشد) موقع رسیدن به آن ایستگاه به کل سوخت اضافه می‌شود. از یک ایستگاه دلخواه شروع می‌کنیم ایستگاه‌ها را یکی یکی در یک صف میریزیم. و هر بار میزان سوخت باقی‌مانده را محاسبه می‌کنیم. اگر میزان سوخت منفی شد ایستگاه‌ها را dequeue می‌کنیم تا میزان سوخت ثابت شود. این کار را ادامه میدهیم تا تمام ایستگاه‌ها در صف enqueue شده باشند.

ب) اگر اولین ایستگاه دوبار dequeue شد یعنی تمام حالت‌ها بررسی شده و همچین مسیری وجود نداشته است.

۳.۲.۴ فرض کنید یک ماتریس  $m \times n$  داریم. هر خانه‌ی  $i, j$  است یا یک گلابی سالم در آن است یا یک گلابی گندیده. هر گلابی گندیده در هر مرحله میتواند گلابی‌های سالم مجاورش را بگنداند. الگوریتمی ارایه دهید که حداقل مراحل مورد نیاز برای گندیدن تمام گلابی‌ها را تعیین کند.

### راه حل:

تمام گلابی های گندیده را در یک صف میریزیم (منظور از ریختن گلابی ها ریختن اندیس خانه است که در آن اند) و برای هر کدام علامت میگذاریم که مشخص شود در صف ریخته شده اند. سپس یک عنصر جداکننده در صف enqueue میکنیم. (مثلاً کاراکتر \$).

سپس صف را dequeue میکنیم و به ازای هر عنصر شده تمام همسایه های سالم آن (که تا کنون در صف ریخته نشده اند) را میگذاریم (!) و در صف میریزیم. هروقت به عنصر جداکننده رسیدیم تعداد مراحل مورد نیاز را یکی زیاد میکنیم و اگر صف خالی نبود عنصر جدا کننده ی دیگری در صف enqueue میکنیم.

اگر صف خالی شد. بررسی میکنیم که آیا گلابی ای سالم مانده یا نه. اگر گلابی ای سالم مانده بود ۱- برمیگردانیم و اگر نه تعداد مراحل مورد نیاز محاسبه شده را برمیگردانیم.  
(اگر این سؤال برایتان سخت است اول بخش BFS را بخوانید)

۴. فرض کنید یک لیست پیوندی یک طرفه ی مرتب (صعودی) از اعداد صحیح در اختیار دارید.

الف) الگوریتمی ارایه دهید که بدون شمردن تعداد عناصر میانه را برگرداند.  
ب) الگوریتمی ارایه دهید که بدون شمردن تعداد عناصر  $n$  مینیم بزرگترین عنصر را برگرداند.

الف) دو اشاره گر از اول لیست در نظر میگیریم. هر بار اشاره گر اول را یکی و اشاره گر دوم را دوتا جلو میگیریم. وقتی اشاره گر دوم به ته لیست رسید اشاره گر اول به وسط لیست رسیده و به میانه اشاره میکند.

ب) دو اشاره گر از اول لیست در نظر میگیریم. اشاره گر اول را  $a$  تا جلو میگیریم سپس هر دو اشاره گر را یکی جلو میگیریم تا اشاره گر اول به آخر لیست برسد. اشاره گر دوم به  $a$  مینیم بزرگترین عنصر اشاره میکند.

### نمونه سؤالات مربوط به بخش ۳.۴

۱.۳.۴ فرض کنید یک لیست پیوندی دوطرفه از اعداد صحیح داریم. (برای راحتی کار فرض کنید عدد تکراری در لیست وجود ندارد)  
الف) الگوریتمی ارایه دهید که با هزینه  $O(\log n)$  دو عدد از لیست را پیدا کند که مجموع آن دو عدد دلخواه  $k$  است.  
ب) آیا با لیست یک طرفه هم میتوان در  $O(\log n)$  دو عدد را پیدا کرد؟  
ج) فرض کنید سه لیست از اعداد صحیح داریم. الگوریتمی ارایه دهید که در  $O(n^*n)$  سه عنصر  $a$  و  $b$  و  $c$  از لیست ۱ و ۲ و ۳ پیدا کند که مجموع آنها عدد دلخواه  $k$  باشد.

### راه حل:

الف) لیست را با هزینه  $O(\log n)$  مرتب میکنیم. سپس دو اشاره گر  $a$  و  $b$  به عنصر اول و آخر آن در نظر میگیریم. مجموع این دو عدد اگر از  $k$  کمتر بود اشاره گر  $a$  را یکی زیاد میکنیم و اگر از  $k$  بیشتر بود اشاره گر  $b$  را یکی کم میکنیم. این کار را ادامه میدهیم تا مجموع عناصری که  $a$  و  $b$  به آن اشاره میکنند  $k$  شود یا از  $b$  بزرگ‌تر شود که در این صورت یعنی هیچ دو عددی مجموعشان  $k$  نمیشود.

ب) از این ایده استفاده میکنیم که اگر مجموع دو عدد  $k$  شود حتماً یکی کمتر از  $k/2$  بوده و یکی بیشتر از  $k/2$ . عدد  $k/2$  را در لیست اضافه میکنیم سپس با هزینه  $O(\log n)$  لیست را مرتب میکنیم. لیست را از مکانی که عنصر  $k/2$  در آن قرار دارد به دو لیست کوچکتر (لیست  $i$  عناصر کوچکتر از  $k/2$  و لیست  $j$  عناصر بزرگ‌تر از  $k/2$ ) تقسیم میکنیم (این کار با جایه جا کردن اشاره گرها با  $O(1)$  قابل انجام است) (توجه کنید که خود عنصر  $k/2$  را در این لیست ها نمیگذاریم) حالا لیست  $i$  را نزولی مرتب میکنیم. اشاره گر  $a$  و  $b$  را به اول دو لیست  $i$  و  $j$  تعریف میکنیم. بقیه ای راه حل شیوه قسمت الف است.

ج) حل این قسمت شبیه به قسمت های قبل بوده و بر عهده ی خود شما گذاشته شده است.

۲.۳.۴ دو لیست پیوندی مرتب  $A$  و  $B$  از اعداد صحیح داده شده اند این دو لیست شامل عناصرهای مشترک هستند. میخواهیم لیست پیوندی  $C$  (متشکل از عناصر های  $A$  و  $B$ ) را بسازیم. طوری که مجموع عناصر  $C$  ماکزیمم شود. با این شرط که به ازای هر عنصر  $i$  که در  $C$  است یا  $i$  بین  $A$  و  $B$  مشترک است یا عنصر قبلی آن در  $C$  با عنصر قبلی آن در لیست  $A$  یا  $B$  یکی است. به مثال توجه کنید: (در این مثال عناصر های ۳ و ۹۰ و ۲۴۰ بین لیست  $A$  و  $B$  مشترک هستند)

List A = 1->3->30->90->120->240->511

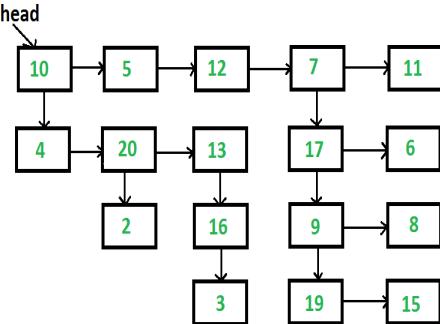
List B = 0->3->12->32->90->125->240->249

List C = 1->3->12->32->90->125->240->511

### راه حل:

دو pointer به اسم a و b به عناصر اول (head) لیست A و B تعریف میکنیم. و دو متغیر suma و sumb در هر مرحله هر کدام از a و b را که محتویات آن کوچکتر بود یکی اضافه میکنیم و محتویات آن را به suma یا sumb اضافه میکنیم. این کار را تکرار میکنیم تا دو محتویات مساوی داشته باشند. بعد بر اساس اینکه suma بیشتر است یا sumb. انتخاب میکنیم که عنصر بعدی در آرایه i عنصر اول A باشد یا B. و تمام عناصر را به ترتیب در C میریزیم تا به عنصر مشترک برسیم. بعد head لیست A و B را برابر عنصر بعدی a و b قرار میدهیم و الگوریتم را تکرار میکنیم تا هر دو اشاره گر a و b به آخر لیست ها برسند.

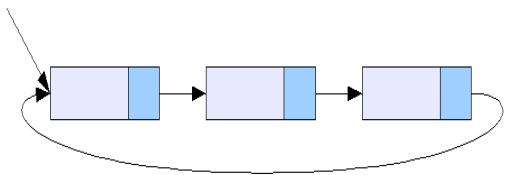
**۳.۳.۴ فرض کنید یک لیست از عناصر دارد که هر عنصر علاوه بر اشاره گر به عنصر بعدی (next) یک اشاره گر به فرزند (child) هم دارد. این لیست را به یک لیست پیوندی یک طرفه تبدیل کنید. طوری که هر عنصر قبل از عنصر بعدی (next) و فرزند (child) خود آمده باشد.**



شکل ۱  
راه حل:

دو اشاره گر در نظر میگیریم. اشاره گر اول a به اول لیست و اشاره گر دوم b به انتهای ردیف اول لیست اشاره میکند. (برای اشاره گر دوم آن را مساوی head قرار میدهیم و آنقدر next میکنیم تا به ته ردیف اول (با توجه به شکل ۱) برسد. حالا عناصر اشاره گر اول را در لیست پیوندی اضافه میکنیم. و به عنصر بعدی میرویم. به ازای هر عنصر که اشاره گر اول به آن اشاره میکند اگر  $child \neq NIL$  بود فرزند را به انتهای لیست اضافه میکنیم. (b=b.next و b.next=a.child). این کار را ادامه میدهیم تا a با b برابر شود.

**۴.۳.۴ تابع زیر روی لیست پیوندی چرخشی (به شکل ۲ نگاه کنید) اجرا میشود.**



شکل ۲ لیست پیوندی چرخشی

```
int Func(List * l){
    if(l==NIL)
        return 0;
    if(l->next == l)
        return l->data;
    l->next = l->next->next;
    return Func(l->next);
}
```

با شروع از عنصر یک تعیین کنید:

- الف) اگر لیست ۱ چهار عنصر [1,2,3,4] داشته باشد خروجی چیست؟
- ب) اگر لیست ۱ ده عنصر داشته [1,2,3,4,5,6,7,8,9,10] داشته باشد خروجی چیست؟
- ج) اگر لیست ۱ از n عنصر [1,2,3,4,...,n] تشکیل شده باشد خروجی چیست؟

### راه حل:

الف) عنصر ۱ خروجی است

ب) عنصر ۰ خروجی است  
 ج) با کمی دقت متوجه می‌شویم تابع هر بار عناصر را یکی درمیان حذف میکند و در آخر عنصری که میماند را بر میگرداند.  
 اگر تعداد عناصر توان دو باشد عنصر اول برگردانده میشود. و گرنه آن عنصری برگردانده میشود که موقعی که به آن رسیدیم تعداد عناصر توان ۲ باشد.  
 فرض کنید  $k$  بزرگترین عددی است که  $n = 2^k + p$  ( واضح است که  $p < n/2$  است)  
 پس عنصری برگردانده میشود که بعد از  $p$  بار حذف کردن به آن برسیم. با کمی دقت متوجه می‌شویم شماره ۰ عنصری که بعد از  $p$  بار حذف کردن به آن میرسیم برابر است با:  

$$p \times 2^{k+1} - p \leq n \Rightarrow p \leq \frac{n}{2}$$
 و چون  $p$  کمتر از  $n/2$  است.  $p \times 2^{k+1} - p < n$  پس عنصری با این شماره حتماً وجود دارد.

۰.۳.۴ فرض کنید یک جور لیست پیوندی داریم که هر عنصر دو اشاره گر `next` و `random` دارد.  
 اشاره گر `next` به عنصر بعدی اشاره میکند و اشاره گر `random` به یک عنصر دلخواه از لیست اشاره میکند.  
 الف) لیست پیوندی از این نوع را در زمان  $O(n)$  کپی کنید.  
 ب) لیست پیوندی از این نوع را در زمان  $O(n)$  کپی کنید. طوری که هزینه ۰ حافظه (۱) باشد.

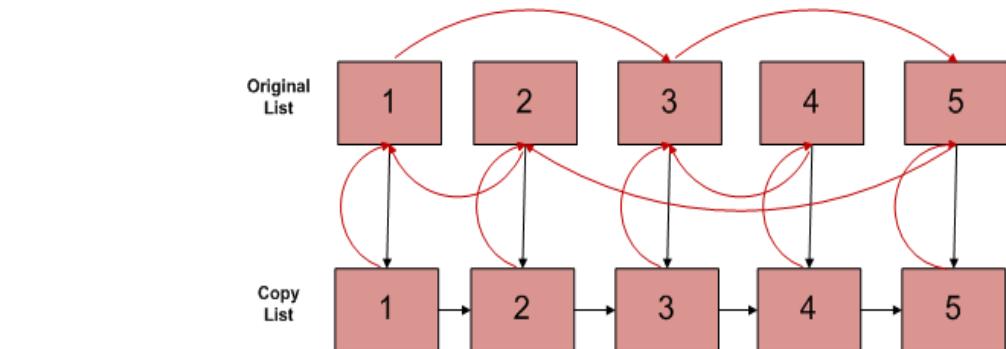
#### راه حل:

الف) هزینه ۰ حافظه :  $O(n)$

۱. برای هر عنصر از لیست پیوندی اصلی اشاره گر `next` را در حافظه ۰ ذخیره میکنیم.
۲. عناصر لیست پیوندی اصلی را کپی میکنیم. و برای هر عنصر کپی شده اشاره گر `next` را به سمت عنصر بعدی قرار میدهیم.(مثل کپی کردن لیست پیوندی معمولی).
۳. برای هر عنصر کپی اشاره گر `random` را به سمت عنصر معادل آن در لیست پیوندی اصلی قرار میدهیم. و در لیست پیوندی اصلی اشاره گر `next` هر عنصر را به سمت معادل آن در لیست پیوندی کپی قرار میدهیم.(اینجا لیست پیوندی اصلی دستکاری شد) (به شکل ۳ توجه کنید)
۴. حالا برای سرتاسر اشاره گر های `random` عناصر کپی این گونه عمل میکنیم.

`Copy_element → random = Copy_element → random → next`

۰. عنصر لیست اصلی را از روی اطلاعات ذخیره شده در مرحله ۱ دوباره بازسازی میکنیم.



شکل ۳ فلش های سیاه اشاره گر های `next` و فلش های قرمز اشاره گر های `random` هستند

ب) هزینه ۰ حافظه:  $O(1)$

۱. برای هر عنصر اصلی یک کپی ایجاد میکنیم و آن را در آرایه ۰ اصلی بعد از همتای اصلیش قرار میدهیم.  
 (اشاره گر `random` عناصر کپی `NIL` است)
۲. اشاره گر `random` عناصر کپی را سرتاسر میکنیم:

for each original element do

`original → next → random = original → random → next`

۳. عناصر کپی و اصلی را از هم جدا میکنیم:

for each original element do {

`copy = original → next`

`original → next = original → next → next`

`copy → next = copy → next → next`

}

---

۴.۳.۶ فرض کنید یک لیست پیوندی یک طرفه مرتب (صعودی) از اعداد صحیح داده شده است

الف) بدون شمردن تعداد عناصر میانه را پیدا کنید

ب) بدون شمردن تعداد عناصر  $k$  امین بزرگ‌ترین عدد را پیدا کنید

راه حل:

الف) دو اشاره گر به اول لیست در نظر می‌گیریم هریار اولی را یکی و دومی را دو تا جلو می‌بریم. وقتی

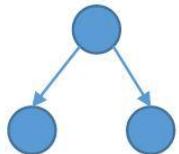
دومی به آخر رسید اولی به عنصر وسط یعنی میانه اشاره می‌کند

ب) دو اشاره گر به اول لیست در نظر می‌گیریم. اولی را  $k$  تا جلو می‌بریم و از این به بعد هر دو را یکی جلو می‌بریم. وقتی اولی به آخر رسید دومی به  $k$  امین بزرگ‌ترین عنصر اشاره می‌کند

## نمونه سوالات مربوط به بخش ۴

۱. یک درخت  $T$  با  $n$  عنصر در نظر بگیرید که هر گره غیر برگ آن ۲ فرزند دارد. فرض کنید  $E(T)$  و  $I(T)$

به ترتیب مجموع عمق برگها و مجموع عمق داخلی (عناصر غیر برگ) در  $T$  باشد. چه رابطه‌ای بین اختلاف  $E(T) - I(T)$  بر حسب  $n$  همواره برقرار است؟ حدس بزنید و با استقرار اثبات کنید.



راه حل:

اثبات میکنیم  $E(T) - I(T) = n - 1$  پایه استقرا برای  $n=1$  یا  $n=3$  برقرار است. مثلاً برای  $n=3$  در درخت روبه رو و  $E(T)=2$  و  $I(T)=0$  پس برقرار است.

---

۲. برای تبدیل یک درخت دو دویی به یک درخت نخ کشی با کمترین زمان ممکن چه الگوریتمی پیشنهاد میدهید.

راه حل:

ابتدا پیمایش preorder از درخت را میسازیم سپس هر گره ای که برگ بود و یا این که یکی از بچه هایش وجود نداشت با توجه به اینکه بچه چپ و یا راستش است به ترتیب به گره قبلی و بعدی در پیمایش میانوند متصل می کنیم

---

۳. اگر بخواهیم به همین درخت نخ کشی بروگی اضافه کنیم با کمترین زمان ممکن چه الگوریتمی پیشنهاد میدهید.

راه حل:

فرض می کنیم این برگ قرار است فرزند سمت راست پدرش شود پس نخ سمت چپ آن به پدرش خواهد بود و نخ سمت راست آن به گره ای خواهد بود که قبلاً نخ سمت راست پدرش به آن بوده است.

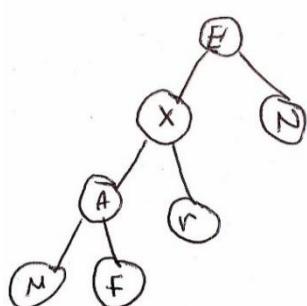
---

۴. درخت زیر را رسم کنید

1. Preorder traversal: EXAMFUN

2. Inorder traversal: MAFXUEN

راه حل:



۵.۴ درختی با  $n$  گره  $v, w$  پایین ترین گرهی است که هر دو این گره ها جز فرزندان یا نوادگان آن باشند (هر گره جز نوادگان خودش است) الگوریتمی بهینه برای پیدا کردن جد مشترک دو گره  $v, w$  ارایه دهید و مرتبه زمانی آن را مشخص کنید

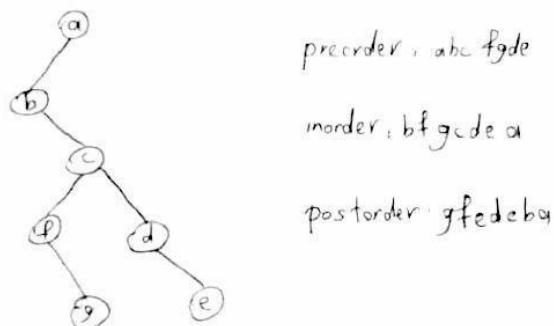
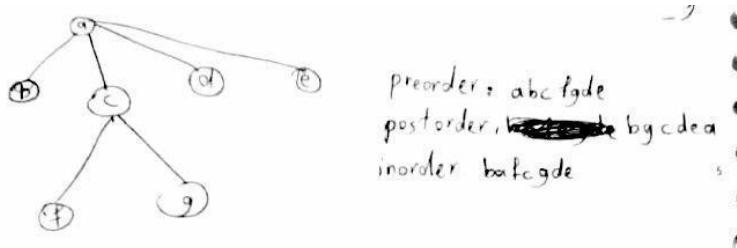
راه حل:

میدانیم که پدر گره  $i$  گره  $\lceil i/2 \rceil$  است . تا هنگامی که این مقدار بزرگتر از یک است این عمل را تکرار می کنیم و در یک آرایه دیگر ، مقدار خانه های ارایه را که با تقسیم  $v$  بر ۲ بدست می اید true می کنیم برای مثال مقادیر  $\lceil v/2 \rceil / 2$  را که عملاً اجداد  $v$  هستند ، در آرایه جدید true شده اند . از آن جایی که درخت  $n$  گره دارد تعداد اجدادش از  $O(\lg n)$  است حال سراغ  $w$  می رویم و اجداد آن را بررسی می کنیم اولین جایی که اندیس آن از اجداد  $w$  در آرایه true بود آن راس کوچکترین جد مشترک است که تعداد اجداد  $w$  نیز  $O(\lg n)$  است پس در مجموع  $O(\lg n)$  است.

#### ۶.۴ درخت دودویی است که درخت $T$ را نمایش میدهد:

- الف) آیا پیمایش preorder درخت  $T$  با پیمایش preorder درخت  $T$  یکسان است؟
- ب) آیا پیمایش post order درخت  $T$  با پیمایش postorder درخت  $T$  یکسان است؟
- ج) آیا پیمایش inorder درخت  $T$  با پیمایش inorder درخت  $T$  یکسان است؟

راه حل:



#### 7.4 الگوریتمی غیر بازگشته برای پیمایش inorder یک درخت دودویی در زمان خطی ارایه دهید

راه حل:

از یک استک و ارایه کمکی (mark) کمک می‌گیریم ابتدا ریشه را در درون استک push می‌کنیم آرایه mark را برای این گرفتیم که چک کنیم آیا گره  $i$  را قبلاً پیمایش کردیم یا خیر. در صورت پیمایش (یا true بودن) دیگر به پیمایش ان نمی‌پردازیم. در هر مرحله عنصر بالای استک را در نظر می‌گیریم مثلاً  $V$  سپس اگر بچه سمت چپ این گره false بود آن را در استک push می‌کنیم در غیر این صورت خود  $V$  را بررسی می‌کنیم [v] را mark می‌کنیم اگر هم از قبل true بود سراغ بچه سمت راست گره  $V$  می‌رویم اگر v.right نیز TRUE بود  $V$  را POP می‌کنیم در غیر این صورت push را v.right در استک می‌کنیم. در این روش inorder درخت را غیر بازگشته طی کردیم

---

#### 8.4 در درختی با $n$ گره، گره آخر در پیمایش Post order با آخرین گره از پیمایش preorder برابر است. ارتقای درخت را بدست آورید.

راه حل:

ریشه در preorder با اولین گره پیمایش شده و در postorder آخرین گره. از آنجایی مه در این دو پیمایش اخرین گره postorder (ریشه) با اولین گره preorder یکی است در نتیجه درخت تنها یک گره (ریشه) دارد.

۹.۴ برنامه زیر چه عملی روی درخت انجام میدهد.

```
bool A(Node Start, Node V) {
    if (v==null)
        return false;
    if (Start == V) {
        print (Start->Value);
        return true;
    }
    else if (A(Start->RChild,V) || A(Start->LChild,V)) {
        print Start->Value;
        return true;
    }
    return false;
}
```

راه حل:

این برنامه اگر گره  $V$  در درخت باشد تمام اجداد  $V$  را از جمله خود  $V$  تا ریشه چاپ می کند

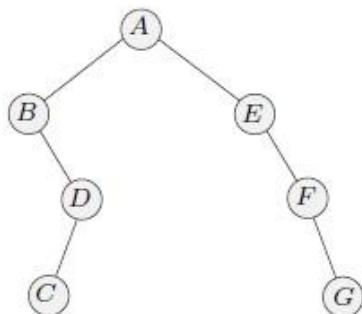
---

۱۰.۴ خروجی این قطعه کد بر درخت وقتی اجرا می شود چیست؟

فرض کنید  $(s)$  رشته / عنصر پارامترش  $(s)$  را در خروجی می نویسد

PRINTTREE ( $T$ )

- 1   if  $T = \text{null}$  then return
- 2   if  $\text{left}[T] \neq \text{null}$  then WRITE('(')
- 3   PRINTTREE( $\text{left}[T]$ )
- 4   if  $\text{left}[T] \neq \text{null}$  then WRITE(')')
- 5   WRITE( $\text{element}[T]$ )
- 6   if  $\text{right}[T] \neq \text{null}$  then WRITE('(')
- 7   PRINTTREE( $\text{right}[T]$ )
- 8   if  $\text{right}[T] \neq \text{null}$  then WRITE(')')

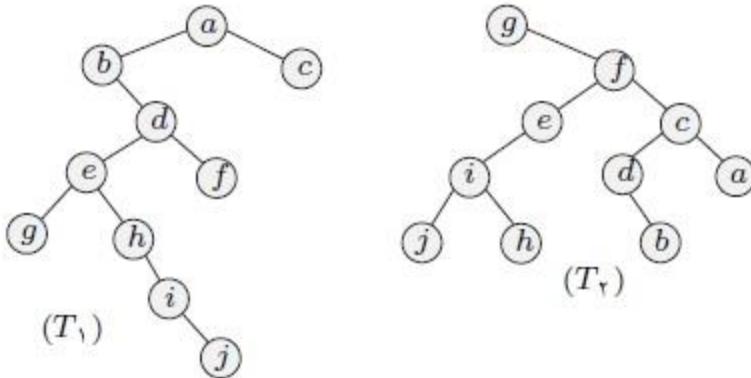


راه حل:

این شبه کد با پرانتز گذاری نمایش  $(B((C) D))A(E(F(G)))$  را چاپ می کند.

۱۱.۴ این دو درخت کدام یک از روش های پیمایش به ترتیب بر روی  $T_1$ ,  $T_2$  دنباله های یکسانی تولید می کند؟

راه حل:



$T_1$ :

Preorder: abdeghijfc

Postorder: gjihefdbc

Inorder: bgehijdfac

$T_2$ :

Preorder: gfeijhcdba

Postorder: jhiebdacfg

Inorder: gjihefdbc

۱۲.۴ با  $n$  عنصر چند درخت دودویی متوازن با ارتفاع  $\lg n$  می توان ساخت؟

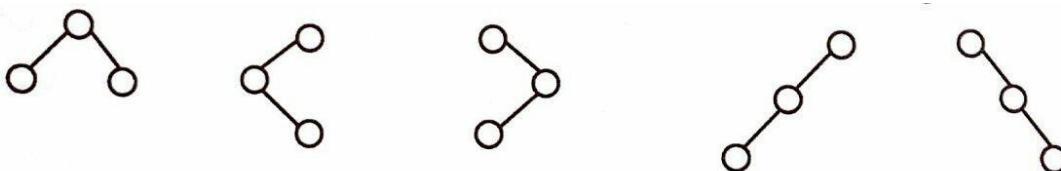
راه حل:

کمینه ارتفاع درخت دودویی با  $n$  عنصر برابر  $\lg n$  است این درخت را  $T^h$  نامیم . با فرض متوازن بودن  $T^h$  ناگزیر است بین ۱ تا  $2^h$  برگ در ارتفاع  $h$  داشته باشد و با حذف این برگها درختی باقی می ماند به نام  $T^{h-1}$  که یک درخت پر با ارتفاع  $h-1$  است  $T^{h-1}$  تک است و  $2^{h-1}$  گره دارد . همان  $T^h$  است که  $(2^h - 1) = 2^h r - 1$  جدید به برگهای آن اضافه شده است پس تعداد  $T^h$  برابر  $\binom{2^h}{r}$  است.

۱۳.۴ تعداد درخت های دودویی غیر هم رخت (Non-isomorphic binary tree) دارای سه راس چه مقداری است

راه حل:

۵ درخت ممکن است در حالت کلی با  $n=3$  گره  $\frac{1}{n+1} \binom{2n}{n}$  درخت متفاوت میتوان رسم کرد



۱۴.۴ درخت  $T$  دارای ۱۰ راس است. تعداد مسیر های موجود در  $T$  با طول حداقل یک چیست؟

راه حل:

در درخت بین هر دو گره دلخواه فقط یک مسیر وجود دارد پس این مساله معادل انتخاب دو نمونه از ۱۰ نمونه است  $\binom{10}{2} = 45$

۱۵.۴ با چند حالت می توان اعداد ۱ تا ۱۱ را در گره های درخت زیر برچسب گذاری کرد تا عدد هر گره از اعداد فرزندانش بزرگتر

باشد ؟ دقیق کنید عدد باید استفاده شود و تکرار مجاز نیست ؟

راه حل:

تعداد حالات را با  $T(11)$  نمایش می دهیم . میدانیم که ۱۱ باید ریشه باشد از بین  $10$  عدد  $= 120 = \binom{10}{3}$  حابت می توان  $3$  عنصر زیر

درخت راست انتخاب کرد بدیهی است که برای هر انتخاب ، عناصر زیر درخت چپ هم مشخص می شود. پس تعداد کل حالات برابر است با

$$T(5) = 4T(3) = 8 \quad T(7) = \binom{6}{1}T(1)T(5) = 2 \quad T(3) = 4T(1) = 120 \quad T(11) = 120 * T(7) * T(3)$$

$$\text{درنتیجه } T(7) = 48 \quad \text{پس پاسخ برابر است با } 11520 = 120 * (6 * 8) * 2$$

## نمونه سوالات مربوط به بخش ۴.۲

۱۰.۲.۴ ثابت کنید هزینه زمانی پیمایش میان ترتیب (inorder) یک درخت دودویی  $O(n)$  میباشد.

راه حل:

زمان لازم برای پیمایش inorder درخت دودویی :  $T(n)$

در پیمایش به روش بازگشتی ، ابتدا زیر درخت چپ و بعد از ریشه ، زیردرخت راست را پیمایش میکنیم که هر دو یک درخت دودویی هستند.  
پس میتوان نوشت:

$$T(n) = T(k) + T(n-k-1) + c$$

با استقرا قوی اگر داشته باشیم  $T(x)=ax+b$  ( $x < n$ ) خواهیم داشت :

$$T(n) = ak+b + a(n-k-1)+b + c = ak+b+an-ak-a+b+c = a'n+b'$$

$$\Rightarrow T(n) = O(n)$$

۲۰.۲.۴ پیمایش inorder یک درخت دودویی را بدون استفاده از recursion و stack پیاده سازی کنید.

راه حل:

این کار به وسیله الگوریتم موریس امکان پذیر است :

ریشه را به عنوان گره فعلی (current) قرار میدهیم ، تا زمانی که current مخالف null باشد عملیات زیر انجام میدهیم :  
اگر current فرزند چپ نداشت ، مقدار آن را چاپ میکنیم و به زیر درخت راست میرویم ( $current = current->right$ )  
اگر current فرزند چپ داشت ، در زیر درخت سمت چپ ، فرزند راست راست ترین گره را current قرار میدهیم و به زیر درخت چپ  
میرویم.

## نمونه سوالات مربوط به بخش ۴.۶

۱۰.۶.۴ روشی ارائه کنید که بتوان به کمک آن مشخص کرد که آیا یک درخت دودویی ، درخت جستجوی دودویی هست یا خیر .

راه حل:

روش ۱ (غیر بهینه) :

برای هر نод ، بررسی میکنیم که همه نود های زیردرخت سمت چپ کوچکتر از آن و همه نود های زیردرخت سمت راست آن بزرگتر از آن باشند.

روش بهینه :

حداکثر مقدار عناصر زیر درخت سمت چپ هر نود ، مقدار آن نود است و حداقل مقدار عناصر زیردرخت سمت راست برابر نیز مقدار نود است یعنی برای هر نود بازه ای وجود دارد که اگر مقدار نود خارج آن باشد، درخت نمیتواند درخت دودویی جستجو باشد. با استفاده از همین موضوع روشنی بازگشتی ارائه میدهیم به این ترتیب که از ریشه شروع میکنیم و بازه را از منفی بینهایت تا مشت بینهایت در نظر میگیریم. در پیمایش هر نود، برای زیر درخت سمت چپ انتهای بازه و برای زیردرخت سمت راست ابتدای بازه را برابر مقدار نود قرار میدهیم و در هر مرحله اگر مقدار گره در حال پیمایش خارج از بازه بود، درخت مورد نظر درخت دودویی جستجو نیست.

۲.۶.۴ روشنی ارائه دهید که بتوان نمایش پس ترتیب یک درخت جستجوی دودویی را از روی نمایش پیش ترتیب آن به دست آورده.

راه حل :

اولین عنصر ریشه است ، با توجه به آن بقیه عناصر را در دو گروه قرار میدهیم و برای هر گروه به صورت بازگشتی پیمایش پس ترتیب را بدست می آوریم. بیشترین تعداد باری که حلقه تکرار میشود برابر تعداد عناصر است پس الگوریتم  $O(n)$  میباشد.

```
void Pre2Post(vector<int> &p){  
    if ( p.size() < 2)  
        return;  
    int root = p[0];  
    vector<int> group1;  
    vector<int> group2;  
    for(int i = 0 ; i < p.size() ; i++)  
        if(p[i] < root)  
            group1.push_back(p[i]);  
        else  
            group2.push_back(p[i]);  
    Pre2Post(group1);  
    Pre2Post(group2);  
    p.clear();  
    for(int i = 0 ; i < group1.size() ; i++)  
        p.push_back(group1[i]);  
    for(int i = 0 ; i < group2.size() ; i++)  
        p.push_back(group2[i]);  
    p.push_back(x);  
}
```

۳.۶.۴ با اعداد ۱ تا  $n$  چند درخت جستجوی دودویی میتوان ساخت؟

راه حل :

برای ریشه  $n$  انتخاب وجود دارد. به ازای هریک از این انتخاب‌ها زیر درخت‌های چپ و راست خود یک درخت جستجوی دودویی هستند. پس داریم :

$$F(0) = 1, F(n+1) = \sum F(i)F(n-i)$$
$$F(n) = \binom{2n}{n} - \binom{2n}{n-1} = \frac{(2n)!}{n!n!} - \frac{(2n)!}{(n-1)!(n+1)!} = \frac{\binom{2n}{n}}{n+1}$$

که این رابطه مربوط به اعداد کاتالان هست که در درس ساختمان‌های گسسته با آن آشنا شده‌ایم.

۴.۶.۴ فرض کنید مسیر رسیدن به یک نود برگ در درخت جستجوی دودویی را داریم. مجموعه رئوس سمت چپ مسیر را  $A$ ، رئوس داخل مسیر را  $B$  و رئوس راست مسیر را  $C$  مینامیم. آیا به ازای هر  $a$  عضو  $A$ ،  $b$  عضو  $B$  و  $c$  عضو  $C$  گزاره زیر صحیح است؟

$a \leq b \leq c$

راه حل :

خیر صحیح نیست. به عنوان مثال نقض داریم :

۴

۲

۶

۱

۳

۵

۷

در این درخت جستجوی دودویی، مسیر رسیدن به برگ ۷ یا ۵ از ریشه مثال نقض میباشد.

۵.۶.۴ شبکه کد درخت جستجوی دودویی را به گونه‌ای تغییر دهید که بتوان با هزینه  $(h)$  تعداد عناصر کوچکتر از یک گره را پیدا کرد.

راه حل :

تعداد عناصر کوچکتر را برای هر عنصر ذخیره میکنیم و برای بروز رسانی آن چنین عمل میکنیم : هنگام عمل درج (insert) اگر از رأسی به سمت چپش رفتیم، مقدار ذخیره شده را یکی افزایش میدهیم. هنگام حذف (delete) نیز در مسیر اگر به چپ رفتیم، مقدار مورد نظر را برای رأس یکی کاهش میدهیم.

به این ترتیب برای یافتن تعداد عناصر کوچکتر از یک گره ، اگر در مسیر رسیدن به آن گره به سمت راست رفتیم مقادیر ذخیره شده را جمع میکنیم.

۶.۶.۴ اگر نود  $X$  در **BST** دو فرزند داشته باشد ، در این صورت **successor** نود  $X$  فرزند چپ نخاهد داشت. با ذکر دلیل درستی این گزاره را نشان دهید.

راه حل:

با توجه به این که  $X$  دو فرزند دارد ، پس **successor** کوچکتر ترین عنصر زیردرخت سمت راست  $X$  میباشد که برای به دست آوردن آن از  $X$  به زیردرخت راست میرویم و بعد از آن به چپ میرویم تا جایی که به **null** برسیم پس تا جایی که فرزند چپ وجود داشته باشد باید به چپ برویم تا به **successor** نمیتواند فرزند چپ داشته باشد. که اگر داشت دیگر **successor** نبود.

۷.۶.۴ الگوریتم یا تابعی ارائه دهید که به کمک آن بتوان عناصر مشترک دو **BST** را چاپ کرد.

راه حل:

با کمک پیمایش میان ترتیب به وسیله پشته(**stack**) به هزینه زمانی  $O(n+m)$  امکان پذیر است . به این صورت که :

```
void printCommon(Node *root1, Node *root2)
{
    // Create two stacks for two inorder traversals
    stack<Node *> stack1, s1, s2;

    while (1)
    {
        // push the Nodes of first tree in stack s1
        if (root1)
        {
            s1.push(root1);
            root1 = root1->left;
        }

        // push the Nodes of second tree in stack s2
        else if (root2)
        {
            s2.push(root2);
            root2 = root2->left;
        }

        // Both root1 and root2 are NULL here
        else if (!s1.empty() && !s2.empty())
        {
            root1 = s1.top();
            root2 = s2.top();

            // If current keys in two trees are same
            cout << root1->key << " ";
        }
    }
}
```

```

if (root1->key == root2->key)
{
    cout << root1->key << " ";
    s1.pop();
    s2.pop();

    // move to the inorder successor
    root1 = root1->right;
    root2 = root2->right;
}

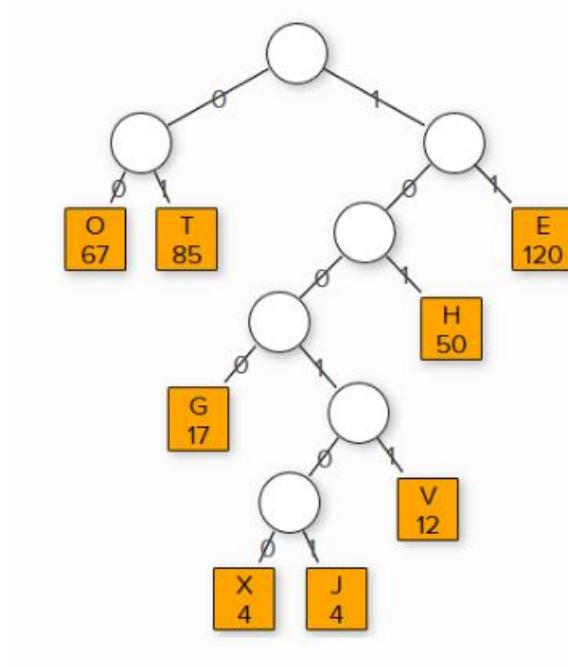
else if (root1->key < root2->key)
{
    // If Node of first tree is smaller, than that of
    // second tree, then its obvious that the inorder
    // successors of current Node can have same value
    // as that of the second tree Node. Thus, we pop
    // from s2
    s1.pop();
    root1 = root1->right;

    // root2 is set to NULL, because we need
    // new Nodes of tree 1
    root2 = NULL;
}
else if (root1->key > root2->key)
{
    s2.pop();
    root2 = root2->right;
    root1 = NULL;
}
}

// Both roots and both stacks are empty
else break;
}
}

```

برای درخت هافمن نشان داده شده در شکل زیر ، عدد 01 چه حرفی را نشان میدهد؟



راه حل:

با پیمایش 01 در درخت ، به حرف T میرسیم

برای درخت سوال قبل ، برای نمایش G از چه رشته ای باید استفاده کنیم؟

راه حل:

برای رسیدن به حرف G باید رشته 1000 را روی درخت پیمایش کنیم

در درخت هافمن سوال ۱ ، برای نمایش رشته THE باید چه رشته ای را پیمایش کنیم؟

راه حل

1011101

در شکل سوال ۱ ، رشته 10000001 نمایشگر چه رشته ای است؟

راه حل

GOT

1.5.4 پیچگی زمانی ساخت درخت هیپ را بدست آورید ، منظور از ساخت درخت هیپ ، ساختن یک درخت هیپ با آرایه داده شده است

( اولین مرحله برای مرتب سازی )

راه حل :

با توجه به الگوریتم ساختن هیپ ، مرتبه‌ی زمانی  $O(n)$  خواهد بود.

2.5.4 فرض کنید برای مرتب سازی یک آرایه متشکل از هشت عدد ، از روش مرتب سازی هیپ استفاده می‌کنیم ، بعد از چند مرحله انجام heapify ، آرایه به صورت  $16\ 14\ 15\ 10\ 12\ 27\ 28\ 22$  است ، چند بار عملیات هیپیفی را بر روی ریشه انجام داده ایم؟

راه حل :

با توجه به عملیات مرتب سازی هیپ ، مشاهده می‌شود که در آرایه داده شده ، دو عنصر بزرگ در انتهای قرار دارند و بنابراین درخت ما درخت max است و دوبار heapify کرده ایم.

3.5.4 یک درخت مکس هیپ رای در نظر بگیرید ، کدام آریه زیر میتواند نمایشگر آن باشد؟

۲۵ ۱۲ ۱۶ ۱۳ ۱۰ ۸ ۱۴

۱۲ ۱۲ ۱۶ ۱۳ ۱۰ ۸ ۱۴

۲۵ ۱۴ ۱۶ ۱۳ ۱۰ ۸ ۱۲

۱۵ ۱۴ ۱۲ ۱۳ ۱۰ ۸ ۱۶

راه حل :

در آرایه ای که ایندکس آن از ۰ شروع میشود ، باید هر خانه را با خانه های ایندکس  $1 + 2i$  و  $2 + 2i$  مقایسه کنیم بنابراین گزینه ۳ درست است

4.5.4 با توجه به سوال قبل ، محتوای آرایه بعد از ۲ عملیات پاک کردن چگونه خواهد شد؟

راه حل:

برای هر مرحله پاک کردن ، ابتدا ریشه پاک شده، سپس عنصر آخرین لول جایگزین آن شده و heapify صورت میگیرد ، بنابراین جواب به صورت  $14 13 12 10 8$  خواهد بود

5.5.4 فرض کنید درخت هیپی با  $n$  عنصر داریم و میخواهیم  $n$  عنصر دیگر به آن وارد کنیم ( نه لزوماً یکی پس از دیگری ) ، زمان لازم برای اینکار را به دست آورید

## راه حل

این سوال را می‌توان به ساختن هیپ با  $2n$  عنصر تبدیل کرد که بنابراین جواب به صورت  $O(n)$  است

6.5.4 در یک درخت مکس هیپ شامل  $n$  عدد ، کوچکترین عنصر با چه پیچدگی زمانی یافت میشود؟

## راه حل

در یک درخت مکس هیپ ، کوچکترین عنصر همیشه برگ است، بنابرین باید تمام برگ‌ها را چک کنیم که پیچدگی زمانی آن در بدترین حالت  $O(n)$  است.

7.5.4 برای دو مکس هیپ داده شده با سایز  $n$ ، حداقل زمان لازم برای ساختن یک مکس هیپ چقدر است؟

## راه حل

می‌توان به روش زیر یک هیپ با  $2n$  عنصر را در  $O(n)$  ساخت

یک آرایه با سایز  $2n$  می‌سازیم و عناصر هردو هیپ را در آن کپی می‌کنیم ، تابع ساخت هیپ را برای آرایه به طول  $2n$  صدا زده و در  $O(n)$  درخت هیپ را می‌سازیم.

8.5.4 یک درخت مکس هیپ رای در نظر بگیرید ، کدام آریه زیر می‌تواند نمایشگر آن باشد؟

5 7 12 1 10 13 6 14 17 23

12 7 5 1 10 13 6 14 17 23

12 6 5 1 10 13 7 14 16 23

7 5 12 1 10 13 7 14 17 23

راه حل :

در آرایه ای که ایندکس آن از ۰ شروع می‌شود ، باید هر خانه را با خانه‌های ایندکس  $1 + 2i$  و  $2i + 2$  مقایسه کنیم بنابراین گزینه ۳ درست است

## نمونه سوالات مربوط به بخش 1.5

1.5.1 در قطعه کد مربوط به `count sort` چه لزومی دارد که در حلقه `for` انتهایی، حلقه از  $n-1$  شروع شود؟ در حالیکه با داشتن `for` صعودی با شروع از 1 نیز عمل مرتب سازی انجام میشود.

راه حل:

برای پایدار بودن `count sort` باید این حلقه از  $n-1$  شروع شده و نزولی باشد. چون اعداد در آرایه نهایی که نتیجه مرتب شده عناصر مبایشد، از انتها به ابتدای آرایه پر میشوند، پس باید از دو عددی که یکی هستند، عدد دوم زوینتر وارد آرایه شود و این به معنی شروع شدن حلقه از  $n-1$  میباشد. در غیر اینصورت ترتیب اولیه اعداد یکسان در آرایه نهایی به هم خورده و خاصیت پایدار بودن خود را از دست میدهد.

1.5.2 آرایه ای از  $n$  عدد صحیح از اعداد صحیح 1 تا  $n$  داریم. الگوریتمی با مرتبه زمانی خطی ارایه دهید که به کمک آن بتوان اعداد تکراری را حذف کرد.

راه حل:

چون تمامی اعداد صحیح بوده و در بازه 1 تا  $n$  میباشند میتوان از `count sort` برای مرتب کردن این اعداد استفاده کرد. بنابراین هزینه  $O(n) = O(5n + n)$  را داده و اعداد را مرتب میکنیم. حال از یک استک برای حذف اعداد تکراری استفاده میکنیم. بدین گونه که عدد اول را در استک `push` میکنیم. حال برای عناصر بعدی، اگر عنصر جاری با عنصر سر استک یکی بود، سراغ عنصر بعدی می رویم و در غیر اینصورت آنرا در استک `push` میکنیم و همین کار را برای عناصر بعدی تکرار میکنیم. در آخر، اعداد موجود در استک، جواب مورد نظر ماست.

1.5.3 لیستی از  $n-1$  عدد صحیح که هر عدد بین 1 تا  $n$  است، داده شده است. هیچ دو عدد برابر نیستند. الگوریم بینهای ای ارایه دهید که به کمک آن، عددی را که در این لیست نیامده است پیدا کرد.

راه حل:

چون تمامی اعداد صحیح بوده و در بازه 1 تا  $n$  میباشند پس با هزینه  $O(n)$  و با استفاده از `count sort` میتوان این اعداد را مرتب کرد. حال از ابتدای لیست مرتب شده شروع به پیمایش میکنیم. ایندکس اولین خانه ای که در آن ایندکس مربوط به آن خانه با مقدار خانه متفاوت باشد، جواب مورد نظر است. (با فرض اینکه ایندکس خانه ها از 1 تا  $n$  میباشد). پس با هزینه زمانی  $O(n)$  عدد موردنظر یافت میشود.

1.5.4 فرض کنید  $n$  عدد صحیح بین 1 و  $k$  دارید و میخواهید بدانید چند تا از آنها در بازه  $[a, b]$  هستند به طوریکه  $a < 1$  و  $b < k$  میباشد. الگوریتمی طراحی کنید که بعد از گرفتن زمان  $\Theta(n+k)$  برای پردازش کردن اعداد صحیح، بتواند پاسخ این سوال را با  $\Theta(1)$  بدهد.

راه حل:

ایده اصلی همان `count sort` است.

[i] را تعداد اعداد صحیح ورودی کوچکتر از i تعریف میکنیم. داریم:

```

Range Preprocessing (A, n, k) begin
    for i = 0 to k do
        C[i] = 0
    for i = 1 to n do
        C[A[i]] += 1
    for i = 1 to k do
        C[i] += C[i-1]
End

```

و برای پاسخ به سوال در زمان  $O(1)$  داریم:

```

Range Query (a, b)
    return C[b] - C[a-1]

```

## نمونه سوالات مربوط به بخش 2.5

**2.5.1** نشان دهید چگونه می‌توان  $n$  عدد صحیح را که در بازه  $[1, n^2]$  هستند، در زمان  $\Theta(n)$  مرتب کرد.

راه حل:

می‌دانیم که برای  $n$  تا عنصر **radix sort** که هر کدام از  $d$  رقم تشکیل شده‌اند. اگر در حلقه‌ی داخلی از **count sort** استفاده کنیم ( $d(n+k)$ ) زمان می‌گیرد.(برای عناصر بین  $[1, k]$ )  
حال اگر با این روش بخواهیم به  $O(n)$  برسیم، می‌بایست  $d = O(n)$  و  $k = O(n)$  باشد.  
فرص کمیم هر عنصر  $ai$  را بصورت  $(qi, ai)$  (بنویسیم که در آن  $ri = ai \bmod n$  و  $qi = ai / r$ ) تعریف می‌کنیم.  
از آنجایی که  $ai \in [1, n^2]$  ، پس  $ri \in [1, n]$  و  $qi \in [1, n]$  ، پس  $2 = d$  و  $n = k$ .  
بنابراین، **radix sort** در اینجا  $O(n)$  زمان می‌برد.  
نکته: در واقع این روش می‌تواند  $n$  عدد صحیح در بازه  $[1, 0(1)]$  را در مرتبه‌ی زمانی  $O(n)$  مرتب کند.

**2.5.2** سریعترین الگوریتمی را ارایه دهید که به کمک آن بتوان  $n$  رشته حرفی به طول حد اکثر هشت حرف را مرتب کرد.

راه حل:

از **radix sort** برای این کار استفاده می‌کنیم. می‌دانیم که کد اسکی هر کاراکتر بین 0 تا 255 می‌باشد. پس از سمت راست ترین کاراکتر شورع می‌کنیم و در هر مرحله رشته‌ها را براساس آن کاراکتر مرتب می‌کنیم.(از **count sort** در هر مرحله استفاده می‌کنیم).

چون هزینه هر **count sort** در هر مرحله  $O(n+k) = O(n+256) = O(n)$  می‌باشد و در کل حداقل هشت مرحله این کار تکرار می‌شود. ( $d = 8$ ). پس هزینه نهایی در کل برابر است با:  $O(n) = O(n) * 8$ . پس با هزینه زمانی  $O(n)$  و با استفاده از **sort** میتوان این کار را انجام داد.

**2.5.3** آرایه‌ای از اعداد که در بازه 0 تا 1000 می‌باشند داده شده است. شبه کد تابعی را بنویسید که این آرایه و یک مقدار  $x$  را گرفته و زوج مرتبی از اعداد این آرایه که مجموع آن‌ها برابر مقدار  $x$  می‌باشد را پیدا کرده و باز گرداند. در صورتی‌که چنین زوجی از اعداد یافت نشد، null برگرداند. هزینه زمانی تابع شما باید  $O(n)$  باشد.

راه حل:

چون اعداد در بازه **0 تا 1000** هستند پس میتوان با استفاده از **radix sort** یا **count sort** آرایه داده شده را در زمان **O(n)** مرتب کرد. بسیار دو اشاره گر به ابتدا و انتهای این آرایه مرتب شده میگیریم و شروع به پیمایش آرایه مورد نظر میکنیم. اگر مجموع خانه هایی که اشاره گر ها اشاره میکنند برابر با **x** باشد که به جواب مورد نظر رسیده ایم و جواب را بر میگردانیم. اگر این مجموع کوچکتر از **x** باشد، چون آرایه مورد نظر مرتب شده است پس اشاره گر اشاره کننده به ابتدای آرایه را یکی به سمت راست آرایه حرکت میدهیم و اگر این مجموع بزرگتر باشد اشاره گر دومی را یک واحد به سمت چپ انتقال میدهیم. اگر این دو اشاره گر به هم رسیدند، چنین زوجی یافت نشده است و در آرایه موردنظر وجود ندارند. پس میتوانتابع موردنظر را بدین صورت نوشت:

**SUM\_X(S, x)**

{

**RADIX\_SORT(S);**

**i = 1;**

**j = n;**

**while i < j**

{

**do if S[i] + S[j] < x**

**then i = i+1;**

**else if S[i] + S[j] > x**

**then j = j-1;**

**else**

**return S[i], S[j];**

}

**if i == j**

**then return null;**

}

2.5.4 الگوریتمی ارایه دهید که بوسیله آن بتوان یک لیست از تاریخ ها را مرتب کرد.

راه حل:

هر تاریخ از سه بخش سال و ماه و روز تشکیل شده است که هر بخش عدد صحیح در یک بازه مخصوص است. پس برای این کار سه مرحله زیر را در نظر اعمال میکنیم:

**1:** تاریخ ها را ابتدا بر اساس روز آنها مرتب میکنیم و برای این کار از **count sort** استفاده میکنیم.

**2:** سپس آن ها را بر برا اساس ماه آنها مرتب میکنیم و برای این کار نیز از **count sort** استفاده میکنیم.

**3 :** سپس آن ها را بر برا اساس سالشان مرتب میکنیم و برای این کار از **count sort** استفاده میکنیم.

در هر مرحله هزینه  $O(n)$  میدهیم و در کل سه مرحله داریم. پس هزینه نهایی برابر  $O(n)$  میباشد.

**2.5.5** اگر در **radix sort** در هر مرحله به جای **count sort**، از **merge sort** استفاده شود، آیا هزینه مرتب سازی بهتر خواهد شد؟ بررسی کنید.

راه حل:

چون در  $d$  مرحله از **radix sort**، در هر مرحله میشود پس هزینه آن به صورت  $O(d * (n + k))$  در می آید. حال اگر  $k$  باشد به صورت  $O(d*n)$  خواهد بود.

اگر از **merge sort** استفاده کنیم هزینه کلی به صورت  $O(d * n \log n)$  خواهد بود. پس اگر هزینه هر **count sort** در هر مرحله باشد استفاده از **count sort** بهتر است.

### نمونه سوالات مربوط به بخش 3.5

**3.5.1** میدانیم که در بازه ای بیشتر باشد، باعث میشود که الگوریتم از حالت خطی بودن خارج شود. فرض کنید که تمامی داده ها اعدادی بین 0 تا 1 میباشند. راه حلی ارایه دهید که به بتوان تا حدودی این تمرکز داده ها را از بین برده و باعث پخش شدن آنها شد.

راه حل:

میتوان از توابع ریاضی مانند رادیکال استفاده کرد. میدانیم که ریشه دوم اعداد بین صفر تا یک از خود اعداد بزرگترند. به عنوان مثال ریشه دوم عدد  $10^6$ ، عدد  $10^3$  و ریشه دوم عدد  $10^8$  عدد  $10^4$  میباشد که عدد بزرگتریست. و نیز به عنوان مثال بهوضوح داریم:

$$10^3 - 10^6 > 10^8$$

یعنی توانستیم فاصله بین اعداد را بیشتر کرده و باعث پخش شدن داده ها شویم. اگر این تمرکز داده ها در بازه ای بیشتر باشد میتوانیم از ریشه های بالاتر مانند ریشه ششم و امثال آن نیز برهه ببریم.

**3.5.2** اگر در **bucket sort** برای مرتب سازی هر **bucket**، از **insertion sort** به جای **merge sort** استفاده کنیم چه تغییری در هزینه زمانی این الگوریتم حاصل میشود؟

راه حل:

بهترین حالت:  $n$  عنصر به صورت یکنواخت بین  $k$  سطل تقسیم شوند.

در این صورت هر سطل  $n/k$  عنصر دارد.

هزینه مرتب سازی هر سطح ( $O(n/k * \log(n/k))$ ) میباشد.

اگر  $k = \Theta(n)$  باشد، هزینه کل برابر است با:

$$K * (n/k * \log(n/k)) = n \log(n/k) = O(n) : k = \Theta(n)$$

بدترین حالت: تمامی عناصر در یک **bucket** قرار گیرند.

در این صورت هزینه کل برابر با هزینه **merge sort** خواهد بود که برابر است با  $O(n \log n)$ .

پس در کل میتوان گفت که در بهترین حالت باز هزینه زمانی  $O(n)$  پایرگاست، ولی در بدترین حالت هزینه زمانی کمتری نسبت به زمانی که از **insertion sort** استفاده میشود، دارد. (هزینه زمانی **merge sort** برای  $O(n \log n)$  و هزینه زمانی  $O(n^2)$  برای **insertion sort**).

### 3.5.3 با استفاده از **bucket sort** داده های زیر را مرتب کنید. **bucket** ها در هر مرحله نشان دهید.)

165 , 659 , 425 , 159 , 893 , 544 , 785

راه حل:

Bucket 1: 165 , 159      sorted Bucket1: 159 , 165

Bucket 2: 425      sorted Bucket2: 425

Bucket 3: 569 , 544      sorted Bucket3: 544 , 569

Bucket 4: 785      sorted Bucket4: 785

Bucket 5: 893      sorted Bucket5: 893

## نمونه سوالات مربوط به بخش 5.5

1.5.5 برای اعداد زیر مرتب سازی حبابی را دنبال کنید.

50 , 57 , 99 , 34 , 56 , 89

راه حل:

در هر فاز بزرگترین عنصر انتهای لیست مربوطه قرار میگیرد و در هر مرحله لیست کوچکتر حاصل میشود و در نهایت لیست دو عنصره مرتب میشود.

مرحله اول : 50 , 57 , 99 , 34 , 56 , 89 -->50 , 57 , 34 , 56 , 89 , 99

مرحله دوم : 50 , 57 , 34 , 50 , 89 -->50 , 34 , 56 , 57 , 89

مرحله سوم : 50 , 34 , 56 , 57 -->34 , 50 , 56 , 57

مرحله چهارم : 34 , 50 , 56 -->34 , 50 , 56

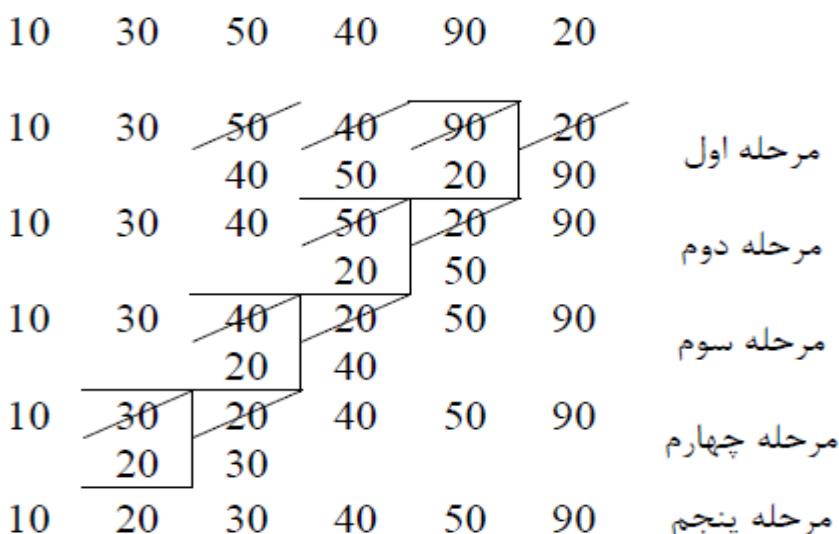
لیست نهایی : 34 , 50 , 56 , 57 , 89 , 99

---

2.5.5 برای اعداد زیر مرتب سازی حبابی را دنبال کنید.

10 , 30 , 50 , 40 , 90 , 20

راه حل:



## نمونه سوالات مربوط به بخش 5.6

1.5.6 برای اعداد زیر مرتب سازی درجی را دنبال کنید..

70 , 57 , 99 , 34 , 56 , 89

راه حل:

نخست عنصر دوم را با عنصر اول مقایسه کرده لیست دو عنصری را مرتب میکنیم سپس عنصر سوم با دو عنصر قبلی تشکیل لیست مرتب سه تایی میدهند و الی آخر.

مرحله اول : 70 , 57 --> 57 , 70

مرحله دوم : 57 , 70 , 99 --> 57 , 70 , 99

مرحله سوم : 57 , 70 , 99 , 34 --> 57 , 70 , 34 , 99 --> 34 , 57 , 70 , 99

مرحله چهارم : 34 , 57 , 70 , 99 , 56 --> 34 , 57 , 70 , 56 , 99 --> 34 , 56 , 57 , 70 , 99

مرحله پنجم : 34 , 56 , 57 , 70 , 99 , 89 --> 34 , 56 , 57 , 70 , 89 , 99

---

2.5.6 آرایه زیر را به روش درجی مرتب کنید.

4	8	5	2	6
---	---	---	---	---

راه حل:

n	i	j	y
5	2	1	8
3	2	5	
4	1	2	
	3	6	
	2		
	1		
	0		
	4		
	3		

A	1	2	3	4	5
4	4	8	5	2	6
4	4	8			
4	5	8			
2	4	5	8		
2	4	5	6	8	

آرایه زیر را به روش درجی مرتب نمایید.

7	8	5	2	4	6	3
---	---	---	---	---	---	---

راه حل:

7 8 5 2 4 6 3

7 8 | 5 2 4 6 3

7 5 | 8 2 4 6 3

5 7 | 8 2 4 6 3

5 7 8 | 2 4 6 3

2 5 7 8 | 4 6 3

2 4 5 7 8 | 6 3

2 4 5 6 7 8 | 3

2 3 4 5 6 7 8 |

---

### نمونه سوالات مربوط به بخش 5.7

1.5.7 مرتب سازی  $n$  عدد ذخیره شده در  $A$  را در نظر بگیرید که ابتدا کوچکترین عنصر  $A$  را یافته و آنرا با عنصر  $A[1]$  تعویض میکنیم . سپس دومین عنصر کوچکتر  $A$  را یافته و آنرا با  $A[2]$  تعویض میکنیم . این روند را برای  $n-1$  عنصر اول ادامه میدهیم. شبه کدی برای این الگوریتم که به مرتب سازی انتخابی معروف است بنویسید . چرا لازم است این الگوریتم برای  $n-1$  عنصر اول به جای  $n$  عنصر اجرا شود؟ زمان اجرای الگوریتم مرتب سازی را در بهترین حالت و در بدترین حالت بیان کنید.

راه حل:

*Selection\_Sort(A)*

$n \leftarrow length[A]$

*for j ← 1 to n-1*

*do smallest  $\leftarrow j$*

*for  $i \leftarrow j+1$  to  $n$*

*do if  $A[i] < A[smallest]$*

*then smallest  $\leftarrow i$*

*exchange  $A[j] \leftrightarrow A[smallest]$*

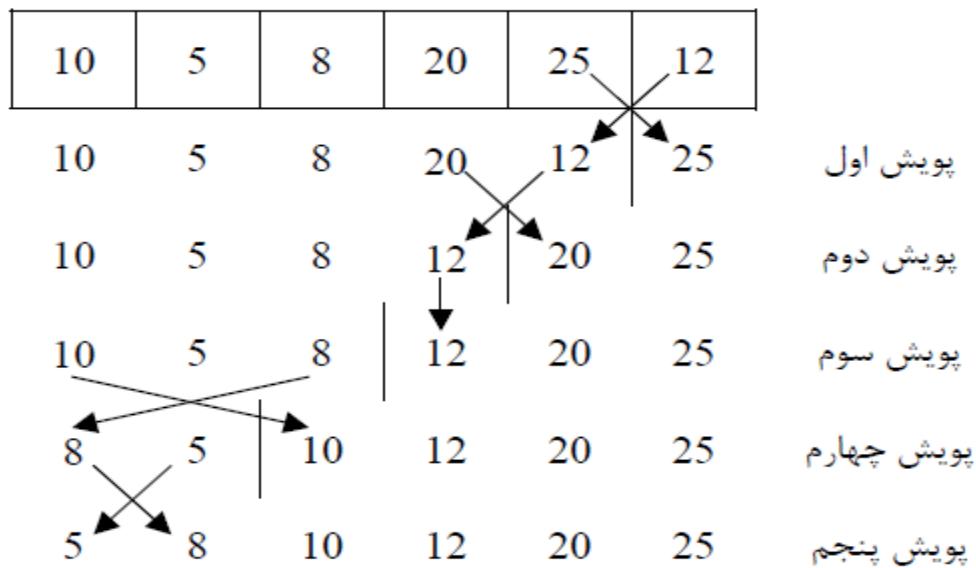
زیر آرایه  $A[1...n]$  شامل  $j$ -ج کوچکترین عنصر در آرایه  $A[1...n]$  میباشد و همچنین این آرایه به صورت مرتب شده میباشد.  $n-1$  عنصر اول در زیر آرایه  $A[1...n-1]$  شامل  $n-1$  کوچکترین عنصر مرتب شده میباشد بنابراین عنصر  $A[n]$  بزرگترین عضو این آرایه میباشد. هزینه اجرای این الگوریتم در هر دو حالت مذکور  $n$  به توان 2 میباشد.

---

آرایه زیر را به روش انتخابی مرتب نمایید.

10	5	8	20	25	12
----	---	---	----	----	----

راه حل:



آرایه زیر را به روش انتخابی مرتب کنید.

2	8	4	1	7
---	---	---	---	---

راه حل:

1)  $2 \underline{8} 4 1 \underline{7} \rightarrow 2 7 4 1 8$

2)  $2 \underline{7} 4 \underline{1} 8 \rightarrow 2 1 4 7 8$

3)  $2 1 \underline{4} 7 8 \rightarrow 2 1 4 7 8$

4)  $\underline{2} 1 4 7 8 \rightarrow 1 2 4 7 8$

در مرحله‌ی اول، کل لیست از ابتدا تا انتهای بررسی شده و بزرگترین عنصر با عنصر انتهایی لیست نامرتب جابجا می‌شود.

در مرحله‌ی دوم، پیمایش از ابتدای لیست تا عنصر چهارم صورت گرفته و بزرگترین عنصر با عنصر انتهایی آن جابجا می‌شود.

علت این که چرا عنصر پنجم بررسی نمی‌شود کاملاً مشخص است. این عنصر در مرحله‌ی قبل به عنوان بزرگترین عنصر به انتهای

لیست منتقل شده است و به طور حتم نیاز به جا به جایی ندارد.

در مرحله‌ی سوم، عناصر اول تا سوم بررسی شده و بزرگترین عنصر به انتهای آن منتقل می‌شود:

و در مرحله‌ی آخر دو عنصر باقیمانده مقایسه می‌شوند:

و به این ترتیب لیست مرتب می‌شود.

---

## نمونه سوالات مربوط به بخش 5.8

1.5.8 آرایه مرتب شده از اعداد صحیح به شما داده شده است . این  $K$  آرایه در مجموع شامل  $n$  عدد هستند میخواهیم این آرایه ها را ادغام و آرایه ای مرتب شامل  $n$  عدد به دست بیاوریم . الگوریتمی از  $O(n \log k)$  ارایه دهید که این کار را انجام دهد.

راه حل:

ابتدا یک آرایه به طول  $n$  در نظر میگیریم . سپس عناصر  $k$  لیست را عنصر به عنصر با یکدیگر مقایسه نموده و کوچکترین عضو را از بین  $k$  آرایه پیدا میکنیم (به عنوان مثال عنصر  $A$ ) در آرایه مذکور میریزیم و عنصر  $A$  را از آرایه ای که شامل  $A$  میباشد حذف میکنیم و تمامی عناصر آرایه ای را که حاوی عنصر  $A$  بود را یک واحد شیفت میدهیم و این کار را همچنان ادامه میدهیم تا تمامی  $K$  آرایه خالی شود .

آرایه زیر را به روش ادغامی مرتب نمایید.

5	1	7	2
---	---	---	---

راه حل:

5	1	7	2
5	1	7	2
5	1	7	2
1	5	2	7
1	2	5	7

L = 3	U = 3	L = 4	U = 4
L = 3	U = 3	i = 3	
L = 1	U = 1	L = 2	U = 2
L = 1	U = 2	i = 1	
L = 1	U = 4	i = 2	

آرایه زیر را به روش ادغامی مرتب نمایید.

5	3	1	4
---	---	---	---

راه حل::

A	1	2	3	4
	5	3	1	4

Merge sort ( 1 , 4 )

1	2	3	4
3	5	1	4

1	3	4	5
---	---	---	---

L = 4	U = 4
L = 3	U = 4
L = 3	U = 4    i = 3
L = 2	U = 2
L = 1	U = 1
L = 1	U = 2    i = 1
L = 1	U = 4    i = 2

جدول زیر را کامل نمایید.

نام الگوریتم	بهترین حالت	حالات متوسط	بدترین حالت	ویژگی ها
مرتب سازی حبابی				
مرتب سازی درجی				
مرتب سازی انتخابی				
مرتب سازی ادغامی				

راه حل:

نام الگوریتم	بهترین حالت	حالات متوسط	بدترین حالت	ویژگی ها
مرتب سازی حبابی	$O(n)$	$O(n^2)$	$O(n^2)$	پایدار است و درجا
مرتب سازی درجی	$O(n)$	$O(n^2)$	$O(n^2)$	پایدار است و درجا
مرتب سازی انتخابی	$O(n^2)$	$O(n^2)$	$O(n^2)$	پایدار نیست و درجا
مرتب سازی ادغامی	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	پایدار است و غیر درجا

## نمونه سوالات مربوط به بخش ۵.۸ - ۵.۹

۵.۸.۱ الگوریتم مرتبسازی سریع را به گونه‌ای تغییر دهید که اعداد را به صورت غیر-صعودی مرتب کند.  
راه حل:

در تابع `partition`, عناصر بزرگ‌تر از `pivot` را به سمت چپ عنصر آن منتقل می‌کنیم و عناصر کوچک‌تر از آن را به سمت راست آن.

---

۵.۸.۲ در الگوریتم مرتبسازی سریع اگر تمامی عناصر مقدار متفاوت داشته باشند، بزرگ‌ترین عنصر حداکثر چندبار جایه‌جا می‌شود؟  
راه حل:

حداکثر  $n-1$  بار جایه‌جا می‌شود. چرا که هر عنصر حداکثر  $n-1$  بار می‌تواند جایه‌جا شود و پس از آن که به جای درست خود بررسد دیگر جایه‌جا نمی‌شود. اگر اعداد ورودی جایگشت اعداد ۱ تا  $n$  به صورت نزولی باشد، در آن صورت بزرگ‌ترین عنصر  $n-1$  بار جایه‌جا خواهد شد.

---

۵.۸.۳ زمان اجرای مرتبسازی سریع را در هنگامی که همه عناصر مقدارشان یکسان باشد، تحلیل کنید.  
راه حل:

در این حالت در تابع `partition` تمامی عناصر در یک سمت عنصر `pivot` قرار می‌گیرند و در این حالت الگوریتم مرتبسازی سریع هزینه‌ای برابر  $\theta(n^2)$  خواهد داشت.

---

۵.۸.۴ نشان دهید بهترین زمان اجرای مرتبسازی سریع  $\Omega(n \log n)$  است.  
راه حل:

داریم:

$$T(n) = \min(T(q) + T(n-q-1)) + \theta(n) \quad (1 \leq q \leq n-1)$$

هنگامی  $T$  کمینه خواهد بود که  $q = \frac{n}{2}$ . و در این حالت خواهیم داشت:

۵.۸.۵ نشان دهید مرتبسازی سریع از  $O(n^2)$  است.  
راه حل:

داریم:

$$T(n) = \min(T(q) + T(n-q-1)) + \theta(n) \quad (1 \leq q \leq n-1)$$

هنگامی  $T$  بیشینه خواهد بود که  $q=0$  یا  $q=n-1$  باشد. پس

---

۵.۸.۶ نشان دهید مرتب‌سازی سریع از  $(n^2) \Omega$  را حل:

به ازای جایگشتی از اعداد ۱ تا  $n$  که به صورت نزولی مرتب شده‌اند، مرتب‌سازی سریع از  $\theta(n^2)$  هزینه خواهد داشت. پس مرتب‌سازی سریع از  $(n^2) \Omega$  است.

---

۵.۸.۷ پایداری الگوریتم مرتب‌سازی سریع را بررسی کنید.  
راه حل:

این موضوع به پیاده‌سازی تابع **partition** برمی‌گردد. در پیاده‌سازی مرسوم تابع **partition**، عنصر آخر به عنوان **pivot** در نظر گرفته‌می‌شود و تمامی عناصر برابر با آن در سمت چپ آن قرار خواهند گرفت. بقیه عناصری که با هم مساوی هستند نیز ترتیب‌شان عوض نمی‌شود. پس الگوریتم مرتب‌سازی سریع پایدار است.

---

۵.۸.۸ درجا بودن یا نبودن الگوریتم مرتب‌سازی سریع را بررسی کنید.  
راه حل:

چون در طی مرتب‌سازی سریع از حافظه اضافی استفاده نمی‌شود این الگوریتم درجا است.

---

۵.۸.۹ چرا زمان اجرای الگوریتم‌ها را در حالت میانگین بررسی می‌کنیم؟  
راه حل:

زیرا احتمال برخورد با بدترین حالت بسیار کم است و پخش کردن داده‌ها به صورت تصادفی می‌تواند این احتمال را بسیار کمتر هم بکند. از این رو ما در اکثر موارد هزینه‌ای برابر حالت میانگین برای الگوریتم‌ها یمان خواهیم داشت.

---

۵.۸.۱۰ الگوریتم مرتب‌سازی سریع در حالت میانگین چه هزینه‌ای دارد؟  
راه حل:

$$O(n \log n)$$

---

۵.۸.۱۱ فرض کنید در هر مرحله از مرتب‌سازی سریع، آرایه به دو بخش با نسبت اندازه  $k$  و  $1-k$  تقسیم شود که  $k$  یک عدد ثابت با شرط  $0 \leq k \leq \frac{1}{2}$  است. نشان دهید حداقل عمق یک برگ در درخت بازگشت (Recursion Tree) است.  
راه حل:

حدقل ارتفاع این‌گونه به دست می‌آید که در هر مرحله قسمت به کوچک‌تر برویم. یعنی در هر مرحله اندازه هر قسمت از  $n$  به کاهش پیدا می‌کند. پس بعد از  $i$  مرحله تعداد عناصر برابر  $k^i n$  خواهد بود. وقتی که به یک برگ می‌رسیم تعداد عناصر برابر ۱ خواهد بود.  $m$  را کوچک‌ترین مقداری در نظر بگیرید که  $k^m = 1/n$ . (یعنی اینکه به یک برگ رسیده‌ایم). پس

---

اگر از دو طرف لگاریتم بگیریم خواهیم داشت:

$$m = \frac{-\log n}{\log(1-k)}$$

۵.۸.۱۲ سوال قبل را در نظر بگیرید. ثابت کنید حداکثر عمق یک برگ در درخت بازگشت (Recursion Tree) است.

راه حل:

حداکثر ارتفاع این گونه به دست می‌آید که در هر مرحله قسمت به بزرگ‌تر برویم. یعنی در هر مرحله اندازه هر قسمت از  $n$  به  $(1-k)^i n$  کاهش پیدا می‌کند. پس بعد از  $i$  مرحله تعداد عناصر برابر  $(1-k)^i n$  خواهدبود. وقتی که به یک برگ مرسیم تعداد عناصر برابر ۱ خواهدبود.  $m$  را بزرگ‌ترین مقداری در نظر بگیرید که  $(1-k)^m n = 1$ . (یعنی اینکه به یک برگ

$$m = \frac{-\log n}{\log(1-k)}$$
 که اگر از دو طرف لگاریتم بگیریم خواهیم داشت:  $(1-k)^m = 1/n$  رسیده‌ایم). پس

۵.۸.۱۳ مرتب‌سازی سریع در حالتی تحلیل کنید که در هر مرحله عناصر را به ۳ قسمت تقسیم کنیم.  
راه حل:

در این حالت با حالت عادی تفاوتی نخواهد کرد و فقط در بهترین حالت و حالت میانگین لگاریتم در مبنای ۳ خواهدبود که باز هم پیچیدگی الگوریتم تغییر نمی‌کند.

۵.۸.۱۴  $n$  عدد را در نظر بگیرید که در آن‌ها  $k$  عدد متفاوت وجود دارد و هر عدد دقیقاً  $\frac{n}{k}$  بار تکرار شده است. الگوریتمی

از  $O(n \log k)$  ارایه دهید برای مرتب کردن این اعداد.  
راه حل:

مانند مرتب‌سازی سریع عمل می‌کنیم اما در هر مرحله ابتدا با هزینه  $O(n)$  میانه اعداد را پیدا کرده و آن را به عنوان pivot در نظر

می‌گیریم. همچنین وقتی به  $\frac{n}{k}$  عدد رسیدیم که همه آن‌ها برابر بودند الگوریتم را خاتمه می‌دهیم. پس بنابراین  $\log k$  مرحله داریم که در هر مرحله  $O(n \log k)$  هزینه می‌کنیم. پس در کل الگوریتم از  $O(n)$  خواهدبود.

## 11.1. کولیژن یا تصادم را تعریف کنید

حل : اگر برای دو داده متفاوت مقدار تابع هش یکسان دراید دو داده باید در یک خانه قرار بگیرند به این پیشامد تصادم میگویند.

## 11.4.2. سیستم اپن ادرسینگ در هش را توضیح دهید

حل : هر گاه برای دو با چند داده یک خانه از تابع هش بیرون میابد تابع هش دوباره برای این داده صدا میشود یعنی اگر خانه ای مرتبط با داده مذبور پر بوده باشد تابع برای این داده با یک مقدار جدید صدا میشود تا خانه ای دیگر به ان اختصاص داده شود ، این امر نه تنها از تصادم جلوگیری میکند بلکه داده ها را در جدول پخش میکند و مانع تجمع در یک محدوده از حافظه میشود. که انواع مختلفی تابع میتوان برای ان تعریف نمود مثلا بررسی خطی و درجه دو.

## 11.4.3. برای جلوگیری از تصادم راه حلی ارایه دهید

حل : راه های مختلفی نظیز گرفتن لیست پیوندی برای ذخیره موارد با ادرس هش یکسان در یک خانه و نیز استفاده از سیستم اپن ادرسینگ برای جلوگیری از تصادم میتواند مورد استفاده قرار گیرد.

داده های زیر را در یک جدول هش به گونه ای ذخیره کنید که هیچ تداخل یا کولیژنی رخ ندهد. (میدانیم حداقل دو داده با حروف اول یکسان وجود دارد و همگی حروف انگلیسی هستند)

AA aa Bd ss Ds hj jj mj xy xa cd

حل یک جدول هش در نظر میگیریم که حداقل خانه داشته باشد با استفاده از سیستم ادرس دهی اپن ادرسینگ داده ها را در جای مناسب قرار میدهیم

## 11.3.5. تعداد زیادی داده با مقادیر بزرگ داریم ، یک تابع هش برای آن بیان کنید.

حل : یک ارایه با سایز بزرگ در نظر گرفته به طوری که تعداد داده ذخیره شده در هر سطر از  $O(1)$  باشد ، حال مقدار کد اسکی داده ها برای هر داده (فرض کنید همه رشته هستند) تا دهمین عدد اول برای هر عدد مناسب با خانه عدد در عدد ضرب کرده همه را با هم جمع میکنیم و در نهایت به تعداد خانه خود تقسیم میکنیم ، در خانه اول هر سطح تعداد داده ذخیره شده در سطر نگه داری میکنیم و اگر تعداد از 10000 تا بیشتر شد از روش open addressing استفاده میکنیم در غیر این صورت به صورت زنجیره در یک سطح قرار میدهیم .

## 11.3.6. کد قسمت قبل را بنویسید .

```
Vector<int> prim(10, 2,3,5,7,11,13,17,19,23,29);
Vector<vector<string> > data; sum = 0;
String s ; Cin>> s; for(int I = 0; i < 29; I++){ if( s.size() < i){break;} else sum = prim[i] * s[prime[i]];}
Int index = sum / size; if(is_more_than_thou(data[index]) ){ h(s, 1)} else {
data[index].push_back(s)}
اين تابع به خانه اول وکتور در ردیف شماره index يک واحد اضافه میکند .
```

## 11.7. فایده جدول درهم سازی نسبت به سایر ساختمان های داده چیست؟

حل : در واقع هش یک دیکشنری است بین معنا که حذف و اضافه کردن و جست و جو در آن با هزینه (1) O انجام میشود ، اما ممکن است میزان مصرف حافظه ان بیشتر باشد .

---

## 11.3.8. هدف از مکاشفه برای یک تابع هش مناسب چیست ؟

حل : کاهش تعداد تصادم ها و نیز استفاده بهینه از حافظه موجود تا جایی که تعداد خانه های خالی به حداقل برسد . علاوه بر این مینوان جست و جو در زمان واقعی را نیز به آن افزود .

---

## 11.9. اگر اسامی تمامی خیابان های کلان شهر های تهران را بخواهیم در یک جدول هش نگهداری کنیم به گونه ای که تمامی خیابان های متقاطع را سریعاً بیابیم تا در یک برنامه نقشه و راهبری استفاده کنیم چه روشی را ارایه میکنید.

حل : برای هر خیابان بعد از ذخیره در محل موردنظر در جدول هش ایندکس تمام خیابان های متقاطع با آن را در آن خانه اضافه میکنیم و نام خیابان متقاطع را نیز در جای خود قرار میدهیم و همین کار را برای ان انجام میدهیم بین صورت با جست و جوی یک خیابان به ترتیب نواحی اطراف آن رویت میشود . باید به صورت ریکر سیو خیابان ها نمایش داده شود .

---

## 11.10. اگر بخواهیم تمام اطلاعات ایرانیان را در یک جدول درهم ذخیره کنیم ، با توجه به این که شماره ملی افراد یکتاست ولی از صفر و یا یک شروع نمیشود چه راهی را دنبال میکنیم؟

حل : ابتدا با توجه به سه رقم اول شماره ملی خانه های جدول را شماره گذاری میکنیم و سپس برای هر 3 رقم مجزا شده 100000 خانه اختصاص میدهیم ، حال برای هر شماره ابتدا 3 رقم را تفکیک میکنیم و بعد باقی مانده 7 رقم باقی را به 100000 میگیریم اگر خانه پر بود از این ادرسینگ استفاده میکنیم ، اما روش کار به این صورت است که ابتدا بدون این ادرسینگ انجام میدهیم بعد از آن وقتی همه به صورت طبیعی در جای خود قرار گرفته اند این روش بهره میگیریم اگر فردی اضافه شد او را به خانه دقیق خود میبریم و برای فردی که میخواهیم جایه جا کنیم این ادرس میکنیم .

## نمونه سوالات مربوط به بخش ۷

۱,۷ یکی از راه های پیمایش، پیمایش میان وند است(**in order**). در این پیمایش ابتدا فرزند چپ سپس پدر و بعد از آن فرزند راست خوانده میشود. رایج ترین نوع پیاده سازی این الگوریتم، استفاده از توابع بازگشتی است. حال شما سعی کنید این پیمایش را بدون استفاده از هیچ تابع بازگشتی پیاده سازی کنید و پیچیدگی زمانی آنرا نیز بدست آورید.

راه حل:

پیاده سازی الگوریتم به صورت زیر است:

(۱) ابتدا یک پشته خالی تعریف میکنیم.

(۲) سپس ریشه گراف را به عنوان نود فعلی در نظر میگیریم.

(۳) نود فعلی را در پشته **push** میکنیم و نود فعلی را برابر با بچه چپ نود فعلی در نظر میگیریم.  
تا زمانی که نود فعلی برابر **NULL** (**current = current->left**) شود.

(۴) اگر نود فعلی **NULL** بود و پسته ما خالی نبود،

الف) عنصر بالای پسته را **pop** میکنیم.

ب) عنصر **pop** شده را چاپ میکنیم و نود فعلی را برابر با بچه راست عنصر **pop** شده قرار میدهیم.  
( **current = popped\_item->right** )

ج) برو به مرحله سه.

(۵) اگر نود فعلی **NULL** بود و پشته نیز خالی بود الگوریتم به پایان رسیده است.

راه حل بهتر دیگری نیز وجود دارد به دلیل بهینه نبودن استفاده از پشته از نظر حافظه برای تبدیل الگوریتم های بازگشتی به غیر بازگشتی که به این صورت است:

(۱) نود فعلی را ریشه درخت در نظر میگیریم

(۲) تا زمانی که نود فعلی **NULL** نشده است:

اگر نود فعلی بچه چپ نداشت:

الف) داده موجود در نود فعلی را چاپ میکنیم

ب) نود فعلی را برابر بچه راست خود میگذاریم ( **current = current->right** )

در غیر این صورت:

الف) زیر درخت سمت چپ که نود فعلی ریشه آن است را در نظر میگیریم و نود فعلی را برابر بجه راست راسترین نود این زیر درخت میگذاریم

( current = current->left ) به بجه چپ این نود میرویم

پیچیدگی زمانی نیز واضح است در هر دو حالت برابر  $O(n)$  است.

---

۲،۷ الگوریتمی غیر بازگشتی برای پیمایش پیشوندی یک درخت بنویسید.

راه حل:

الگوریتم را در مراحل زیر پیاده سازی میکنیم.

۱) ابتدا یک پشته خالی در نظر میگیریم و ریشه را در آن push میکنیم.

۲) تا زمانی که پشته خالی نبود مراحل زیر را انجام میدهیم.

الف) یک عنصر را از پشته pop کن و آنرا چاپ کن.

ب) بجه سمت راست عنصر چاپ شده را در پشته push کن.

ج) بجه سمت چپ عنصر چاپ شده را در پشته push کن.

۳) با خالی شدن پشته الگوریتم به پایان میرسد.

راه حل بدون استفاده از پشته:

```
void BinarySearchTree::preorderNonRecursive(BinarySearchTreeNode* root)
{
    BinarySearchTreeNode* current = root;
    while (current) {
        if (!current->visited) {
            current->print();
            current->visited = true;
        }

        if (current->left && !current->left->visited) {
            current = current->left;
            continue;
        } else if (current->right && !current->right->visited) {
            current = current->right;
        }
    }
}
```

```

        continue;
    }

    if (current->left)
        current->left->visited = false;
    if (current->right)
        current->right->visited = false;

    current = current->parent;
};

if (root)
    root->visited = false;
}

```

---

۳،۷ پیمایش پسوندی به این صورت است که ابتدا فرزند چپ و راست و سپس پدر فرزندان مشاهده میشود. (**Post Order**)  
پیاده سازی این الگوریتم پیمایش به صورت بازگشتی روش متداول آن است. حال سعی کنید الگوریتمی به صورت غیر بازگشتی برای پیاده سازی این پیمایش ارایه دهید. در مرحله بعد سعی کنید با یک ساختمان داده نیز راه حلی را ارایه دهید.

راه حل:

راه حل اول: پیاده سازی با دو پشته:

(۱) ابتدا ریشه درخت را در پشته اول **push** میکنیم.

(۲) تا زمانی که پشته اول خالی نیست:

الف) یک عنصر را از پشته اول **pop** و در پشته دوم **push** میکنیم

ب) به ترتیب بچه چپ و راست عنصر **pop** شده را در پشته اول **push** میکنیم

(۳) محتوای پشته دوم را به ترتیب **pop** و چاپ میکنیم

راه حل دوم: پیاده سازی با یک پشته:

(۱) ابتدا یک پشته خالی تعریف میکنیم.

(۲) تا زمانی که نود فعلی **NULL** نشده:

الف) ابتدا بچه راست سپس بچه چپ را در پشته **push** میکنیم

ب) نود فعلی را برابر بچه چپ نود فعلی قرار میدهیم ( **current = current->left** )

(۳) یک عنصر را از پشته **pop** میکنیم و نود فعلی را برابر آن قرار میدهیم (**current = popped**)

الف) اگر عنصر **pop** شده بچه راست داره و آن بچه راست عنصر بالای پشته است، بچه راست را از پشته **pop** میکنیم و نود فعلی را در پشته **push** میکنیم و سپس نود فعلی را برابر بچه راست نود فعلی قرار میدهیم.

ب) در غیر این صورت، نود فعلی را چاپ کرده و سپس نود فعلی را **NULL** میگذاریم.

(۴) مراحل ۲ و ۳ را تا زمانی که پشته خالی نیست تکرار میکنیم.

راه حل بدون استفاده از پشته:

```
void BinarySearchTree::postorderNonRecursive(BinarySearchTreeNode* root) {  
    BinarySearchTreeNode* current = root;  
    while (current) {  
        if (current->left && !current->left->visited) {  
            current = current->left;  
            continue;  
        } else if (current->right && !current->right->visited) {  
            current = current->right;  
            continue;  
        }  
  
        if (!current->visited) {  
            current->print();  
            current->visited = true;  
        }  
  
        if (current->left)  
            current->left->visited = false;  
        if (current->right)  
            current->right->visited = false;  
  
        current = current->parent;  
    };  
    if (root)  
        root->visited = false;  
}
```

---

۴.۷ الگوریتمی غیر بازگشتی ارایه دهید که با دریافت دو گراف به ما بگوید آیا این دو گراف مشابه هستند یا خیر.

راه حل:

ابتدا دو صفت را تعریف میکنیم و به نام های **q1** و **q2** مینامیم. سپس ریشه های دو گراف را به تابع الگوریتم خود میدهیم که نام های **root1** و **root2** دارند. ایتدا حالت های خاص را چک میکنیم به این صورت که اگر دو گراف داده شده تهی بودند **true** بازگردانده شود. اگر تنها یکی از آنها تهی بود **false** برگردانده شود.

سپس اگر دو گراف از این شرایط عبور کردند یعنی هیچ کدام تهی نیستند. حال **root1** و **root2** را به ترتیب در **q1** و **q2** میکنیم. سپس تا زمانی که **q1** و **q2** هر دو غیر خالی هستند:

عنصر جلو دو صفت را مقایسه میکنیم و اگر برابر نبودند **false** برگردانده میشود. عناصر جلو صفت را از صفت خارج میکنیم. حال چک میکنیم اگر هر دو عنصر بچه چپ داشتند در صفات ها وارد میکنیم در غیر این صورت اگر یکی داشت و دیگری نداشت **false** برگردانده میشود. همین کار را با بچه های چپ انجام میدهیم.

اگر الگوریتم به پایان خود رسید و مقداری برگردانده نشده بود **true** برمیگردانیم.

```
// Iterative method to find height of Binary Tree
bool areIdentical(Node *root1, Node *root2)
{
    // Return true if both trees are empty
    if (!root1 && !root2) return true;

    // Return false if one is empty and other is not
    if (!root1 || !root2) return false;

    // Create an empty queues for simultaneous traversals
    queue<Node *> q1, q2;

    // Enqueue Roots of trees in respective queues
    q1.push(root1);
    q2.push(root2);

    while (!q1.empty() && !q2.empty())
    {
        // Get front nodes and compare them
        Node *n1 = q1.front();
        Node *n2 = q2.front();

        if (n1->data != n2->data)
            return false;

        // Remove front nodes from queues
        q1.pop(), q2.pop();

        /* Enqueue left children of both nodes */
        if (n1->left && n2->left)
        {
            q1.push(n1->left);
            q2.push(n2->left);
        }
    }
}
```

```

    q2.push(n2->left);
}

// If one left child is empty and other is not
else if (n1->left || n2->left)
    return false;

// Right child code (Similar to left child code)
if (n1->right && n2->right)
{
    q1.push(n1->right);
    q2.push(n2->right);
}
else if (n1->right || n2->right)
    return false;
}

return true;
}

```

---

۵.۷ الگوریتمی غیر بازگشتی و بهینه برای بدست آوردن عدد  $x$  به توان  $y$  با پیچیدگی زمانی  $O(\log y)$  ارایه دهد.

راه حل:

اگر از حلقه به صورت معمول استفاده کنیم به پیچیدگی  $O(n)$  میرسیم برای همین راه حل را با استفاده از داده های به دست آمده در مراحل اجرا بهینه تر میکنیم.

```

/* Iterative Function to calculate (x^y) in O(logy) */
int power(int x, unsigned int y)
{
    int res = 1; // Initialize result

    while (y > 0)
    {
        // If y is odd, multiply x with result
        if (y & 1)
            res = res*x;

        // n must be even now
        y = y>>1; // y = y/2
        x = x*x; // Change x to x^2
    }
    return res;
}

```

---

۶،۷ الگوریتم جستجو دودویی را بصورت غیر بازگشتی بنویسید و سعی کنید راه شما از نظر حافظه نیز بهینه تر از الگوریتم معمول جستجو دودویی باشد و پیچیدگی زمانی آنرا نیز به دست آورید.

راه حل:

تکه کد زیر مراحل الگوریتم را به سادگی شرح میدهد.

```
// A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r-l)/2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was not present
    return -1;
}
```

---

۷،۸ الگوریتمی غیر بازگشتی و بهینه از نظر حافظه برای پیدا کردن عمق یک درخت ارایه دهید.

راه حل:

در هر مرحله مل پیمایش Level Order میزنیم و راس ها را در صفات اضافه میکنیم. تکه کد زیر به خوبی این مراحل را شرح میدهد.

```

// Iterative method to find height of Binary Tree
int treeHeight(node *root)
{
    // Base Case
    if (root == NULL)
        return 0;

    // Create an empty queue for level order traversal
    queue<node *> q;

    // Enqueue Root and initialize height
    q.push(root);
    int height = 0;

    while (1)
    {
        // nodeCount (queue size) indicates number of nodes
        // at current level.
        int nodeCount = q.size();
        if (nodeCount == 0)
            return height;

        height++;

        // Dequeue all nodes of current level and Enqueue all
        // nodes of next level
        while (nodeCount > 0)
        {
            node *node = q.front();
            q.pop();
            if (node->left != NULL)
                q.push(node->left);
            if (node->right != NULL)
                q.push(node->right);
            nodeCount--;
        }
    }
}

```

---

۸,۷ مساله برج هانوی به این صورت است که سه میله و تعدادی دیسک با اندازه های متفاوت و به ترتیب در یکی از میله ها گذاشته شده است. هدف بازی جا به جایی همه دیسک ها به همان ترتیب به یک میله دیگر است. به صورتی که در هر مرحله فقط حق جا به جا کردن یک دیسک وجود دارد و نمیتوان دیسک بزرگتر را بالای دیسک کوچکتر از خود قرار داد. این الگوریتم را به ازای تعداد دلخواه دیسک بصورت غیر بازگشتی پیاده سازی کنید.

راه حل:

۱) ابتدا تعداد حرکات لازم را حساب میکنیم به این صورت که، تعداد حرکات برابر دو به توان تعداد دیسک ها منهای یک است

۲) اگر تعداد دیسک ها زوج بود میله مقصد و کمکی را در مرحله ۳ جا به جا میکنیم (یعنی هرجا مقصد بود مینویسیم کمکی و  
(بالعکس)

۲) از  $1 = \alpha$  تا تعداد حرکات لازم و کامل شدن برج:

الف) اگر  $1 = \alpha$  دیسکی که مجاز است را بین مبدا و مقصد جا به جا میکنیم

ب) اگر  $2 = \alpha$  دیسکی که مجاز است را بین مبدا و کمکی (میله وسط) جا به جا میکنیم

ج) اگر  $0 = \alpha$  دیسکی که مجاز است را بین کمکی و مقصد جا به جا میکنیم

منظور از حرکت مجاز به هم نخوردن شرط اینکه دیسک بزرگتر نباید روی دیسک کوچکتر باشد است.

و به دلیل شرط اولیه که با تعداد دیسک ها تعین کردیم پیچیدگی زمانی مساله  $O(2^n)$  است.

## نمونه سوالات بخش ۱,۲,۳

۱,۲,۳,۴ الگوریتمی ارایه دهید که یال های برشی یک گراف را به دست آورد و مرتبه زمانی الگوریتم خود را نیز به دست آورید.

راه حل:

برای پیدا کردن یال های برشی ابتدا سعی میکنیم راس های برشی را پیدا کنیم. از یکی از راس ها شروع کرده و DFS میزنیم. دو مقدار زیر را نگه داری میکنیم، اولی عدد راس دیده شده هنگامی که توسط DFS دیده میشود ( $num(v)$ ) و دومین عدد هم کمترین مقدار اولین عدد است که با زیر درخت راس دیده شده توسط DFS است ( $low(v)$ ). برای اولین با که یک راس دیده میشود. مقدار  $low(v) = num(v)$  خواهد بود و واضح است که مقدار  $low(v) < num(v)$  نمیشود چون تمام رئوس زیر درخت  $v$  دارای فرزند  $v$  وقتی الگوریتم DFS بررسی راس  $v$  را تمام میکند، آنگاه  $low(v) \geq num(v)$  باشد. این یعنی آن حالت خاص نیز پیش نمیاید. اگر این حال برای راس  $v$  پیش بیاید یعنی  $low(v) < num(v)$  است. اگر حذف شود فرزندش که  $v$  میباشد دیگر هیچ مسیری به پدران  $v$  ندارد و گراف ناهمبند میشود مگر حالتی که آن ریشه درخت باشد. شرط اینکه راس برشی باشد این است که حداقل دو فرزند باشد. فقط مساله ای که میماند این است که  $low(v) = min[low(u), num(v)]$  را تعیین کنیم. این عدد را باید در بین DFS زدن تعیین کرد. به این صورت که اگر در راس  $v$  باشیم و DFS بررسی یک فرزند آن مانند  $v$  را تمام کند آنگاه  $low(v) = min[low(u), num(v)]$  میشود و حالت دیگر نیز این است که راس  $v$  را قبل از تمام کنند  $v$  را باشیم، در اینصورت  $low(v) = min[low(u), low(v)]$  میشود. چک کردن برشی بودن یک راس بعد از تمام شدن DFS روی یک فرزند و قبل از بروز رسانی  $low(v)$  پدر صورت میگیرد. در صورتی که به ازای فرزندی مانند  $v$  فهمیدیم ( $low(v) \geq num(v)$ ) است؛ نتیجه میگیریم که راس  $v$  برشی است و یال  $v$  یال برشی. هزینه زمانی این الگوریتم هم  $O(E + V)$  است.

---

۱,۲,۳,۴ الگوریتمی با استفاده از DFS ارایه دهید که حداقل یال هایی که نیاز است به گراف اضافه شود که قویا همبند شود را پیدا کند و هزینه زمانی را نیز بدست آورید.

راه حل:

ابتدا دو بار روی گراف و گراف transpose آن DFS میزنیم تا مولفه های قویا همبند را پیدا کنیم. حال یک گراف جدید ایجاد میکنیم که رئوس آن مولفه های قویا همبند هستند و اگر بین دو مولفه قویا همبند یال وجود داشته باشد، در گراف جدید بین آن دو، یال جهت دار میگذاریم. به وضوح گراف جدید یک DAG است چرا که اگر دور داشت به این معنی بود که رئوس روی دور همگی به یکدیگر راه داشتند و از ابتدا مولفه های قویا همبند جداگانه نبودند و باید همگی یک مولفه میشندند و نه چند مولفه. در این DAG تعداد رئوسی که یال ورودی ندارند را میشماریم و آنرا  $c_1$  مینامیم. سپس تعداد رئوسی که یال خروجی ندارند را

میشماریم و آنرا  $c_2$  مینامیم. اگر  $DAG$  فقط یک راس داشته باشد، لازم نیست بالی اضافه کنیم در غیر این صورت باید به تعداد  $\max(c_1, c_2)$  یال اضافه شود تا گراف قویا همبند شود.

---

۳,۸,۲,۱ مجموعه ای از عبارات منطقی داریم که هر عبارت منطقی از  $or$  دو متغیر میباشد که هر کدام از این دو متغیر میتواند به شکل نقیض شده باشد. الگوریتم بهینه ای ارایه دهید که مشخص کند آیا میتوان متغیر های یک مجموعه عبارات را به گونه ای مقدار دهی کرد که تمام عبارات مقدار  $true$  پیدا کنند. برای مثال:

$$E1 = A + B$$

$$E2 = \sim A + C$$

$$E3 = \sim C + \sim B$$

اگر مقدار  $A$  برابر  $true$  و مقدار  $B$  برابر  $false$  و مقدار  $C$  برابر  $true$  باشد آنگاه تمامی عبارت های  $E1, E2, E3$  مقدار  $true$  خواهند داشت.

راه حل:

مساله را با گراف به این صورت که به ازای هر متغیر  $X$  در مساله دو راس  $\sim X$ ،  $X$  اضافه میکنیم و به ازای هر عبارت مانند  $y + x$  یال های  $(y, \sim x)$  و  $(\sim y, \sim x)$  را نیز به گراف اضافه میکنیم، مدل سازی میکنیم. وجود یال  $(u, v)$  در گراف بیان میکند که اگر  $U$  درست باشد آنگاه  $V$  هم درست است. حال روی این گراف دو بار  $DFS$  میزنیم و مولفه های قویا همبند را پیدا میکنیم. سپس به ازای هر مولفه قویا همبند یک راس میگذاریم و اگر بین دو راس از دو مولفه یال وجود دارد با جهت درست دو راس را با یال به هم وصل میکنیم اما نمیتوان اطمینان حاصل کرد که دور بوجود نماید چون اگر دور داشته باشیم، اجتماع این دو مولفه همبند خود که مولفه همبند بوجود میاورد. سپس به ازای هر متغیر شماره مولفه آن را نیز ذخیره میکنیم. در آخر هم به ازای هر متغیر  $X$  چک میکنیم شماره مولفه  $X$  با  $\sim X$  مساوی نباشد. اگر به ازای همه رئوس این قائد برقار بود میتوان گفت امکان حل این مساله وجود دارد. حالا برای اینکه بفهمیم مقدار هر متغیر چقدر باشد به این شکل عمل میکنیم که آخرین گرافی را که از مولفه های قویا هم بند ساختیم یک  $DAG$  است. حالا روی آن **Topological Sort** میزنیم. اگر مولفه متغیر  $X$  قبل از متغیر مولفه  $\sim X$  باید، آنگاه  $X$  را  $false$  میگذاریم و در غیر این صورت  $true$ . البته توجه کنید که اگر گراف ناهمبند باشد و مسیری از مولفه شامل  $X$  به مولفه شامل  $\sim X$  نباشد یا بالعکس، در این صورت متغیر  $X$  میتواند هر مقداری داشته باشد.

---

۴,۸,۲,۱ الگوریتمی برای پیدا کردن مولفه های قویاً همبند گراف های جهت دار ارائه دهید.

راه حل:

یک گراف جهت دار قویاً همبند است اگر بین هر چفت از راس های آن مسیری وجود داشته باشد. به بزرگترین زیر گراف ممکن که قویاً همیند باشد مولفه ای قوی همبندی  $SCC$  آن گراف می گوییم. با استفاده از الگوریتم موسوم به **kosaraju** که از **DFS** استفاده می کند می توانیم تمام مولفه های همبندی یک گراف را با هزینه ای زمانی  $O(v+e)$  بدست آوریم.

الگوریتم به این ترتیب اجرا می شود:

(۱) پشته ای خالی ای به نام  $S$  می سازیم و پیمایش **DFS** را روی گراف اجرا می کنیم. در پیمایش **DFS** پس از صدا کردن بازگشتی **DFS** بر روی راس های مجاور یک راس، آن راس را در پشته **push** می کنیم

(۲) تمام یال ها را برعکس می کنیم تا گراف ترانهاده بدست آید.

(۳) تا وقتی که پشته خالی نشده است تک تک راس ها را از پشته **pop** می کنیم، راس **pop** شده را  $v$  می نامیم، و بر روی  $v$  الگوریتم **DFS** را اجرا می کنیم. **DFS** حاصل راس های مولفه ای همبندی راس  $v$  را می دهد.

بررسی بیچیدگی این الگوریتم: الگوریتم **DFS** را صدا می زند، سپس گراف را ترانهاده می کند و دوباره **DFS** را صدا می زند، هزینه ای  $O(v+e)$  را دارد، همچنین ترانهاده کردن گراف نیز همین هزینه را دارد، برای ترانهاده کردن گراف کافی است تمامی لیست های مجاورت را بیماییم. از نظر **order** هزینه ای زمانی این الگوریتم بهینه ترین است ولی الگوریتم های دیگری هم با همین **order** وجود دارند ولی تنها از یک **DFS** استفاده می کنند، مانند الگوریتم **.tarjan**.

---

۵,۸,۲,۱ فرض کنید  $T$  درختی است که حداقل دو راس دارد. کمترین و بیشترین تعداد مفصل هایی که  $T$  دارد را بدست آورید و بگویید در هر حالت چند مولفه دوسو همبند دارد.

راه حل:

الف)  $T$  میتواند حداقل یک و حداقل  $n-1$  مفصل داشته باشد. اگر  $T$  شامل شامل یک راس با درجه  $n-1$  باشد این راس تنها مفصل است. اگر  $T$  مسیری با  $n$  راس و  $n-1$  یال باشد، آنگاه  $n-2$  راس درجه ۲ همگی مفصل اند.

ب) در همه حالت ها درختی با  $n$  راس دارای  $n-1$  مولفه دوسو همبند است. هر یال یک مولفه دوسو همبند است.

---

۶,۸,۲,۱ یک ماتریس ۲ بعدی داریم، تعداد مولفه های همبندی گراف متناظر با آن را بدست آورید.

راه حل:

به مجموعه  $A$  هایی که کنار هم قرار گرفته اند یک مولفه  $i$  همبندی می گوییم. در نظر گرفتن این نکته ضروری است که هر خانه  $i$  ماتریس می تواند با  $8$  همسایه اش همبند باشد. با استفاده از ساختمان داده  $i$  مجموعه های مجزا به حل سوال می پردازیم.

الگوریتم ما به این شیوه عمل می کند:

(۱) متغیر  $result$  که تعداد مولفه های همبندی است را برابر  $0$  در نظر بگیر

(۲) تمام درایه های ماتریس را پیمایش کن

(۳) اگر مقدار هر درایه  $1$  بود، هر  $8$  همسایه اش را بررسی کن، اگر همسایه اش نیز یک بود آن ها را متعدد کن و همسایه های آن ها را نیز بررسی کن

(۴) حال آرایه ای به سایز ( $\text{طول} * \text{عرض}$ ) ماتریس بساز که فراوانی هر مجموعه را ذخیره کند

(۵) حال دوباره درایه های ماتریس را پیمایش کن

(۶) اگر مقدارایه  $1$  بود، مجموعه  $i$  متناظرش را بیاب

(۷) اگر فراوانی مجموعه در آرایه  $i$  بالا  $0$  بود،  $result$  را یک واحد افزایش بده

---

۷.۸.۲.۱ الگوریتمی برای پیدا کردن مولفه های قویاً همبند گراف های جهت دار ارائه دهید. به طوری که تنها مجاز به یک بار استفاده کردن از **DFS** باشد.

راه حل:

یک گراف جهت دار قویاً همبند است اگر بین هر جفت از راس های آن مسیری وجود داشته باشد. به بزرگترین زیر گراف ممکن که قویاً همیند باشد مولفه  $i$  قوی همبندی **SCC** آن گراف می گوییم. با استفاده از الگوریتمی موسوم به **tarjan** که از **DFS** استفاده می کند می توانیم تمام مولفه های همبندی یک گراف را با هزینه  $O(v+e)$  بدست آوریم.

الگوریتم به این ترتیب اجرا می شود:

(۱) جست و جوی **DFS** یک درخت یا جنگل **DFS** نتیجه می دهد.

(۲) مولفه های قویاً همبند زیر درخت های درخت **DFS** را تشکیل می دهند.

(۳) اگر ما بتوانیم **head** این زیر درخت را بیابیم، می توانیم تمام راس های این زیر درخت به علاوه **head** را ذخیره کنیم که در واقع یک مولفه  $i$  قویاً همبند را به ما می دهد.

(۴) هیچ یال بازگشتی ای از یک مولفه به مولفه  $i$  دیگر وجود ندارد.

برای پیدا کردن **head** مولفه ها به محاسبه کردن آرایه های **desc** و **low[u]** می پردازیم. **low[u]** را نمایش میدهد که زودتر از بقیه ای راس ها مشاهده شده است که توسط زیر درختی با ریشه **u** قابل دسترسی است. یک راس **u** همان **head** است اگر  $\text{disc}[u] = \text{low}[u]$  باشد.

مقدار **disc** در واقع نشان دهنده زمانی است که برای اولین بار یک راس را در حین پیمایش **DFS** مشاهده می کنیم. مقدار **low** برای هر راس نشان دهنده بزرگترین جد قابل دسترس در زیر درخت آن راس است. برای هر راس **u** وقتی **DFS** شروع می شود، **low** همان مقدار اولین **disc** اش است. بعداً در حین اجرای **DFS** بر روی تک تک فرزندان مقدار **low** راس **u** در دو حالت تغییر می کند.

$$1) \text{ low}[u] = \min(\text{low}[u], \text{low}[v])$$

$$2) \text{ low}[u] = \min(\text{low}[u], \text{disc}[v])$$

پیچیدگی این الگوریتم همان پیچیدگی **DFS** است.

تکه کد زیر الگوریتم را پیاده سازی کرده است.

```
// A C++ program to find strongly connected components in a given
// directed graph using Tarjan's algorithm (single DFS)
#include<iostream>
#include <list>
#include <stack>
#define NIL -1
using namespace std;

// A class that represents an directed graph
class Graph
{
    int V; // No. of vertices
    list<int> *adj; // A dynamic array of adjacency lists

    // A Recursive DFS based function used by SCC()
    void SCCUtil(int u, int disc[], int low[],
                stack<int> *st, bool stackMember[]);
public:
    Graph(int V); // Constructor
    void addEdge(int v, int w); // function to add an edge to graph
    void SCC(); // prints strongly connected components
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}
```

```

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}

// A recursive function that finds and prints strongly connected
// components using DFS traversal
// u --> The vertex to be visited next
// disc[] --> Stores discovery times of visited vertices
// low[] --> earliest visited vertex (the vertex with minimum
//           discovery time) that can be reached from subtree
//           rooted with current vertex
// *st --> To store all the connected ancestors (could be part
//           of SCC)
// stackMember[] --> bit/index array for faster check whether
//           a node is in stack
void Graph::SCCUtil(int u, int disc[], int low[], stack<int> *st,
                    bool stackMember[])
{
    // A static variable is used for simplicity, we can avoid use
    // of static variable by passing a pointer.
    static int time = 0;

    // Initialize discovery time and low value
    disc[u] = low[u] = ++time;
    st->push(u);
    stackMember[u] = true;

    // Go through all vertices adjacent to this
    list<int>::iterator i;
    for (i = adj[u].begin(); i != adj[u].end(); ++i)
    {
        int v = *i; // v is current adjacent of 'u'

        // If v is not visited yet, then recur for it
        if (disc[v] == -1)
        {
            SCCUtil(v, disc, low, st, stackMember);

            // Check if the subtree rooted with 'v' has a
            // connection to one of the ancestors of 'u'
            // Case 1 (per above discussion on Disc and Low value)
            low[u] = min(low[u], low[v]);
        }
    }

    // Update low value of 'u' only if 'v' is still in stack
    // (i.e. it's a back edge, not cross edge).
    // Case 2 (per above discussion on Disc and Low value)
}

```

```

        else if (stackMember[v] == true)
            low[u] = min(low[u], disc[v]);
    }

    // head node found, pop the stack and print an SCC
    int w = 0; // To store stack extracted vertices
    if (low[u] == disc[u])
    {
        while (st->top() != u)
        {
            w = (int) st->top();
            cout << w << " ";
            stackMember[w] = false;
            st->pop();
        }
        w = (int) st->top();
        cout << w << "\n";
        stackMember[w] = false;
        st->pop();
    }
}

// The function to do DFS traversal. It uses SCCUtil()
void Graph::SCC()
{
    int *disc = new int[V];
    int *low = new int[V];
    bool *stackMember = new bool[V];
    stack<int> *st = new stack<int>();

    // Initialize disc and low, and stackMember arrays
    for (int i = 0; i < V; i++)
    {
        disc[i] = NIL;
        low[i] = NIL;
        stackMember[i] = false;
    }

    // Call the recursive helper function to find strongly
    // connected components in DFS tree with vertex 'i'
    for (int i = 0; i < V; i++)
        if (disc[i] == NIL)
            SCCUtil(i, disc, low, st, stackMember);
}

```

---

۸.۸.۲.۱ فرض کنید  $C'$  مولفه قویا همبند متمایز گراف جهت دار  $G$  باشند. فرض کنید یال  $(u, v)$  وجود دارد به طوری که  $u$  عضو  $C$  و  $v$  عضو  $C'$  باشد. آنگاه  $f(c) > f(C')$

راه حل:

با توجه به اینکه مولفه همبند قوی،  $C$  یا  $C'$  در جریان DFS راس پیدا شده اول داشتند دو حالت را در نظر میگیریم.  
 اگر  $d(C') > d(c)$ ، اولین راس پیدا شده در  $C$  باشد در زمان  $x.d$  تمام راس ها در  $C$  و  $C'$  سفید هستند. در آن زمان  $G$  حاوی یک مسیر از  $x$  به هر راس در  $C$  است که فقط متشكل از راس های سفید هستند. از آنجا که یال  $(u, v)$  برای هر راس  $w$  عضو  $C'$  یک مسیر در  $G$  در زمان  $d(x)$  از  $x$  به  $w$  فقط متشكل از راس های سفید وجود دارد. بنا به قضیه مسیر سفید تمام راس ها در  $C'$  بچه های  $x$  در درخت DFS میشوند از آنجایی که  $x$  دیرترین زمان خانمه هر یک از بچه هایش را دارد از این رو  $x.f = f(c) > f(C')$

اگر به جای آن داشته باشیم  $d(c) > d(C')$ . فرض کنید  $y$  اولین راس پیدا شده در  $C'$  باشد. در زمان  $y.d$  تمام راس ها در  $C'$  سفید هستند و  $G$  حاوی یک مسیر از  $y$  به هر راس در  $C'$  است که فقط از راس های سفید تشکیل شده است. با به قضیه مسیر سفید تمام راس ها در  $C'$  بچه های  $y$  در درخت DFS میشوند بنابراین  $f(C') = f(y.f) = y.d$ . در زمان  $y.d$  تمام راس ها در  $C$  سفید هستند. از آنجایی که یال  $(u, v)$  از  $C$  به  $C'$  وجود دارد نتیجه میکیریم که یک مسیر از  $C'$  به  $y$  نمیتواند وجود داشته باشد. در اینجا هیچ راس در  $C$  از  $y$  قابل دسترسی نیست بنابراین در زمان  $y.f$  تمام راس ها در  $C$  همچنان سفید هستند پس برای هر راس  $w$  عضو  $C$  داریم  $f(c) > f(C')$  که نتیجه میدهد  $w.f > y.f$

## نمونه سوالات بخش ۸,۱

۱,۸,۱ فرض کنید گراف  $G$  داده شده است به صورتی که وزن هر یال یا یک است یا  $m$ . الگوریتمی با مرتبه زمانی  $O(V+E)$  بنویسید که کوتاه ترین مسیر از یک راس  $s$  به راس  $t$  در گراف ما را مشخص کند.

راه حل:

از گره  $s$  جستجوی سطح اول میزنیم. دو صفحه برای گره های اضافه شده با فاصله یک و دیگری برای گره های اضافه شده با فاصله  $M$  در نظر میگیریم. برای انتخاب گره بعدی در جستجوی سطح اول با توجه به فاصله های ثبت شده در ابتدای دو صفحه، گره بعدی را انتخاب میکنیم. فاصله نهایی گره های خارج شده از صفحه دوم را در صورتی که قبل از صفحه خارج نشده بودند در نظر میگیریم و اگر قبل از خارج شدن آنرا با استفاده از صفحه یک پیموده بودیم دیگر آن گره را در نظر نمیگیریم.

---

۲,۸,۱ فرض کنید گراف  $G$  داده شده است به صورتی که وزن یال های آن عددی بین صفر و  $N$  است. الگوریتمی ارایه دهید که کوتاه ترین مسیر از یک راس  $s$  به  $t$  را با زمان  $O(E + V^*N)$  و حافظه  $O(V + N)$  مشخص کند.

راه حل:

یک ارایه از صفحه های داریم، از گره  $s$  جستجوی سطح اول میزنیم. گره های مجاور را با توجه به فاصله شان در صفحه مربوطه به خود اضافه میکنیم. برای مثال اگر گره فعلی فاصله  $a$  از  $s$  داشته باشد و یال به گره مجاورش وزن  $b$  داشت، گره مجاور را در صفحه  $a+b$  اضافه میکنیم. توجه کنید که صفحه ها در خود فاصله ها را نیز نگه میدارند پس لازم نیست دفعه ای بعدی صفحه های با فاصله کمتر از  $a$  را بررسی کنیم. فاصله های نهایی گره های خارج شده از صفحه مقدار دهی میکنیم و اگر گره ای را قبل از خارج شدن پیموده بودیم، دیگر در نظر نمیگیریم.

---

۳,۸,۱ گراف دو بخشی گرافی است که بتوان رئوس آنرا به دو دسته افراز کرد به نحوی که در بین رئوس دسته ها یالی وجود نداشته باشد. الگوریتمی ارایه دهید که با گرفتن اطلاعات یک گراف به ما بگوید دو بخشی است یا نه.

راه حل:

با یک بار **BFS** زدن روی گره ها حرکت میکنیم و گره ها به دو بخش تقسیم میکنیم. گره نخست را در یکی از بخش ها قرار میدهیم و به ترتیب گره هایی را که میبینیم در بخش مخالف قرار میدهیم. در حین حرکت اگر به تناقضی برخوردیم یعنی گراف دو بخشی نیست و در غیر این صورت گراف دو بخشی است زیرا به بخش مجزا افزایش شده است.

---

۴.۸.۱ در یک درخت، قطر را چگونه میتوان پیدا کرد؟

راه حل:

از یکی از گره ها شروع میکنیم و **BFS** میزنیم و دور ترین گره به این گره را بدست میاوریم. از گره جدید دوباره **BFS** میزنیم و دور ترین گره را به این گره نیز به دست میاوریم. مسیر به دست آمده بین این دو گره قطر درخت است.

---

۵.۸.۱ یک نینجا در ته چاهی با دو دیوار که هر کدام از  $n$  بلوك آجری تشکیل شده است گرفتار شده. وینگی این چاه ها در این است که بعضی از بلوک های دیواره های این چاه سمی است. نینجا ما در ابتدا در پایین ترین نقطه این چاه است. او در هر مرحله میتواند یک بلوك بالا یا پایین ببرود یا به  $k$  بلوك بالاتر در دیوار مخالف بپردازد. با هر حرکتی که نینجا ما میکند سطح آب کف چاه یک بلوك بالاتر میاید. در این شرایط که به دیوار های سمی هم نباید بخورد الگوریتمی با گرفتن اطلاعات دیوار ها عدد  $n$  ارایه

دهید تا نینجا نجات پیدا کند.

راه حل:

بلوک ها را گره گراف در نظر میگیریم و هر گره به گره بالایی و پایینی و  $n$  تا بالاتر از خودش یال دارد. گره های سمی و یال هایشان را حذف میکنیم و از دو گره پایینی **BFS** میزنیم و در حین حرکت بررسی میکنیم که سطح گره از سطح فعلی پایین تر نباشد.

---

۶.۸.۱ در مسائل مربوط به پیدا کردن کوتاه ترین مسیر بین دو راس از یک گراف اغلب با تعداد بیشتر از یک مسیر مواجه هستیم، الگوریتمی با پیچیدگی زمانی خطی بدست آورید که تعداد تمام کوتاهترین مسیر های متمایز بین دو راس  $u$  و  $v$  را پیدا کند.

راه حل:

با تغییر دادن **BFS** الگوریتمی برای حل این مسئله می باییم. از راس **v** الگوریتم را شروع می کنیم در طی اجرای الگوریتم ما همواره دو متغیر اضافی برای هر راس ذخیره می کنیم، یکی **distance** که فاصله این راس تا راس **v** است و دیگری تعداد مسیر ها با طول **distance** از راس **v**. الگوریتم بر این اساس کار می کند: هر مسیر به طول **k** از **u** به **v** را می توان به مسیری به طول **k-1** از **u** به همسایه های **v** و مسیری به طول یک به راس **v** تقسیم کرد. بنابراین اگر ما تعداد کوتاه ترین مسیر ها به تمامی راس های میانی رو دنبال کنیم به تعداد تمام کوتاه ترین مسیر ها بین **u** و **v** میرسیم.

```
function NumShortestPaths(G,u,v)
```

```
Let Q be an empty queue
```

```
u.visited = true
```

```
u.distance = 0
```

```
u.numPaths = 1
```

```
add u to Q
```

```
while Q is not empty do
```

```
curr = Q.dequeue()
```

```
for each neighbor t of curr do
```

```
if t.visited == false then
```

```
t.visited = true
```

```
t.distance = curr.distance + 1
```

```
t.numPaths = curr.numPaths
```

```
add t to Q
```

```
else
```

```
if t.distance == curr.distance + 1 then
```

```
t.numPaths += curr.numPaths
```

```
end if
```

```
end for
```

```
end while
```

```
return v.numPaths
```

۷,۸,۱ فرض کنید بخواهیم یک گراف غیرجهتدار و همبند را با دو رنگ سیاه و سفید رنگ آمیزی کنیم به طوری که راس های مجاور همنگ نباشند. اثبات کنید اگر گراف دوری به طول فرد داشته باشد همچین رنگ آمیزی ای وجود ندارد.

راه حل:

برهان خلف. فرض می کنیم این کار در حالی که گراف دور دارد شدنی باشد. فرض میکنیم  $v_1$  سفید،  $v_2$  سیاه،  $v_3$  سفید و به همین ترتیب راس ها با ایندکس فرد سفید و راس های با ایندکس زوج سفید باشند. از انجایی که اندیس راس آخر فرد است پس سفید است و این راس با راس ۱ مجاور است، بنابراین هر دو همنگ خواهند بود پس برهان خلف باطل می باشد و نمی توانیم این رنگ آمیزی را داشته باشیم.

الگوریتمی با پیچیدگی  $O(v+e)$  بیابید که این رنگ آمیزی را با در نظر گرفتن این نکته که گراف دور فرد نداشته باشد، بیابد.

فرض کنید راس  $v$  سفید است، روی این راس الگوریتم BFS را اجرای می کنیم و تمام راس های level اول باید سیاه باشند چون راس اول سفید بود، سپس تمام راس های level دوم باید سیاه باشند و به همین ترتیب شما می توانید با استفاده از پارامتر فاصله ای هر راس با راس  $v$ ، تمام راس های گراف را رنگ آمیزی می کنیم، پیچیدگی این الگوریتم همان پیچیدگی BFS است یعنی همان  $O(v+e)$ .

---

۸,۸,۱ در یک کشور ناکجآباد  $n$  شهر وجود دارد که توسط  $n-1$  جاده به هم متصل شده اند؛ در این کشور از هر شهر میتوان به شهر دیگر رفت. بر اثر سیل تعدادی از جاده ها خراب شده است و نیاز به تعمیر دارد. برای تعمیر هر جاده اداره راه شهر های دو سمت این جاده باید به صورت همزمان مشغول به تعمیر جاده شوند. تعمیر هر جاده دقیقاً یک روز زمان میبرد. همچنین اداره راه هر شهر در یک روز تنها میتواند روی یک جاده کار کند. تعداد کمترین روز هایی که برای تعمیر همه جاده های تخریب شده نیاز است چقدر است؟ یک برنامه زمانی پیشنهاد دهید که نشان در هر روز کدام جاده ها باید تعمیر شوند.

راه حل:

از شهری که بیشترین جاده خراب  $k$  را دارد BFS و هر جاده تخریب شده را با یک رنگ از  $k$  رنگ، رنگ آمیزی میکنیم به طوری که جاده های مجاور همنگ نباشند. جاده های همنگ را در یک روز تعمیر میکنیم.

---

۹,۸,۱ دانشجویان کامپیوتر همیشه بعد از کلاس ساختمان داده با هم این بازی را انجام میدهند. ابتدا در یک صف به ترتیب قد قرار میگیرند (هیچ دو نفر قد یکسان ندارند). سپس در هر مرحله از بازی هر کسی با احتمالی خاص کوتاه ترین فرد را از بازی حذف میکند اما خودش را نمیتواند حذف کند. شما باید با دانستن مقدار احتمال برای دانشجویان بگویید که پس از حداقل  $k$  مرحله چه زیر مجموعه‌ای از دانشجویانی که در ابتدا شروع به بازی کرده اند ممکن است باقی بمانند. (دقت کنید که احتمال دو نفر متفاوت لزوماً متفاوت و یکسان نیست و این احتمال میتواند هر مقداری میان بازه بسته صفر و یک باشد)

راه حل:

هر حالتی که ممکن است بوجود بیاید شرایط زیر را دارد:

$M \rightarrow M$

$Z \rightarrow Z$

تعداد بزرگتر مساوی صفر مرده  $\rightarrow M^*$

تعداد بزرگتر مساوی صفر زنده  $\rightarrow Z^*$

پس هر حالت را میتوان با دو زنده ابتداییش به صورت یکتا مشخص کرد و آنرا با  $(A, B)$  نشان میدهیم.

از هر حالت به چهار حالت میتوان برویم:

$(A, B) \rightarrow (A, B)$

به شرط آنکه احتمال هیچ کس  $100\%$  نباشد

$(A, B) \rightarrow (A, B+1)$

به شرط آنکه  $A$  صفر درصد و از  $B$  به بعد کسی  $100\%$  نباشد

$(A, B) \rightarrow (B, B+1)$

به شرط آنکه  $A$   $100\%$  نباشد و از  $B$  به بعد همه صفر درصد نباشد

$(A, B) \rightarrow (B+1, B+2)$

به شرط آنکه  $A$  صفر درصد نباشد و از  $B$  به بعد همه صفر درصد نیاشند

چون تعداد حالت‌ها محدود است میتوان از حالت آغازین به عمق  $k$  BFS زد.

۱۰،۸،۱ معین میخواهد شهر را بگردد. برای اینکار او میخواهد حداقل یکبار همه خیابان‌های شهر را ببیند. شما باید به او کمک کنید تا هرچه سریعتر بتواند این کار را انجام دهد. او میداند که طول همه خیابان‌های شهر عددی بین یک تا  $W$  است و به همه تقاطع‌ها دقیقاً ۴ خیابان متصل است به جز دو تقاطع که سه خیابان دارند. او میخواهد از جایی شروع به حرکت کند و همه خیابان‌های را ببیند و به نقطه شروع بازگردد و این کار را در سریع ترین زمان ممکن انجام دهد.

راه حل:

مساله مدل میشود به گرافی همبند با دو راس درجه فرد و تعدادی راس با درجه زوج. انتخاب دنباله‌ای از یال‌ها به طوری که هر دو یال مجاور در دنباله با هم در یک راس اشتراک داشته باشند و همه یال‌ها در دنباله حداقل یکبار آمده باشد.

مسیری که انتخاب میشود درواقع دوری اویلری در گراف است که از بعضی از یال‌ها بیش از یکبار استفاده شده است. اگر یال‌هایی که یکبار استفاده شده اند را کنار بگذاریم باقی یال‌ها تبدیل میشوند به مسیری از یکی از دو راس فرد به دیگری. پس مساله تبدیل میشود به پیداکردن کوتاه‌ترین مسیر بین این دو راس و اضافه کردن آن به گراف تا تبدیل به گراف اویلری شود.

برای اینکار آرایه‌ای از صفحه هارا در نظر میگیریم از یکی از گره‌ها BFS میزنیم. گره‌های مجاور را با توجه به فاصله شان در صفحه مربوط اضافه میکنیم. برای مثال اگر گره فعلی فاصله  $a$  از راس ابتدایی داشت و یال به گره مجاورش وزن  $b$  را داشت گره مجاور را در صفحه  $a+b$  اضافه میکنیم. به این ترتیب هر گره در صفحه با شماره فاصله اش اضافه شده است. فاصله‌ها را در هنگام خارج شدن از صفحه نهایی میکنیم و اگر گره‌ای قبلاً پیموده شده بود دیگر آنرا در نظر نمیگیریم و کوتاه‌ترین فاصله بین دو راس با درجه فرد به دست می‌آید.

---

۱۱،۸،۱ برخی از کاربردهای جست و جوی اول-سطح را نام ببرید.

راه حل:

مسئله‌ی کوتاه‌ترین مسیر و درخت پوشای کمینه برای گراف‌های بدون وزن: در گراف‌های بدون وزن کوتاه‌ترین مسیر، مسیری است که شامل کمترین یال‌ها باشد، با BFS زدن روی یک راس همیشه با کمترین تعداد یال‌ها به مقصد می‌رسیم، همچین در گراف‌های بدون وزن هر درختی پوشایی، درخت پوشای کمینه است بنابراین با استفاده از BFS می‌توانیم همچین درختی ایجاد کنیم.

سیستم‌های مسیر یابی: BFS تمام موقعیت‌های همسایه را به ما می‌دهد.

زیاله روبی در زبان‌های برنامه نویسی مانند جاوا: با استفاده از BFS حافظه‌ای که دیگر مورد استفاده ای برنامه نیست شناسایی میشود و حذف میشود.

پیدا کردن دور در گراف های غیر جهت دار: در گراف های غیر جهت دار می توان با این الگوریتم دور را شناسایی کرد در حالی که در گراف های جهت دار این کار با BFS امکان پذیر نیست.

تشخیص دو بخشی بودن یا نبودن گراف: با استفاده از BFS و رنگ آمیزی یکی در میان هر level این کار امکان پذیر است.

پیدا کردن مسیر بین دو راس: با استفاده از این الگوریتم می توان مسیر بین دو راس را در صورت وجود یافت.

پیدا کردن تمام راس های یک مولفه‌ی همبند: با استفاده از BFS می توان تمام راس های قابل دسترس از راس داده شده را یافت.

همچنین بسیاری از الگوریتم های معروف همچون الگوریتم پراایم برای یافتن درخت پوشای کمینه یا الگوریتم داکسترا برای یافتن کوتاه ترین مسیر با مبدأ معین از ساختاری مشابه BFS استفاده می کنند.

---

## ۱۲.۸.۱ چه زمانی بهتر است از الگوریتم های DFS یا BFS استفاده کنیم؟

راه حل:

اینکه برای حل مسئله از کدام روش جست و جو استفاده کنیم تا حدود زیادی به مسئله و ساختار درختی که می خواهیم جست و جویش کنیم و همچنین شرایط و ویژگی های آن بستگی دارد. اگر بدانیم پاسخ از ریشه‌ی زیاد فاصله‌ای ندارد، BFS بهتر خواهد بود. اگر درخت ما عمیق باشد و پاسخ ها مسئله کمیاب باشند DFS به احتمال زیاد زمان بسیار زیادی طول خواهد کشید، در عوض BFS سریع تر خواهد بود. اگر درخت ما دارای پهنه‌ی زیادی باشد و عریض باشد BFS نیاز به حافظه‌ی بسیار زیادی خواهد داشت، بنابراین نشدنی خواهد بود. اگر جواب در درخت ما زیاد رخدده و در عمق باشد BFS ممکن است نشدنی باشد، اگر درخت بسیار عمیق باشد ممکن است در جست و جوی DFS برای پیمودن عمق درخت محدودیتی قائل شوید و گرنه اینکار بسیار طولانی و غیر ممکن خواهد شد.

---

## ۱۳.۸.۱ وقتی از ماتریس مجاورت استفاده می کنیم، اکثر الگوریتم های گراف با پیچیدگی زمانی امگا $V^7$ کار می کنند، ولی چندین استثنای نیز وجود دارد. نشان دهید با داشتن ماتریس مجاورت گراف، تشخیص اینکه آیا گراف ما دارای یک "حرفه‌ی فراگیر" (راسی با درجه‌ی ورودی ۱ - ۷ و درجه‌ی خروجی ۰) است در زمان $O(V)$ انجام می شود.

راه حل:

ابتدا توجه به این دو نکته ضروری است اول اینکه در ماتریس مجاورت، عناصر ردیف  $A$ ، یال های خارج شده از راس  $A$  به بقیه راس ها را نشان میدهد. و عناصر ستون  $Z$ ام، یال های وارد شده به راس  $Z$ را نشان میدهد. و دوم اینکه از آنجایی که از "حفره فراگیر" به بقیه ای راس ها یالی خارج نمی شود بنابراین ماقسیموم ۱ همچین راسی وجود خواهد داشت.

ما به دنبال حفره ای فراگیر هستیم، راسی با درجه ای ورودی  $1 - 7$  و درجه ای خروجی  $0$ . کاری که ما باید انجام بدیم در واقع پیدا کردن  $k$  است، به طوری که ردیف  $k$  ام تماماً صفر و ستون  $k$  ام تماماً  $1$  به جز عنصر  $k$  ام که باید صفر باشد. از پیمایش ردیف اول شروع می کنیم، و به محض مشاهده ای اولین  $1$  متوقف میشویم. این کار در بدترین حالت  $O(V)$  طول میکشد. اگر به  $1$  بر نخوردیم و از انجایی که حفره ای فراگیر یکتا است پس یعنی این ردیف تنها کاندید احتمالی برای حفره ای فراگیر است. سپس بر روی اولین ستون در زمان  $(V)$  بررسی می کنیم که آیا تمام عناصر آن ها  $1$  میباشد یا نه در صورتی که اینگونه باشد راس اول "حفره ای فراگیر" است، در غیر اینصورت این گراف همچین راسی ندارد. الگوریتم زمانی پایان می پذیرد که ما به ردیفی با تمام عناصر صفر بر بخوریم و بررسی کنیم که آیا این راس "حفره ای فراگیر" هست یا خیر. بنابراین مطمئن هستیم که هزینه ای این الگوریتم  $O(V)$  باقی می ماند.

---

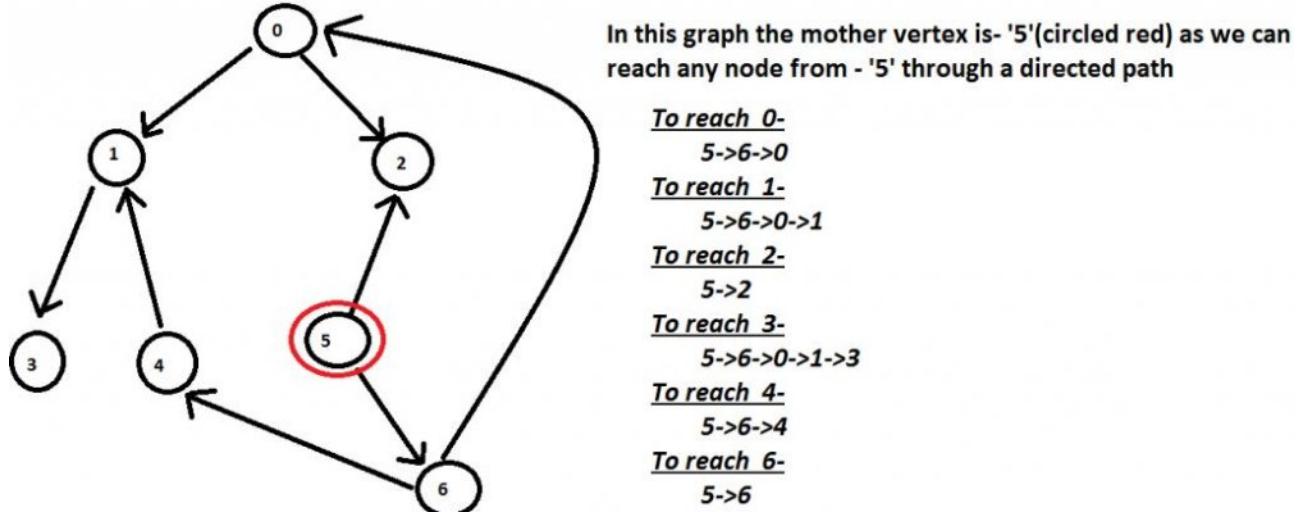
۱۴.۸. الگوریتم BFS را به گونه ای تغییر دهید که بتواند کوتاه ترین مسیر در گراف وزن دار را بدست بیاورد.

راه حل:

برای این منظور ابتدا گراف وزن دار را به گراف غیر وزن دار تبدیل می کنیم به طوری که به ازای هر یال با وزن  $x$  به تعداد  $1 - x$  راس بین این دو راس قرار می دهیم و در نتیجه  $1$  یال به  $x$  یال متوالی تبدیل میشود. حال اگر بر روی این گراف جدید الگوریتم BFS را اجرا کنیم، کوتاه ترین مسیر ها را به ما می دهد. از آنجایی که تعداد یال ها و راس های گراف جدید هم order با تعداد یال ها و راس های گراف اصلی است بنابراین این الگوریتم نیز با پیچیدگی  $O(v+e)$  کوتاه ترین مسیر ها را به ما می دهد.

## پرسش اول

الگوریتمی طراحی کنید که در یک گراف یک راس مادر را پیدا کند ( راس مادر راسی است که از آن به همه راس‌های مسیری ساده یا بدون دور وجود دارد)



اگر گراف ناهمبند باشد هیچ راس مادری وجود ندارد چراکه از هیچ راسی به همه راس‌های دیگر مسیر نداریم از طرفی برای گراف غیرجهتدار همبند نیز همه راس‌ها راس مادر هستند زیرا از هر راس به راس دیگر مسیری وجود دارد برای گراف‌های جهتدار الگوریتم مورد نظر را بدست می‌آوریم بدین منظور روی گراف الگوریتم جستجوی عمق اول را اجرا می‌کنیم و مشابه الگوریتم پیدا کردن اجزای قویا همبند ابتدا راسی با بیشترین زمان پایان را پیدا می‌کنیم این راس راسی است که به همه راس‌های دیگر مسیری دارد

## پرسش دوم

در یک گراف جهتدار الگوریتمی طراحی کنید که نشان دهد آیا گراف دور دارد یا خیر؟

با استفاده از تعاریف یال‌ها در گراف در صورتی که گراف یال برگشتی داشته باشد دور به وجود می‌آید برای بررسی این ویژگی روی گراف مورد نظر الگوریتم جستجوی عمق اول را اجرا می‌کنیم در هر مرحله در صورتی که در میان راس‌های انتخابی از راس‌های مجاور راس کنونی راسی وجود داشت که قبلاً بازدید شده است یال برگشتی داریم بنابراین گراف دارای دور می‌باشد که هزینه این الگوریتم برابر هزینه جستجو عمق اول است

## پرسش سوم

مدیریت راه‌های یک شهر تازه کشف شده قصد دارد برای ساختن راه‌های شهر خود برنامه ریزی کند این امر در حالی است که راه‌هایی در حال حاضر در شهر وجود دارند با توجه به این که همه راه‌ها یک طرفه اند به مدیریت کمک کنید تا با افزودن کمترین راه‌ها طوری عمل کند که همه مراکز شهر به هم راه داشته باشند؟

هر مرکز شهر را یک راس گراف در نظر می‌گیریم سوال در اصل یافتن حداقل یال‌های ممکن به یک گراف است به شرطی که گراف قویا همبند شود برای این منظور در ابتدا اجزای قویا همبند گراف را پیدا می‌کنیم برای این کار ابتدا روی گراف الگوریتم دی اف اس را اجرا می‌کنیم و راس‌های را به ترتیب نزولی بر اساس زمان پایان آن‌ها مرتب می‌کنیم سپس گراف را به ترانهاده آن تبدیل می‌کنیم و بار دیگر روی آن بر اساس ترتیب به دست آمده دی اف اس می‌زنیم چون گراف ترانهاده شده مسیرهای بین مولفه‌های قویا همبند بر عکس می‌شوند و با شروع دی اف اس از راس مادر در هر مرحله یک مولفه را بدست می‌آوریم

حال هر مولفه را یک سوپر نود در نظر می‌گیریم و در گراف ایجاد شده تعداد راس‌های با درجه ورودی صفر را پیدا می‌کنیم و در

مجموعه الف میریزیم سپس راس های با درجه خروجی صفر را پیدا کرده و در مجموعه ب قرار میدهیم این راس ها ایزوله هستند و تنها امکان وجود و یا خروج از آن ها وجود دارد بنابراین برای آن که به همه راس ها مسیر داشته باشیم باید این راس ها را از این حالت خارج کنیم بنابراین حداقل به ماکزیمم تعداد اعضای الف و ب یال نیاز داریم که تعداد حداقل راه هایی است که باید ساخته شود

### پرسش چهارم

برای ساخت منبع آب در یک نقطه بین روستاها به دنبال یافتن مکانی هستیم که بیشترین تعداد جاده هایی که کوتاه ترین مسافت را دارند از آن ها بگذرد به ما در پیدا کردن این مکان کمک کنید؟

از هر راس الگوریتم دایکسترا را اجرا میکنیم و درخت حاصل از اجرا را نگه میداریم. در عین حال برای هر راس آرایه ای مانند A نگه میداریم که در آن تعداد راس های موجود در زیر درخت آن راس در درخت ایجاد شده را یادداشت میکنیم. برای بدست آوردن تعداد راس های زیر درخت یک راس از روی آن DFS میزنیم. این عدد برابر مجموع راس های زیر درخت چپ به اضافه زیر درخت راست می باشد. در نهایت آرایه ای که در یک راس بیشترین مقدار را داشته باشد راس متناظر آن راس مورد نظر است.

### پرسش پنجم

الگوریتم prim را برای گراف  $G = (V, E)$  که با ماتریس مجاورت بیان شده طوری بیابیم که پیچیدگی زمانی آن  $O(V^8)$  باشد. با شروع از یک راس دلخواه بنابر الگوریتم باید یالی را بیابیم که safe باشد که این یال یالی با کمترین وزن از راس فعلی به دیگر راس هایی است که در مجموعه ای فعلی نیستند بنابراین در هر مرحله باید  $V$  راس دیگر را ملاقات کنیم و از آنجا که این عمل را (عمل ریلکس کردن) به اندازه تعداد راس ها انجام میدهیم عملیات از مرتبه زمانی  $O(V^8)$  خواهد بود.

### پرسش ششم

ثابت کنید که در درخت DFS، راس  $v$  از نوادگان راس  $u$  است، اگر و تنها اگر در زمان  $d[u]$  که تابع DFS شروع به بررسی راس  $u$  میکند، مسیری از  $u$  به  $v$  وجود داشته باشد که تمام رئوس آن سفید باشند.

فرض کنید  $v$  نواده  $u$  باشد . اگر راسی در مسیر  $u$  به  $v$  باشد  $d[u] < d[v]$  بنابراین  $w$  سفید است. طرف دوم : با برهان خلف فرض میکنیم چنین مسیر سفیدی موجود باشد اما  $v$  نواده  $u$  نباشد فرض کنید بقیه رئوس این مسیر سفید نواده  $u$  باشند. اگر  $w$  پدر  $v$  در این مسیر باشد  $w$  نواده  $u$  است بنابراین داریم :

$$d[u] < d[w] < f[w] < f[u]$$

و چون  $w$  پدر  $u$  است بنابراین  $d[w] < d[v] < f[w]$  در نتیجه داریم :

$$d[u] < d[v] < f[v] < f[u]$$

پس  $v$  نواده  $u$  است.

### پرسش هفتم

نوید یک لیست از دوستانش را در یک فایل نگه داشته که هر یک از آن ها شامل اسم ایمیل و شماره تلفن آن ها است او سال ها است که این لیست را تکمیل میکند و ممکن است دو نفر در لیست او یکسان باشند که این در صورتی اتفاق می افتد که حداقل یکی از فیلد ها بین دو نفر

یکسان باشد به او کمک کنید تا به وسلیه گراف ها به این هدف برسد

هر یک از دوستان را یک راس گراف در نظر میگیریم دو راس را در صورتی به هم متصل میکنیم که یکی از ویژگی های آن دو یکسان باشد حال سوال به پیدا کردن مولفه های همبندی خلاصه میشود حال با اجرای یکبار عملیات جستجوی سطح اول با عمق اول هر مولفه را پیدا میکنیم و آن هارا یکسان سازی میکنیم

### پرسش هشتم

در یک گراف غیر جهت دار داده شده الگوریتمی را بیابید که حداقل تعداد یال لازم برای تبدیل گراف به گرافی با دور فرد را بیابید؟

بنابر قضیه میدانیم یک گراف دور فرد ندارد اگر و فقط اگر دو بخشی باشد. بنابراین ابتدا با یک بار رنگ آمیزی گراف با عملیات bfs بررسی میکنیم که میتوان گراف را با دو رنگ رنگ آمیزی کرد با خیر اگر اینطور نباشد گراف دو بخشی نیست و دور فرد داریم در غیر این صورت بررسی میکنیم که اگر گراف همه راس هایش از درجه صفر بود به سه یال اگر حداقل درجه رئوس یک بود دو یال و در غیر این صورت به یک یال نیاز خواهیم داشت.

### پرسش نهم

گراف وزن دار بدون جهت  $G$  و زیر گراف  $F$  از آن را در نظر بگیرید که یک جنگل است. الگوریتم بهینه ای ارانه دهید که کم هزینه ترین درخت پوشای  $G$  را طوری بباید که همه یال های  $F$  را در بر بگیرد.

در ابتدا از جنگل  $F$  شروع میکنیم و هر مولفه همبندی آن را با اجرای عملیات DFS بدست می آوریم سپس هر مولفه را یک راس در نظر گرفته و مشابه الگوریتم prim یا kruskal انقدر یال های safe را به مجموعه یال ها می افزاییم تا به درخت پوشای بهینه برسیم.

### پرسش دهم

فرض کنید در گراف  $G$  درخت پوشای بهینه را داریم. حال اگر وزن یکی از یال ها در این درخت را کم کنیم با کمترین هزینه درخت پوشای جدید را بیابید.

اگر بالی که وزن آن کم شده است یال  $(u,v)$  باشد آنگاه از راس  $u$  مسیری را به راس  $v$  پیدا میکنیم (در درخت پوشای  $G$ ) در این مسیر یال  $(u,v)$  را بیشترین وزن را در نظر میگیریم اگر این یال وزنی بیشتر از یال  $(u,v)$  آن را حذف کرده و  $(u,v)$  را جایگزین آن میکنیم در این صورت درخت همبند مانده و مینیمم میماند.

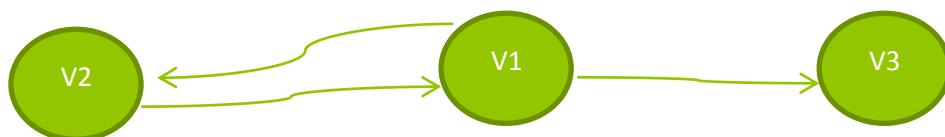
## نمونه سوالات مربوط به بخش ۸.۴

۸.۴.۱ پروفسور Deaver ادعا می کند که الگوریتم برای اجزای همبندی قوی می تواند با استفاده از گراف اولیه (به جای ترانهاده) در دومین جستجوی اول عمق و بررسی رؤوس به ترتیب افزایش زمان های خاتمه ساده تر شود. آیا ادعای پروفسور درست می باشد؟

راه حل:

نه، همواره درست نیست.

مثال نقض شکل زیر است. فرض کنید که DFS بزنیم از راس  $V_1$ ، سپس اگر بر اساس افزایش FINISHING TIME هاشون مرتب کنیم می شود:



Finishing Time:  $V_2, V_1, V_3$

حال برای دومین بار که از راس  $V_2$  ما DFS بزنیم، و به ترتیب افزایش Finish Time ها مرتب کنیم

Finish Time : $v_1, v_2, v_3$

در حقیقت دو تا SCCs در گراف داریم.  $\{V_1, V_2\}$ ,  $\{V_3\}$

۸.۴.۲ گراف جهت دار  $G(V, E)$  داده شده است. تبدیل  $T(G)(V, E)$  به صورت زیر به دست می آید. مجموعه رؤوس گراف  $G$  و  $T(G)$  یکسان هستند. از راس  $U$  به  $V$  در گراف تبدیل، یال جهت دار قرار می دهیم. اگر و تنها اگر مسیری از  $U$  به  $V$  در  $G$  وجود داشته باشد.

خوشه در یک گراف جهت دار مجموعه ای از رؤوس مانند  $U$  است. که به ازای هر دو راس  $t, S$  داخل  $U$  یال جهت داری از  $S$  به  $t$  یا از  $t$  به  $S$  و یا هر دو وجود داشته باشد. اندازه خوشه برابر تعداد رؤوس داخل آن است.

الگوریتمی برای یافتن بزرگترین خوشه ای  $T(G)$  ارائه دهید.

راه حل:

در این مسئله به دنبال بزرگترین زیر مجموعه از رؤوس هستیم. که بین هر دو راس آن مسیر وجود داشته باشد. اگر گراف DAG باشد. پاسخ طولانی ترین مسیر موجود در آن است.

در غیر این صورت با دو DFS مولفه های قوی همبندی گراف را پیدا می کنیم. و به ازای هر مولفه همبندی یک راس با برچسبی که برابر تعداد راس های آن مولفه است قرار می دهیم. سپس topological sort می زنیم، و از راس آخر شروع کرده (راسی که یال

خروجی ندارد) و یکی یکی به عقب می رویم. بزرگترین مسیر خارج شده از هر راس برابر است با ماکریمم بزرگترین مسیر های خارج شده از رئوسی که به آن ها يال دارد به علاوه ای بر چسب خودش به این ترتیب با  $(V + E)$  بزرگترین خوش به دست می آید.

#### 8.4.3 اگر یک يال به گراف اضافه کنیم. اجزای همبند قوی گراف چگونه تغییر می کند؟

راه حل:

دو حالت رخ می دهد. یا این که تغییر نمی کند. یا کاهش می یابد.

فرض کنید  $m$  گراف اصلی  $SCCs$

$m' =$  گراف جدید  $SCCs$

$m' \geq 1$  ،  $m' \leq m$  داریم:

#### 8.4.4 گراف جهت دار $G(V, E)$ "نیمه همبند" گفته می شود. اگر برای همه جفت رئوس $v, u \in V$

$V \rightarrow U$  یا  $U \rightarrow V$  داشته باشیم.

یک الگوریتم موثر برای تعیین این که آیا  $G$  نیمه همبند است یا خیر ارائه دهید. و زمان اجرای الگوریتم خود را هم محاسبه کنید.

راه حل :

۱: اجزای همبندی گراف را باید پیدا کرد

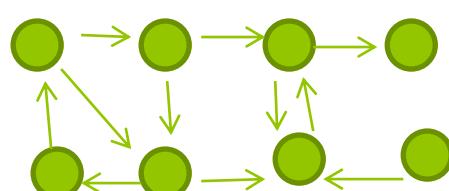
۲: حال این اجزای همبندی را مانند یک راس در نظر بگیرید. و کل گراف که الان یک DAG است را به عنوان یک گراف کلی جدید در نظر بگیرید.

۳: روی اجزای گراف توپولوژی سوت بزنید

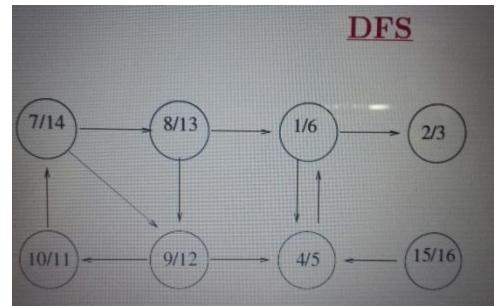
۴: بررسی کنید که يال های  $(V_1, V_2), (V_2, V_3), \dots, (V_{k-1}, V_k)$

وجود دارد در اجزای گراف اگراین راس ها یک حلقه به وجود آورند. پس گراف اصلی نیمه همبند است. در غیر این صورت خیر

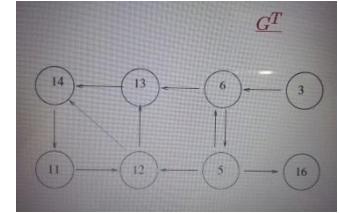
#### 8.4.5 نشان دهید روال **strongly\_connected\_components** چگونه بر روی شکل زیر کار می کند.



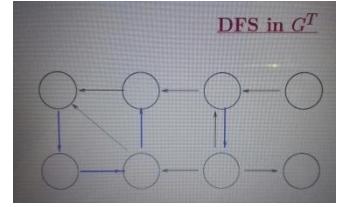
راه حل:



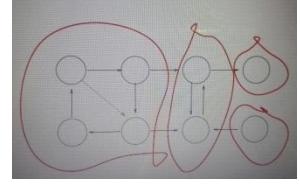
اول:



دوم:



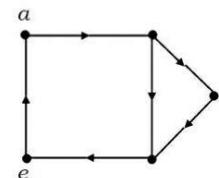
سوم:



چهارم:

---

#### 8.4.6 گراف زیر همبند قوی است؟ یا همبند ضعیف چرا؟



راه حل:

همبند قوی زیرا مسیری جهت دار بین همه راس هایش وجود دارد.

---

8.4.7 الگوریتمی ارائه دهید تا تعداد یال هایی که باید به یک گراف اضافه کنیم. تا قوی همبند شود را محاسبه کند؟

هزینه این الگوریتم را بیابید.

راه حل:

ابتدا از گراف DFS می زنیم. و بعد از ترانهاده آن DFS می زنیم. با این کار مولفه های قوی همبند به دست می آیند.

حال یک گراف جدید ایجاد می کنیم. که رئوس آن مولفه های قوی همبند هستند. و اگر بین دو مولفه قوی همبند یال وجود داشته باشد. در گراف جدید بین آن دو یال جهت دار می گذاریم. به وضوح گراف جدید یک DAG است. (چرا که اگر دور داشت به این معنی بود که رئوس روی دور همگی به یکدیگر راه داشتند. و از ابتدا مولفه های قوی همبند جداگانه نبوده است. و باید همگی یک مولفه می شدند.)

در این DAG تعداد رئوسی که یال ورودی ندارند. را می شماریم. و آن را C1 می نامیم. سپس تعداد رئوسی که یال خروجی ندارند. را می شماریم. و آن را C2 می نامیم. اگر DAG فقط یک راس داشته باشد. لازم نیست یالی اضافه کنیم. در غیر این صورت باید MAX (C1, C2) یال اضافه کنیم.

---

8.4.8 مجموعه ای از عبارات منطقی داریم. به گونه ای که هر عبارت منطقی، تشکیل شده از or دو متغیر می باشد. که هر کدام از این دو متغیر، می توانند به شکل نقیض یا not شده باشند. الگوریتم بهینه و غیر مبتنی بر سعی و خطأ ارائه دهید که مشخص نماید. آیا می توان متغیر های یک مجموعه ای عبارت را به گونه ای مقدار دهی کرد که تمامی عبارات داده شده، مقدار true پیدا کنند؟

به عنوان مثال: مجموعه عبارات زیر را در نظر بگیرید:

$$E1 = A + B$$

$$E2 = \sim A + C$$

$$E3 = \sim C + \sim B$$

اگر مقدار A برابر true، مقدار B برابر false و مقدار C برابر true باشد. آنگاه تمامی عبارات E1, E2, E3 مقدار true خواهند کرد.

راه حل:

به ازای هر متغیر دخیل در مسئله مانند X دو راس تحت نام های  $\sim X$ ، X اضافه می کنیم. به ازای هر عبارت مانند  $X+Y$  یال های  $(\sim X, Y)$  و  $(X, \sim Y)$  را به گراف اضافه می کنیم. وجود یال  $(U, V)$  در گراف بیان می کند. که اگر U درست باشد. آنگاه V هم حتما درست است. با این کار عملاً داریم محدودیت های مسئله را با گراف مدل سازی می کنیم. حال روی این گراف، با دو بار اجرای الگوریتم dfs مولفه های قوی همبند را استخراج می کنیم. سپس به ازای هر مولفه قوی همبند یک راس می گزاریم و اگر بین دو راس از دو

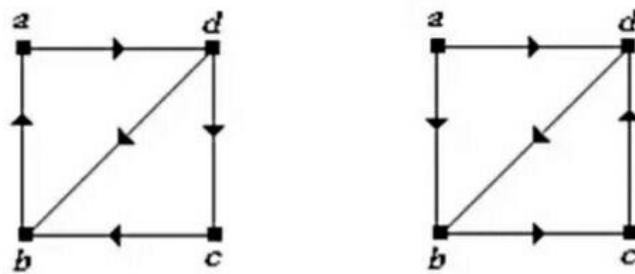
مولفه یال وجود دارد. با جهت مناسب ، دو مولفه را به هم وصل می کنیم. اما مطمئن هستیم که دور به وجود نمی آید. زیرا اگر دور داشتیم. آن وقت اجتماع این دو مولفه همبندی باز خود یک مولفه همبندی می بود. و الگوریتم ما استخراجش می کرد.

ضمن این کارها ، به ازای هر متغیر شماره مولفه‌ی آن را نیز ذخیره می کنیم. در پایان به ازای هر متغیر  $X$  چک می کنیم. که شماره مولفه‌ی  $X$  و  $\sim X$  مساوی نباشند. اگر به ازای همه رؤوس این قاعده برقرار بود. می توان گفت که امکان حل این مسئله وجود دارد.

حالا برای این که بفهمیم مقدار هر متغیر باید چه باشد به این شکل عمل می کنیم. آخرین گرافی که از مولفه‌های قوی همبند ساختیم یک dag است. حالا روی آن topological sort می زنیم. اگر مولفه متغیر  $X$  قبل از مولفه متغیر  $\sim X$  باید آنگاه  $X$  را  $.true$  می گذاریم. در غیر این صورت  $false$ .

نکته مهم : اگر گراف ناهمبند باشد. و مسیری از مولفه‌ی شامل  $X$  به مولفه شامل  $\sim X$  نباشد یا برعکس آن ، در این صورت متغیر  $X$  هر مقداری می تواند داشته باشد.

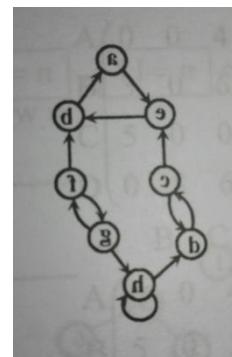
#### 8.4.9 کدام یک از شکل‌های زیر همبند قوی و کدام یک همبند ضعیف است؟



راه حل:

شکل سمت چپ همبند قوی زیرا مسیری جهت دار بین همه راس‌هایش وجود دارد و شکل سمت راست همبند ضعیف است.

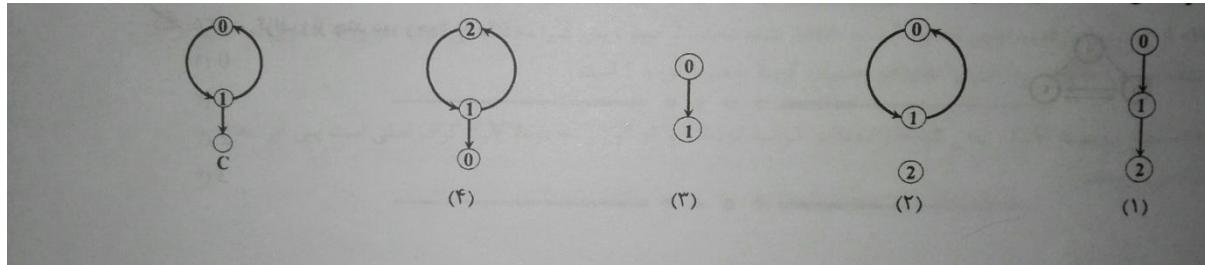
#### 8.4.10 گراف جهت دار زیر چند مولفه متصل به صورت قوی دارد؟



راه حل :

{ a,e,b} , {g,f} ,{c,d} , {h} ۴ تا ۴

مولفه های کاملا متصل گراف زیر کدام است؟ 8.4.11



راه حل:

گرینه دوم درست: نکته: گراف با یک راس خود کاملا متصل است.

## نمونه سوالات مربوط به بخش ۸.۵

۸.۵.۱ گراف بدون جهت و وزن دار ( $G = (V, E)$ ) را در نظر بگیرید که در آن وزن هر یال عددی بین صفر و یک است. از  $G$  گراف  $G'$  را با همان راس ها و یال ها می سازیم و وزن هر یال  $e$  در  $G'$  را برابر  $w_e - 1$  قرار می دهیم. حال با استفاده از الگوریتم کروسکال درخت فرآگیر کمینه  $G'$  را (به نام  $T$ ) می سازیم.

کدام یک از روابط زیر درست است:

- (۱) یال های  $T$  در گراف  $G$  تشکیل درخت پوشایشینه میدهد.
- (۲) یال های  $T$  در گراف  $G$  تشکیل MST میدهد.
- (۳) هر MST در  $G$ ، دست کم در یک یال با  $T$  اشتراک دارد.
- (۴) هیچ کدام

راه حل:

گزینه ۱: اگر بخواهیم درخت فرآگیر بیشینه را در  $G$  با الگوریتم کروسکال بسازیم، باید یالها را به ترتیب نزولی مرتب کنیم. این ترتیب مشابه ترتیب صعودی است اگر یال ها وزنشان  $w_e - 1$  شده باشد.

---

۸.۵.۲ قدرت یک درخت فرآگیر در  $G$  برابر وزن سنگین ترین یال در آن درخت است. می خواهیم در گراف داده شده  $G$  از میان تمام درخت های فرآگیر، آن را انتخاب کنیم که قدرتش از همه کمتر باشد. خروجی کدام یک از الگوریتم های پریم یا کروسکال همواره کم قدرت ترین درخت فرآگیر است؟

راه حل:

کم قدرت ترین بودن الگوریتم کروسکال معلوم می باشد. این الگوریتم به صورت حریصانه و بعد از مرتب سازی یال ها بر اساس وزنشان هر یالی را که میبینند برای زیر درخت فرآگیر انتخاب میکنند تا هرچه زودتر درخت فرآگیر کمینه اش را بسازد و زمان کمتر به معنی استفاده از یال های سبکتر است.

در الگوریتم پریم اگر یال های کم قدرت ترین درخت فرآگیر را در نظر بگیریم میبینیم که این الگوریتم تمامی این یال ها را مشاهده می کند و در پایان کار جوایش بهتر یا مساوی جواب بهینه است.

---

در حالت کلی اثبات میشود که دنباله های وزنی یال های تمام درخت های فرآگیر کمینه  $G'$  یک گراف ثابت با هم برابرند.

۸.۵.۳ فرض کنید  $G$  یک گراف ساده، بدون جهت و وزن دار با وزن های مثبت است و  $K$  تعداد درخت های فرآگیر کمینه  $G$  است. دو درخت فرآگیر کمینه مختلف اند اگر دست کم در یک یال اختلاف داشته باشند. کدام ۲ گزاره زیر در مورد رابطه  $K$  با متفاوت بودن وزن یال های  $G$  درست اند؟

(۱) اگر  $k=1$ ، ناگزیر وزن یال های  $G$  دو به دو متفاوت است.

(۲) اگر دست کم دوتا از یال های  $G$  وزن یکسان داشته باشند، آنگاه  $k=2$

راه حل:

(۱) فرض کنید  $G$  یک درخت ساده و همبند با  $n$  یال باشد که وزن همه یال های آنها یک است. روشن است که  $G$  تنها یک درخت فرآگیر کمینه دارد و آن هم خودش است و در این حالت  $k=1$  ولی وزن یال ها همگی یکسان اند پس این جمله نادرست است.  
(۲) همان مثال گزاره یک را در نظر بگیرید، این جمله با توجه به همان مثال نادرست است.

---

۸.۵.۴ در گراف وزن دار  $G$  فرض کنید که وزن های یال ها دو به دو متفاوت اند و مجموعه های  $A, B$  یک افزار دلخواه از راس های گراف اند که  $A \cup B = V$ . اگر یال  $e$  کم وزن ترین یالی باشد که  $A$  را به  $B$  وصل میکند، آنگاه کدام یک از گزاره های زیر درست اند؟

(۱)  $e$  ناگزیر در تمام درخت های فرآگیر کمینه  $G$  آمده است.

(۲)  $e$  در هیچ یک از درخت های فرآگیر کمینه  $G$  نیامده است.

(۳)  $e$  شاید در بعضی درخت های پوشایی کمینه  $G$  بیاید.

(۴) هیچ کدام

راه حل:

گزینه یک: فرض خلف:  $e$  در درخت فرآگیر کمینه نیامده باشد، پس بین  $V$ ،  $U$  یک مسیر در درخت فرآگیر مربوطه وجود دارد فرض کنیم این مسیر  $a_0, a_1, a_2, \dots, a_{k-1}, a_k = V$  باشد. روی مسیر از  $U$  شروع میکنیم و به  $V$  میرویم. روشن است که اندیس  $i$  وجود دارد که  $a_i \in A$  و  $a_{i+1} \in B$ . حال چون  $e$  کم وزن ترین یال بین مجموعه  $A$  و  $B$  است، پس وزن یال  $e = (a_i + a_{i+1})'$  حتما از  $e$  بیشتر است. پس اکنون با افزودن  $e$  و حذف  $e'$  میتوان به یک درخت فرآگیر کمینه با وزن کمتر رسید که تناقض است پس  $e$  حتما در درخت پوشایی کمینه هست.

---

۸.۵.۵ گراف ساده و وزن دار  $G$  را در نظر بگیرید. کوتاهترین مسیر بین هر دو راس  $v, u$  را محاسبه و خود مسیر را در درایه  $D_{uv}$  از ماتریس  $D$  ذخیره می کنیم، به وزن هر یال  $G$  به طور دقیق به اندازه یک واحد افزایش می دهیم. پاسخ دو پرسش زیر کدام است؟

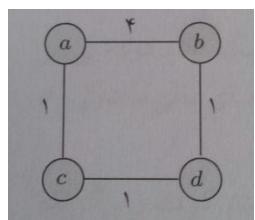
الف) آیا برای دو راس  $v, u$  مسیر ذخیره شده در  $D_{uv}$  همچنان کم وزن ترین مسیر است؟

ب) آیا هر درخت فراگیر کمینه در  $G$  همچنان درخت فراگیر کمینه باقی می ماند؟

راه حل:

الف: خیر – به عنوان مثال گراف زیر را در نظر بگیرید، در این گراف کوتاهترین مسیر بین راس های  $a, b$  وزنی برابر ۳ دارد

حال آنکه با افزایش یک واحد مسیر بهینه بین  $a, b$  به ۵ تغییر کرده است.



شکل شماره ۱

ب: بله- چون ترتیب یال ها از نظر وزن ثابت می ماند و الگوریتم کروسکال همان رویه پیشین را می پیماید.

۸.۵.۶  $T$  درخت فراگیر کمینه برای گراف  $G$  است، وزن یک یال  $e$  را کاهش می دهیم. با چه مرتبه ای می توان درخت فراگیر کمینه گراف جدید را به دست آورد؟

راه حل:

در صورتی که این یال در  $T$  باشد،  $T$  همچنان درخت فراگیر کمینه باقی می ماند. اما در صورتی که این یال در  $T$  نباشد، تنها تغییر ممکن افزودن  $e$  به  $T$  و حذف یک یال از  $T$  است. می دانیم در صورت افزودن  $e$  به  $T$ ، یک دور به وجود می آید و سنگین ترین یال این دور باید حذف شود.

برای یافتن این دور کافی است مسیر یکتای بین دو سر  $e$  در  $T$  را بیابیم و از آنجا که  $T$  درخت است، این کار با الگوریتم BFS یا DFS از زمان  $O(|V(T)|)$  قابل انجام است.

۸.۵.۷ درخت فراگیر گلوگاه  $T$  در یک گراف بدون جهت و وزن دار  $G$  یک درخت فراگیر است که وزن سنگین ترین یال آن نسبت به هر درخت فراگیر دیگر کمینه باشد. می گوییم که مقدار یک درخت فراگیر گلوگاه، وزن سنگین ترین یال آن درخت است. کدام یک از گزینه های زیر درست است؟

۱) هر درخت فراگیر کمینه یک درخت فراگیر گلوگاه است.

۲) هر درخت فراگیر گلوگاه یک درخت فراگیر کمینه است.

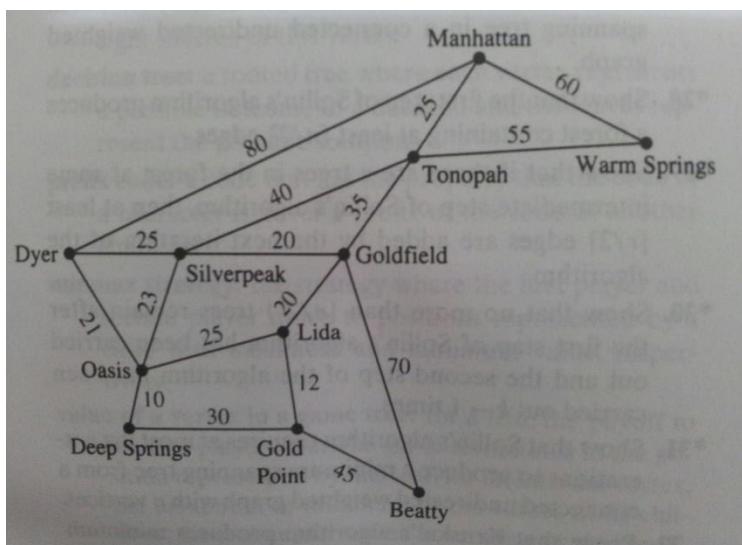
۳) هر درخت فراگیر بیشینه یک درخت فراگیر گلوگاه است.

۴) هر درخت فراگیر گلوگاه یک درخت فراگیر بیشینه است.

راه حل:

گزینه یک: فرض کنید  $T$  که یک درخت فراگیر کمینه است و گلوگاه نیست، پس یک درخت فراگیر به نام  $T'$  وجود دارد که وزن سنگین ترین یال  $T$  به نام  $e$  از همه یال های  $T'$  بیشتر است. یال  $e$  را از  $T$  بر میداریم و بجای آن یک یال از  $T'$  قرار می دهیم که درخت ساخته شود. درختی را به دست آوردهیم که کمینه تر از  $T$  است. تناقض!

#### ۸.۵.۸ درخت پوشانه گراف زیر را بیابید.



شکل شماره ۲۰

راه حل:

Deep Springs- Oasis , Oasis-Dyer , Oasis-Silverpeak , Silverpeak-Goldfield , Lida-Gold Point , Gold Point-Beatty , Lida-Goldfield , Goldfield-Tonopah , Tonopah-Manhattan , Tonopah- Warm Springs.

۸.۵.۹ گراف ساده و همبند و بدون جهت  $G$  را در تظر بگیرید که وزن تمامی یال‌های آن یک است. سریع ترین الگوریتم برای یافتن درخت پوشای کمینه از چه مرتبه زمانی است؟

راه حل:

از آنجا که هر درخت فراگیری در این گراف کمینه است پس کافیست با استفاده از الگوریتم DFS یا BFS یک درخت پوشای کمینه در آن بیابیم.

۸.۵.۱۰ نشان دهید که درخت پوشای کمینه در گراف همبند وزن‌دار که وزن تمامی یال‌ها متفاوت است یکتاست.

راه حل:

اثبات با استفاده از برهان خلف:

فرض می‌کنیم دو درخت پوشای کمینه وجود دارد  $A, B$

فرض می‌کنیم یالی با کمترین وزن  $e$  فقط در یکی از این دو وجود دارد (درخت  $A$ ). پس درخت  $B$  شامل  $e$  نمی‌باشد.

پس  $B \cup e$  تشکیل دور میدهد که یکی دیگر از یال‌های این دور  $e'$  در درخت  $A$  وجود ندارد. با توجه با اینکه وزن یال  $e$  از یال  $e'$  کمتر است  $(W(e) < W(e'))$  و  $T = B \cup \{e\} \cup \{e'\}$  یک درخت پوشای کمینه است در این صورت وزن درخت  $T$  از وزن  $B$  کمتر است. تناقض!

۸.۵.۱۱ الگوریتم زیر جهت به دست آوردن درخت فراگیر کمینه  $M$  در هر گراف ساده و وزن‌دار و بدون جهت  $G$  داده شده است.

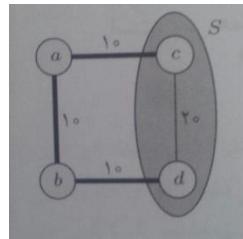
۱. در صورتی که گراف دست کم دو راس دارد، راس‌های آن را به دو مجموعه دلخواه  $S$ ،  $V-S$  تقسیم کن.

۲. درخت فراگیر کمینه را برای هر دو مجموعه و به صورت بازگشتی توسعه همین الگوریتم به دست آور.

۳. کم وزن ترین یال بین  $S$  و  $V-S$  از گام اول را به  $M$  اضافه کن.

آیا این الگوریتم همواره درست کار می‌کند؟

راه حل:



شکل شماره ۳

دقیق کنید که در جواب بهینه ممکن است زیرگراف القایی هر یک از دو بخش  $S$  و  $S^c$  ناگزیر خود یک زیردرخت فراگیر کمینه نباشد.

مثال: در شکل بالا برای رسیدن به جواب بهینه می بایست از مجموعه  $S$  تنها دو راس (و هیچ یال) انتخاب شود که با الگوریتم گفته شده در تناقض است.

#### ۸.۵.۱۲ در مورد درخت فراگیر کمینه در گراف بدون جهت و وزندار کدام گزینه بهترین جواب است؟

۱. الگوریتم های پریم و کروسکال ممکن است درخت های بهینه مختلفی را محاسبه کنند.
۲. اگر تعادل یال های گراف خیلی زیاد باشند الگوریتم پریم ناگزیر سریع تر از کروسکال است.
۳. اگر یال ها منفی ولی گراف بدون دور باشد هر دو الگوریتم نادرست جواب می دهند.
۴. در الگوریتم پریم درخت فراگیر کمینه همان درخت جستوجو سطح اول است.

راه حل:

گزینه یک: گراف کامل  $n$  راسی که وزن تمام یال های آن ۱ است را در نظر بگیرید. هر درخت  $n-1$  راسی در این گراف می تواند جواب باشد الگوریتم پریم چنین گرافی را بر اساس اندیس راس ها اضافه می کند در حالی که در الگوریتم کروسکال ترتیب یال دهی ها مهم است. پس ممکن است متفاوت باشند.

## نمونه سوالات مربوط به بخش 8.6

8.6.1 حسن تعدادی چراگاه دارد که بین بعضی از آنها یالهای وزن دار است. هر چراگاه متعلق به یک گاو است و همه‌ی گاوها می‌خواهد در یک چراگاه جمع شوند. چراگاهی را پیدا کنید که گاوها در آن جمع شوند و جمع مسافت طی شده توسط آنها کمینه شود.

راه حل:

مسئله را بر عکس حل کنید. فرض کنید چراگاه محل تجمع را دارید و می‌خواهید گاوها به جای اول خود برگردند. این مسئله کوتاهترین مسیر است. برای حل سوال کوتاهترین مسیر را از هر رأس حساب کنید و به ازای مکان‌های گاوها با هم جمع کنید. رأس با کمترین مجموع جواب است.

8.6.2 حسن می‌خواهد به یک مزرعه‌ی دیگر نقل مکان کند که مسافتی که هر روز طی می‌کند را کمینه کند. جایی که حسن به آن می‌رود ( $1 \leq N \leq 10000$ ) شهر دارد.  $M$  جاده‌ی دوطرفه ( $1 \leq M \leq 50000$ ) این شهرها را به هم وصل کرده‌اند. همه‌ی شهرها از یکدیگر با دنباله‌ای از جاده‌ها قابل دسترس هستند. حسن باید بهترین شهر را به عنوان خانه‌ی جدیدش انتخاب کند. فروشگاه در شهرها هستند ( $1 \leq K \leq 5$ ) که حسن باید هر روز به آنها برود. به علاوه هر روز حسن می‌خواهد از خانه‌اش به این  $K$  شهر برود و به خانه‌اش برگردد. حسن با هر ترتیب لخواه می‌تواند فقط خانه‌ها را ببیند. وقتی یک شهر را به عنوان خانه ( محل ساخت مزرعه جدید) انتخاب می‌کند، می‌خواهد فقط خانه‌اش از بین  $N-K$  شهری باشد که فروشگاه ندارند، چون قیمت خانه در این شهرها کمتر است. به حسن کمک کنید که کمترین فاصله‌ای که هر روز باید طی کند را کمینه کند؛ اگر بخواهد مزرعه‌ی جدیدش را در بهترین مکان بسازد و ترتیب دیدن مغازه‌ها را هم تا حد امکان هوشمندانه انتخاب کند.

راه حل:

کوتاهترین مسیر را از این  $K$  شهر پیدا می‌کنیم. به ازای هر شهر از  $N-K$  شهر به عنوان خانه جدید و دو شهر از  $K$  شهر فاصله‌ی این دو شهر از خانه جدید و مسافت کمترین ترتیب  $K$  شهر طوری که این دو شهر دو سر گشت قرار بگیرند؛ را حساب می‌کنیم. کمینه‌ی اینها جواب مسئله است. (به ازای هر ترتیب  $K$  شهر باید کوتاهترین مسیر را حساب کنیم. می‌توانیم همه‌ی کوتاهترین مسیرها بین  $K$  رأس با خودشان را به دست بیاوریم که کار ساده‌تر باشد).

8.6.3 علی و امیر می‌خواهند از قم به مشهد بروند و چون مسیر طولانی است، می‌خواهند نوبتی رانندگی کنند و در هر استراحتگاه میان راهی جای خود را عوض کنند. چون علی حسجه‌یابی بهتری دارد، باید در شهرهای مبدأ و مقصد او رانندگی کند و امیر فقط در جاده رانندگی می‌کند. نقشه‌ی مسیر به شکل یک گراف وزن دار جهت‌دار داده است که وزن یالهای آن مثبت است و رأسهای آن استراحتگاه‌ها و یالهای آن جاده‌ها بین آنها هستند. الگوریتمی ارائه دهید که در صورت امکان، کوتاهترین مسیر بین قم و مشهد را پیدا کند، طوری که علی و امیر بتوانند نوبتی رانندگی کنند.

راه حل:

کوتاهترین مسیرها بین همه‌ی زوج رأسها را حساب می‌کنیم و تعداد یالهای آنها را هم علاوه بر طول آنها نگه می‌داریم. کوتاهترین مسیر با طول فرد بین قم و مشهد را در صورت وجود بر می‌گردانیم و اگر کوتاهترین مسیرها همه طول زوج داشتند جواب خیر (غیرممکن) است.

8.6.4 در یک شهر قوانین رانندگی عجیبی وجود دارد. این شهر تعدادی تقاطع دارد که بین آنها راههای دوطرفه بین تقاطع‌های متمايز است. چراغ‌های راهنمایی در هر تقاطع وجود دارند که رنگ آنها به صورت تناوبی مدتی آبی و مدتی قرمز است. در صورتی می‌شود در یک راه تردد که وقتی به آخر آن می‌رسیم چراغ راهنمایی همنگ وقته باشد که در ابتدای آن هستیم (اگر چراغ همان لحظه عوض شود رنگ جدید آن ملاک است). در نقشه‌ی شهر زمان طی کردن همه‌ی جاده‌ها، زمان هر رنگ

هر چراغ راهنمایی و رنگ اولیه هر چراغ راهنمایی و زمان باقیمانده تا تغییر رنگ آن داده شده‌اند. (همه‌ی این اعداد صحیح هستند). شما باید مسیری را پیدا کنید که کمترین زمان برای رسیدن از یک مبدأ داده شده به یک مقصد داده شده را بدهد. اگر بیش از یک مسیر با چنین خصوصیاتی وجود دارد، یکی از آنها را پیدا کنید. در ضمن ماشین‌ها می‌توانند در یک تقاطع توقف کنند.

راه حل:

الگوریتم کوتاهترین مسیر بین همه‌ی رأسها را تعییر بدھید تا توقف در تقاطع‌ها را هم حساب کند. در اینه متناظر مبدأ و مقصد جواب مسئله است.

---

8.6.5 گرافی وزندار و جهت دار  $(V, E) = G$  بدون دور منفی را در نظر بگیرید. فرض کنید  $m$  حداقل تعداد جفت رفوس  $V$  از  $u, v \in u, v$  به  $v$  باشد. (در اینجا کوتاه ترین مسیر وزندار مدنظر است نه تعداد یالها) در الگوریتم Bellman-Ford تغییری ایجاد کنید تا در  $m+1$  بار عبور پایان یابد

راه حل:

ویژگی همگرایی نشان میدهد که برای هر راس  $v$ , الگوریتم کوتاه ترین مسیر  $[v]d$  را تخمین می‌زند مقدار نهایی انرا (هر کوتاهترین مسیر به  $v$ ) بعد از تکرار الگوریتم BELLMAN-FORD بست می‌ورد. بدین گونه پس از  $m$  بار تکرار, BELLMAN-FORD به اتمام میرسد. در ابتدا ما  $m$  را نمیدانیم, پس نمیتوانیم تکرار حلقه را دقیقا  $m$  بار انجام و سپس به ان خاتمه دهیم. ولی ما دقیقا وقتی هیچ چیز دیگر تغییر نمیکند الگوریتم را خاتمه میدهیم, و پس از  $m+1$  تکرار پایان می‌بینیم (پس از یک تکرار که هیچ تغییری اتفاقی نیافتد)

BELLMAN-FORD-(M+1)(G,w,s)

INITIALIZE-SINGLE-SOURCE(G,s)

changes  $\leftarrow$  TRUE

while changes= TRUE

do changes  $\leftarrow$  FALSE for each edge  $(u,v) \in E[G]$

do RELAX-M( $u,v,w$ )

RELAX-M( $u,v,w$ )

if  $d[v] > d[u]+w(u,v)$

then  $d[v] \leftarrow d[u]+w(u,v)$

$\pi[v] \leftarrow u$

changes  $\leftarrow$  TRUE

چک کردن عدم وجود حلقه منفی (بر این اساس که بک  $d$  وجود دارد که اگر یک relaxation دیگر انجام شود تغییر می‌کند) حذف شده به این علت که این الگوریتم از حلقه خارج نمی‌شود مگر انکه همه  $d$  ها تغییر نکنند.

---

8.6.6 گرافی جهت دار  $(V, E) = G$  را بگیرید که در ان هریال  $E \in (u, v)$  ارزشی برابر با  $r(u, v)$  که عددی حقیقی بین 0 تا 1 است که قابلیت اطمینان ارتباط کانال از راس  $u$  به  $v$  را نشان میدهد دارد. ما  $r(u, v)$  را بعنوان احتمال اینکه کانال از  $u$  به  $v$  شکست خورد در نظر میگیریم و فرض میکنیم که این احتمالات مستقل هستند. الگوریتمی ارایه دهید که قابل اطمینان ترین مسیر بین دو راس را پیدا کند.

راه حل:

برای پیدا کردن قابل اطمینان ترین راه بین  $s$  و  $t$  الگوریتم دایکسترا را با وزن بالهای  $w(u, v) = -\lg r(u, v)$  برای پیدا کردن کوتاه ترین مسیر از  $s$  در زمان  $O(E + V \lg V)$  اجرا میکنیم. قابل اطمینان ترین مسیر کوتاه ترین مسیر از  $s$  به  $t$  و ارزش اطمینان ان برابر با حاصلضرب ارزش اطمینان یال های ان است.

به دلیل اینمه احتمالات مستقل هستند، احتمال این که یک مسیر شکست خورد برابر است با حاصلضرب احتمالات اینکه بالهای ان شکست خورند. ما میخواهیم مسیری از  $t \rightarrow s$  به طوری که  $\prod_{(u,v) \in p} r(u, v)$  حداقل شود. این هم ارز است با اینکه عبارت

$$\lg(\prod_{(u,v) \in p} r(u, v)) = \sum_{(u,v) \in p} \lg r(u, v)$$

$$\lg r(u, v) = \lg r(u, v) - \lg r(u, v)$$

میکنیم. همچنین اگر اوزان را برابر با  $w(u, v) = -\lg r(u, v)$  میتواند صفر باشد و 0 تعريف نشده است. پس در این الگوریتم  $-\infty = 0$  تعريف

چون  $1 < x < 0$  for  $0 < \lg x = 0$  و  $\lg 1 = 0$ ،  $\lg x < 0$  for  $x < 1$  را تعريف کردیم، همه ی اوزان  $w$  نامنفی هستند و حالا میتوانیم از دایکسترا برای پیدا کردن کوتاه ترین مسیر ها از  $s$  در زمان  $O(E + V \lg V)$  استفاده کنیم.

روش دوم:

هم چنین میتوان از اصل احتمالات با اجرای ورژنی تغییر داده شده از الگوریتم دایکسترا که حاصلضرب قابلیت اطمینان ها در طول یک مسیر را ماسکیم میکند بجا اینکه مجموع اوزان یال ها در طول یک مسیر را مینیم کنیم

در دایکسترا از قابلیت اطمینان بعنوان وزن یال ها استفاده کنید

- max (and EXTRACT-MAX) for min (and EXTRACT-MIN) in relaxation and the queue,
- $\times$  for  $+$  in relaxation,
- 1 (identity for  $\times$ ) for 0 (identity for  $+$ ) and  $-\infty$  (identity for min) for  $\infty$  (identity for max).

برای مثال الگوریتم پایین برای استفاده بجای عمل relax استفاده میشود

RELAX-RELIABILITY( $u, v, r$ )

if  $d[v] < d[u]$   $r(u, v)$

then  $d[v] \leftarrow d[u]$   $r(u, v)$

$\pi[v] \leftarrow u$

---

8.6.7 فرض کنید  $G = (V, E)$  گرافی وزندار و جهت دار دارای تابع وزن  $R \rightarrow E : w$  و راس مبدا  $s$  است. ثابت کنید برای همه ی یال ها  $e \in E$  داریم  $(u, v) \in E : \delta(s, v) \leq \delta(s, u) + w(u, v)$ .

راه حل:

فرض کنید که کوتاه ترین مسیر  $p$  مبدا  $s$  به راس  $v$  وجود دارد. پس  $p$  از هر مسیر دیگری از  $s$  به  $v$  کم وزن تر است. بویژه مسیر  $p$  از مسیر منحصر بفردی که کوتاه ترین مسیر از مبدا  $s$  به راس  $v$  را در اختیار دارد و سپس یال  $(u, v)$  را اختیار میکند وزن بیشتری ندارد.

---

**8.6.8 گراف وزندار و جهت دار**  $G = (V, E)$  با راس مبدا  $s \in V$  را فرض کنید و  $G$  تابع-INITIALIZE-SINGLE-SOURCE( $G, s$ ) بروی ان فراخوانی شده است. ثابت کنید اگر دنباله ای از ریلکسیشن ها مقدار  $[s]_{\pi}$  را به تغییر دهد، سپس  $G$  دارای دوری منفی است.

راه حل:

هرگاه RELAX $(\pi)$ , پدر  $(\pi)$  را برای راسی تعیین میکند همچنین مقدار  $d$  را کاهش میدهد. سپس اگر  $[s]_{\pi}$  به مقدار non-NIL است شده  $[s]_{\pi}$  از مقدار اولیه اش که صفر بوده به مقداری منفی میرسد. اما  $[s]_{\pi} d$  وزن مسیری از  $s$  به  $s$  باشد، که در واقع دوری شامل  $s$  است. پس دوری منفی داریم.

---

**8.6.9 گراف وزندار و جهت دار**  $G = (V, E)$  که هیچ دور منفی ندارد را فرض کنید. راس  $s \in V$  را راس مبدا در نظر بگیرید و روی  $G$  تابع INITIALIZESINGLE-SOURCE( $G, s$ ) اجرا شود. ثابت کنید که دنباله ای با  $-|V|$  بار ریلکسیشن که  $d[v] = \delta(s, v)$  را برای همه  $v \in V$  تولید میکند.

راه حل:

فرض کنید درخت کوتاه ترین مسیر هارا که اسمش  $(\pi)G$  است را در اختیار داریم. یال های داخل  $(\pi)G$  را بر اساس ترتیبی که BFS اونهارو ملاقات میکنیم. سپس ما تضمین میکنیم که یال های هر کوتاه ترین مسیر به ترتیب ریلکس شده اند. بر اساس ویژگی path-relaxation برای هر  $v \in V$  داریم  $d[v] = \delta(s, v)$ . چون  $(\pi)G$  حداقل  $-|V|$  یال را شامل میشود، ما فقط نیاز داریم تا برای همه راس ها  $d[v] = \delta(s, v)$  بشود

---

**8.6.10 گراف وزندار و جهت دار را فرض کنید. همچنین کوتاه ترین مسیر از راس  $s$  به راس  $t$  داده شده. اگر وزن هر یال را واحد زیاد کنیم ایا کوتاه ترین مسیر در گراف جدید همان مسیر قبلی میماند؟**

راه حل:

کوتاه ترین مسیر ممکن است تغییر کند. به این دلیل که امکان دارد که در مسیر های گوناگون از  $s$  به  $t$  تعداد یال ها متفاوت باشد. برای مثال، کوتاه ترین مسیر دارای وزن 15 می باشد دارای 5 یال. فرض کنید مسیر دیگری با 2 یال و وزن کل 25 باشد. وزن کوتاه ترین مسیر  $5 * 10$  افزایش یافته است و می شود  $15 + 50$ . وزن مسیر دیگر  $2 * 10$  افزایش یافته و  $25 + 20$ . بنابراین کوتاه ترین مسیر به مسیر دیگر با وزن 45 تغییر می یابد.

---

**8.6.11 این شبیه به سوال فوق است. آیا کوتاه ترین مسیر با ضرب وزن همه یال ها در 10 تغییر می یابد؟**

راه حل:  
اگر ما وزن تمام یال ها را ضربدر 10 کنیم، کوتاه ترین مسیر تغییر نمی کند. دلیل آن ساده است، وزن تمام مسیر های از  $s$  به  $t$  توسط همان مقدار ضرب شده است. تعداد یال ها در یک مسیر مهم نیست. این مانند تغییر واحد وزن است.

---

**8.6.12 گرافی جهت دار داده شده که در آن هر یال دارای وزن 1 یا 2 است، کوتاه ترین مسیر از یک راس مبدا  $s$  را به یک راس مقصد  $t$  پیدا کنید. انتظار می رود پیچیدگی زمانی  $(V + E)O$  شود.**

راه حل:

اگر ما الگوریتم کوتاه ترین مسیر دیکسترا را اجرا کنیم، می توانیم کوتاه ترین مسیر را در زمان  $O(V + E) \log V$  پیدا کنیم. چگونه آن را در زمان  $O(V + E) \log V$  پیدا کنیم؟ ایده این است که از BFS استفاده کنیم. یکی از مشاهدات مهم در مورد BFS است، راه موردن استفاده در BFS همیشه حداقل تعداد یالهای بین هر دو راس را دارد. بنابراین اگر همه یال ها وزنشان یکی باشد، ما می توانیم از BFS برای پیدا کردن کوتاه ترین مسیر استفاده کنیم. برای این مشکل، ما می توانیم نمودار تغییر و تقسیم تمام لبه وزن 2 به دو لبه وزن 1 هر یک از در نمودار اصلاح شده، ما می توانیم برای پیدا کردن کوتاه ترین مسیر گراف را بهینه کنیم. به این صورت که یالهای با وزن دو را به دو یال با وزن 1 تبدیل کنیم و در گراف بهینه میتوانیم از BFS برای پیدا کردن کوتاه

ترین مسیر استفاده کنیم. این رویکرد چگونه  $(V + E)O$  است؟ در بدترین حالت، تمام یال‌ها دارای وزن 2 هستند و مانیز به انجام  $O(E)$  عملیات برای تقسیم تمام یال‌ها داریم ، به طوری که پیچیدگی زمانی می‌شود:  $O(E) + O(V + E)$  که برابر است با :  $O(V + E)$

---

8.6.13 در یک بازار ارز برای تبدیل هر دو واحد پول یک ضریب وجود دارد . مثلاً اگر ضریب تبدیل واحد پول  $a$  به واحد پول  $b$  برابر 3 باشد میتوان 10 واحد از  $a$  را به 30 واحد از  $b$  تبدیل کرد. وقت که نرخ تبدیل میتواند غیر صحیح باشد ( مثلاً  $1/5$ ) الگوریتمی ارایه دهید تا یک چرخه سود اور را در صورت وجود پیدا کند (چرخه‌ای که بتوان با شروع از مبلغی پول به واحد  $a$  و با تعدادی تبدیل نرخ به مقدار بیشتری پول در واحد  $a$  دست یافت )

راه حل:

مساله را با گراف مدل سازی می‌کنیم به این شکل که رئوس را برابر واحداً قرار میدهیم و وزن یال‌ها را به این شکل قرار میدهیم که اگر نرخ یک تبدیل برابر  $w$  باشد وزن یال متناظر را  $w/1$  قرار میدهیم یک چرخه نیز معادل یک دور در گراف خواهد بود. چرخه‌ی سودآوری چرخه ایست که حاصل ضرب یال‌های آن کوچک تراز 1 باشد. معنا این جمله این است که حاصل جمع لگاریتم وزن یال‌ها منفی باشد. حال از الگوریتم Bellman Ford برای پیدا کردن دور منفی استفاده می‌کنیم به این صورت که اگر در آخرین دفعه‌ی استفاده از Bellman Ford برای فهمیدن دور منفی فهمیدیم که  $v$  جزو این دور است . از  $v$  شروع میکنیم و به  $(v) \Pi$  میرویم و همین منوال را ادامه میدهیم تا دوباره به  $v$  برسیم . به این شکل دور منفی را پیدا خواهیم کرد .

---

8.6.14 الگوریتمی بهینه ارائه دهید تا در یک گراف جهت دار با وزن مثبت ، کوتاه‌ترین مسیر از  $s$  به  $t$  را حساب کند با در نظر گرفتن اینکه اجازه دارید وزن یکی از یال‌ها صفر کنید . هزینه‌ی زمانی الگوریتم ارائه شده را نیز محاسبه کنید .

راه حل:

ابتدا با اجرای الگوریتم دایکسترا بر روی گراف ، کوتاه‌ترین مسیر از  $s$  به تمام رؤوس را محاسبه میکنیم . سپس با اجرای دوباره الگوریتم دایکسترا کوتاه‌ترین مسیر از تمام رؤوس تا  $t$  را محاسبه میکنیم . سپس به از ای هر یال  $(u,v)$  مقدار  $d(s,u)+d(v,t)$  را حساب میکنیم و بین این مقادیر مینیمم میگیریم . هزینه زمانی الگوریتم هم  $O(E + V)\log v$  خواهد بود .

---

8.6.15 بدون تغییر الگوریتم دایکسترا، الگوریتمی ارائه دهید تا در یک گراف که وزن یال‌ها عددی صحیح میباشد و در صورت وجود چندین کوتاه‌ترین مسیر از  $s$  به  $v$  مسیر با تعداد یال کمتر را پیدا کنیم درستی الگوریتم خود را اثبات کنید .

راه حل:

ایده‌ی اصلی این است که به تمام یال‌ها وزن مساوی اضافه کنیم و در این بین دو مسیر با طول برابر ، مسیری که دارای تعداد یال بیشتری است ، درنهایت طول بیشتری پیدا خواهد کرد.اما مشکلی که وجود دارد این است که کوتاه‌ترین مسیر بعد از این تغییرات دیگر کوتاه‌ترین مسیر نباشد . نکته این است که باید به وزن هر یال حداقل  $(v-1+1)/v$  واحد اضافه کنیم چرا که طول یک مسیر حداقل  $v-1$  است و اینگونه تضمن میکنیم که طول هر مسیر کمتر از 1 واحد افزایش پیدا نمیکند . با توجه به اینکه وزن یال‌ها عددی صحیح می‌باشد تفاوت طول دو مسیر حداقل 1 میباشد . لذا امکان ندارد مسیر دلخواه  $s_1$  که طلوش کمتر از مسیر دلخواه  $s_2$  بود بعد از اعمال این روش طلوش از  $s_2$  بیشتر شود و همچنان کوتاه‌ترین مسیر باقی می‌ماند . بعد از اعمال این تغییر ، الگوریتم دایکسترا را روی گراف تغییر یافته اعمال می‌کنیم.

---

8.6.16 گراف  $G$  داده شده است . تعداد کوتاه‌ترین مسیر‌های موجود در این گراف را بیابید .

راه حل:

الگوریتم فلويد-وارشال را اینگونه تغییر میدهیم: در هر گام الگوریتم علاوه بر طول کوتاه‌ترین مسیر پیدا شده تعداد مسیر‌های با این طول را نیز نگه میداریم (در ماتریس count) برای گام بهروزرسانی اینگونه عمل میکنیم: اگر طول مسیری از که از  $a$  به

b از طریق  $v$  میرود کوتاه تر از مقدار فعلی بود ،  $\text{count}[a][b] = \text{count}[a][v] \times \text{count}[v][b]$  فرار میدهیم، اگر طول مسیر جدید مساوی مقدار قبلی بود ،  $\text{count}[a][b] = \text{count}[a][b] + \text{count}[a][v] \times \text{count}[v][b]$  فرار میدهیم .

---

**8.6.17** گراف  $G$  داده شده است . با استفاده از الگوریتم فلوید-وارشال وجود دور منفی در  $G$  را بررسی کنید .

راه حل:

ادعا میکنیم گراف دور منفی دارد اگر و تنها اگر پس از اجرا الگوریتم فلوید-وارشال اگر مقدار  $\text{distance}[v][v]$  به ازای حداقل یک راس منفی باشد یک سمت ادعا واضح است، در صورتی که چنین راسی موجود باشد گراف دور منفی دارد. از سمت دیگر ، یک دور منفی در گراف را در نظر میگیریم . دنباله روسوس موجود در این دور را  $(v(1) \dots v(k))$  مینامیم به گونه ای که  $v(k)$  راس با بیشترین شماره است. هنگامی که در زمان اجرای الگوریتم فلوید-وارشال راس میانی برابر  $v(k)$  است . مقدار  $\text{distance}[v(1)][v(k-1)]$  حداقل برابر با طول مسیر  $(v(1) \dots v(k))$  است . سپس با بروزرسانی از طریق  $\text{distance}[v][v]$  با توجه به منفی بودن طول دور یک عدد منفی میشود .

---

**8.6.18** گراف  $G$  داده شده است طول کوتاه ترین دوری در  $G$  که تعدادی فردی یال دارد را بیابید .

راه حل:

ابتدا طول کوتاه ترین مسیرهای با زوج یال و فرد یال را بین هرجفت راس پیدا میکنیم. برای این کار الگوریتم فلوید-وارشال را اینگونه تغییر میدهیم: بجای نگه داشتن یک ماتریس شامل طول کوتاه ترین مسیرهای بین روسوس تا مرحله فعلی ، یک ماتریس شامل طول کوتاه ترین مسیر با زوج یال و یک ماتریس شامل طول کوتاهترین مسیر با فرد یال نگه میداریم ، برای اشاره به این دو ماتریس به ترتیب  $\text{sep}[u][v]$  و  $\text{sop}[u][v]$  را استفاده میکنیم. مقادیر اولیه این دو ماتریس به شکل های زیرند:

$$\text{sop}[u][v] = w[u][v] , \text{if } u \neq v , \text{sop}[u][u] = \text{Inf}$$

$$\text{sep}[u][v] = \text{inf} , \text{if } u \neq v , \text{sep}[u][u] = \text{.}$$

برای گام بروزرسانی اینگونه عمل میکنیم:

$$\text{sop}[a][b] = \min(\text{sop}[a][b], \text{sep}[a][v] + \text{sop}[a][v], \text{sop}[a][v] + \text{sep}[a][v])$$

$$\text{sep}[a][b] = \min(\text{sep}[a][b], \text{sep}[a][v] + \text{sep}[v][b], \text{sop}[a][v] + \text{sop}[v][b])$$

سپس طول کوتاه ترین دور فرد به شکل زیر پیدا میشود :

$$\min_{u,v,w \in v(G)} (\text{sop}[u][v] + \text{sop}[v][w] + \text{sop}[w][u], \text{sep}[u][v] + \text{sep}[v][w] + \text{sep}[w][u])$$

## نمونه سوالات مربوط به بخش ۸.۷

۸.۷.۱ دو آرایه  $A$  مرتب شده‌ی به طول  $m$  و  $n$  داده شده‌اند. می‌خواهیم مانع ام  $m+n$  عدد را پیدا کنیم.

(الف) روشی با زمان  $O(\log m * \log n)$  برای این کار ارائه دهد.

(ب) روشی با زمان  $O(\log m + \log n)$  برای این کار ارائه دهد.

راه حل:

(الف) عضو  $a$  از آرایه اول و عضو  $b$  از آرایه دوم را به صورت مخواهیم که  $\frac{m+n}{2} = i+j$  باشد. با دو جستجوی دودویی تو در تو این دو عضو را پیدا می‌کنیم. binary search اول روی اعضای آرایه  $A$  اول به جستجوی  $a$  می‌پردازد (با زمان  $O(n\log m)$ ) و به ازای هر عضو با زمان  $O(\log m)$  دنبال عضو  $b$  هستیم. خاستی که این دو عضو دارند این است که عدد زاز آرایه  $i$  دوم نزدیک ترین عدد به عضو  $a$  از آرایه اول می‌باشد. (در نتیجه در زمان  $\log m$  بدست می‌آید) حال اگر  $j = \frac{m+n}{2} - i$  کمتر بود عضو  $b$  را به جلو حرکت می‌دهیم و اگر بیشتر بود عضو  $b$  به عقب حرکت می‌کند.

(ب) مجدد دنبال همان اعضا هستیم، اما نازی نیست به ازای هر عدد  $a$  که از آرایه  $A$  اول پیدا می‌کنیم زمان  $\log m$  مصرف کنیم و دو اندیس  $i$  و  $j$  را همزمان حرکت می‌دهیم تا هر کدام در مجموع به ترتیب زمان‌های  $\log n$  و  $\log m$  طول بکشند. نکته‌ی این قسمت این است که اگر  $j+i < \frac{m+n}{2}$  کمتر از  $\frac{m+n}{2}$  باشد هچ  $'i$  کوچکتری از  $a$  هم نمیتواند  $'j$  معادلی پیدا کند که  $'j+i = \frac{m+n}{2}$  باشد. همسنطور برای  $j$  هم چنین چزی داریم. در نتیجه یکی درمان یکی از آنها را رو به جلو یا عقب در آرایه  $A$  خود حرکت می‌دهیم تا به جایی برسند که دیگر حرکت نکنیم (یعنی نزدیک ترین اعداد به هم باشند) و شرط مورد نظر نز برقرار باشد.

---

۸.۷.۲ مساله زیر آرایه بیشینه (maximum-subarray problem)، در این مسأله هدف پیدا کردن بزرگترین زیر مجموعه پوسه از اعداد یک آرایه است، که بیشترین مقدار ممکن را داشته باشد. فرض بر این است که تعدادی از اعداد منفی میباشند، زیرا در صورت مثبت بودن اعداد مسأله بدیهی است.

راه حل:

مراحل مختلف این روش را به شکل زیر دسته بندی میکنیم:

• زیرمسأله‌ها: زیرآرایه بیشینه از  $A[\text{low} \dots \text{high}]$  که در فراخوانی اول  $\text{low} = 1$  و  $\text{high} = n$  است.

• تقسیم: محاسبه نقطه مانی که آن را  $mid$  مینامیم و تقسیم کردن زیرآرایه به دو زیرآرایه با اندازه‌های تقریباً یکسان.

• حل: یافتن زیرآرایه‌ای بیشینه از  $A[\text{mid}+1 \dots \text{high}]$  و  $A[\text{low} \dots \text{mid}]$ .

• ترکیب: یافتن زیرآرایه‌ای بیشینه که از نقطه مانی مسگذرد و انتخاب بهترین جواب از میان این ۳ راه حل.  
برای ترکیب از تابع زیر استفاده میکنیم :

FIND-MAX-CROSSING-SUBARRAY( $A, \text{low}, \text{mid}, \text{high}$ )

```
1  left-sum = -∞
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > left-sum
```

```

6      left-sum = sum
7      max-left = i
8 right-sum = -∞
9 sum = 0
10 for j = mid + 1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)

```

این تابع در حلقه اول که  $mid + 1 - low$  بار اجرا می شود، زیرآرایهای بخشنه که شامل عنصر مانی و عناصری که در سمت چپ آن قرار دارد مسابد.

حلقه دوم هم که  $mid + 1 - high$  بار اجرا می شود، زیرآرایه ای بخشنه که شامل اولین عنصر بعد از عنصر مانی و عناصری که در سمت راست آن قرار دارد می یابد. پس مرتبه اجرای این الگوریتم  $\Theta(n \cdot n = high - low)$  است که  $+1$  روند حل و تقسیم برای مسئله زیرآرایه بخشنه توسط تابع زیر انجام می شود:

---

**Algorithm 2** FIND MAXIMUM SUBARRAY

---

```

function FMS(A, low, high)
    if high == low then
        return (low, high, A[low])
    else
        mid ← ⌊(low+high)/2⌋
        (lowl, highl, suml) ← FMS(A, low, mid)
        (lowr, highr, sumr) ← FMS(A, mid + 1, high)
        (lowc, highc, sumc) ← FMCS(A, low, mid, high)
        if suml ≥ sumr and suml ≥ sumc then
            return (lowl, highl, suml)
        else
            if sumr ≥ suml and sumr ≥ sumc then
                return (lowr, highr, sumr)
            else
                return (lowc, highc, sumc)

```

---

برای سادگی تحلیل زمان اجرای الگوریتم فرض میکنیم  $n$  توانی از ۲ باشد. زمان اجرا را هنگامی که اندازه ورودی مسئله  $n$  است را با  $n(T)$  نشان مدهیم. برای حالت پایه یعنی وقتی که  $1 = n$  است، خط دوم اجرا خواهد شد که زمان ثابتی طول میکشد پس:

$$T(1) = \Theta(1)$$

حالت بازگشتی زمانی رخ خواهد داد که  $1 < n$  باشد، خطوط ۱ و ۳ زمان ثابتی طول میکشند. همچنین زمان اجرای خطوط چهارم و پنجم ممباشد. همچنین میانم زمان اجرای FMCS از مرتبه  $\Theta(n^2)$  است، پس:

$$T(n) = \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) = 2T(n/2) + \Theta(n)$$

در نتیجه رابطه کلی  $n(T)$  به شکل زیر خواهد بود:

$$T(n) = \Theta(1) : n = 1$$

$$T(n) = 2T(n/2) + \Theta(n) : n > 1$$

که همان رابطه بازگشته MergeSort میباشد، پس:

$$T(n) = \Theta(n \log n)$$

۸.۷.۳ . n نقطه در فضای سه بعدی داریم. الگوریتمی با بهترین زمان ارائه دهد که نزدیک ترین جفت نقاط را پیدا کند. اگر فضا چهار بعدی باشد چه تغییری در روش ایجاد می شود و آیا زمان اجرای آن تفاوتی با حالت قبل دارد؟

راه حل:

به روش تقسیم و غلبه حل می کنیم و هر بار صفحه ای انتخاب می کنیم  $z=k$  (که k وسط های نقاط در این مرحله باشد) و دو طرف صفحه را جداگانه حل می کنیم و d را کمترین نقاطی را که نیاز به دو قسمت درنظر میگیریم. حال باید از هر دو قسمت نقاطی را بررسی کنیم که در فاصله i d از این صفحه هستند. به طور مشابه حالت دو بعدی، برای هر نقطه تعداد متنها یی نقطه وجود دارند که باید بررسی شوند. (زیرا در قسمتی که این نقطه است فاصله i حداقل d دارند و نقاطی که در قسمت دیگر نزدیک تر از d هستند از هم حداقل d فاصله دارند) برای این که هزینه مرتب سازی نداشته باشیم (یا مجبور نباشیم همه نقاط را بررسی کنیم) مانند حالت دو بعدی که نقاط را روی خط تصویر می کردیم، با  $O(n)$  نقاط را روی صفحه انتخاب شده تصویر می کنیم. تا اینجا هزینه  $T(n) = 2(T(n/2) + O(n))$  شده، درنتجه  $T(n) = 2^{\log_2 n} O(n) = O(n \log n)$  حل مساله در دو بعد است را برای آنها انجام داد. در واقع:

$$T(n, 3) = 2(T(n/2, 3)) + T(n, 2) + O(n)$$

که میدانیم  $T(n, 2) = O(n \log n)$  است.

برای محاسبه  $T(n, d)$  از استقرار استفاده می کنیم و فرض می کنیم  $T(n, d-1) = n(\log n)^{d-2}$  باشد. با استفاده از قضیه اصلی داریم:

$$T(n, d) = 2T(n/2, d) + n(\log n)^{d-2} + O(n) = n(\log n)^{d-1}$$

تفاوت سه بعدی و چهار بعدی بودن مسئله هم در زمان اجرا مشخص است.

۸.۷.۴ دو عدد  $n$  بستی A و B داریم. به دلیل بزرگ بودن آنها جمع این دو عدد در  $O(1)$  توسط پردازنده امکان پذیر نیست و ام اعمال جمع و تفریق این دو عدد  $O(n)$  زمان زم دارد. ابتدا محاسبه کنید که ضرب این دو عدد چه مقدار زمان نیاز دارد و سپس

الگوریتمی با مرتبه زمانی  $O(n^{1.5})$  برای ضرب این دو عدد ارائه دهد.

راه حل:

ضرب دو عدد در حالت عادی  $O(n^2)$  است. هر کدام از بسته های عدد A را در عدد B ضرب می کنیم و این  $n$  عدد بدست آمده را (که عدد  $i$  تا شifted داده شده) با هم جمع می کنیم که  $n^2$  مشود.

هر کدام از دو عدد را از وسط به دو قسمت مساوی کم ارزش و پر ارزش تقسیم می‌کنیم، در این صورت:

$$A = A1 * a + A0$$

$$B = B1 * b + B0$$

که  $a$  و  $b$  برابر ۲ به توان  $\text{len}(A0)$  هستند. (وظیفه‌ی شفت دادن بیت‌های  $A1$  و  $B1$  را دارند تا ارزش انها حفظ شود) سپس حاصل  $A * B$  را منویسیم:

$$A * B = (A1 * a + A0) * (B1 * b + B0) = A1 * B1 * a^2 + (A1 * B0 + A0 * B1) * a + A0 * B0$$

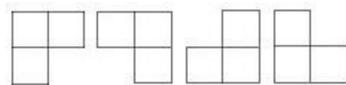
که  $4$  ضرب اعداد  $n/2$  بیتی دارد و زمان الگوریتم  $T(n) = 4T(n/2)$  است و همان  $n^2$  می‌شود. ولی متوجه همن عبارت را با  $3$  ضرب حساب کرد که زمان مورد نظر سوال بدست باید. دو ضرب اول و آخر انجام مشوند و برای محاسبه  $A1 * B0 + A0 * B1$  به صورت زیر عمل می‌کنیم:

$$(A1 * B0 + A0 * B1) = (A1 + A0)(B1 + B0) - A1 * B1 - A0 * B0$$

که  $A1 * B1$  و  $A0 * B0$  یکبار حساب شده‌اند و نیاز نیست دوباره محاسبه شوند. در نتیجه در کل سه ضرب انجام مشوند و تعداد کمی هم جمع داریم که  $O(n)$  هستند. زمان الگوریتم از رابطه  $T(n) = 3T(n/2) + cn$  بدست می‌آید که با توجه به قضیه اصلی محاسبه شده و  $O(n^{\log_2 3})$  می‌باشد.

---

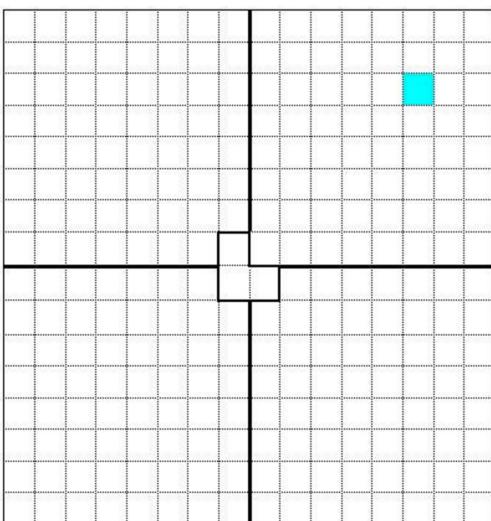
۸.۷.۵ یکی از مسائل جالب طراحی الگوریتم مسئله‌ی کاشکاری یا فرش کردن زمین با موزاییک است فرض کنید قطعه زمین مربعی شکل با ابعادی از توان عدد دو داریم. مثلاً با ابعاد ۱۶ متر هدف فرش کردن این قطعه زمین با استفاده از موزاییک‌هایی با شکل‌های زیر است:



به قسمی که یکی از خانه‌های زمین شطرنجی شده‌ی فوق پوشیده نشود. می‌توان فرض کرد از این خانه برای احداث با غچه یا حوضچه‌ی کوچکی استفاده خواهد شد. توجه داشته باشید که اندازه‌ی اضلاع مربع‌های کوچک موزاییک‌ها نز همانند صفحه‌ی شطرنجی فوق یک متر است. در ضمن حق شکستن این موزاییک‌ها به تکه‌های کوچک‌تر را نداریم.

راه حل:

با توجه به اینکه ابعاد زمان توانی از عدد دو است، روش تقسیم و حل را برای پوشاندن این قطعه زمان امتحان می‌کنیم. بر اساس تعریف روش تقسیم و حل، باید بتوان مسئله را به زیرمسئلی از نوع خود مسئله تقسیم کرد و با ادغام نتایج حاصل از آنها، به نتجه‌ی اصلی دست پیدا کرد. چون ابعاد این قطعه زمان توانی از عدد دو است، پس می‌توان آن را به دو قسمت تقسیم کرد. با تقسیم طول و عرض زمان به دو قسمت، چهار قطعه زمان کوچکتر به دست می‌آید. اما همه‌ی این چهار قطعه زمان شرایط مسئله‌ی اصلی را دارا نستند. در صورت مسئله عنوان شده است که باید از پوشاندن یکی از خانه‌های قطعه زمان صرف نظر کرد. از چهار قطعه زمان کوچکتر تنها یکی از آنها این شرط را برآورده می‌کند و بقیه‌ی قطعه زمان‌ها باید به طور کامل پوشانده شوند. این نقص را می‌توان با به کار بردن یک موزاییک حل کرد:



این موزاییک در مرکز قطعه زمان به قسمی قرار داده شده است که هر کدام از سه قسمت مربعی شکل آن، داخل یکی از قطعه زمان‌های کوچکتری قرار گرفته است که شرط مسئله را برآورده نمی‌کردد. با این کار، داخل این قطعه زمان‌ها نز خانه‌ای وجود دارد که باید پوشانده شود. چرا که از قبل توسط موزاییکی پوشیده شده است. به این ترتیب همه‌ی چهار قطعه زمان کوچکتر خانه‌ای دارند که نیاز به پوشش ندارد. در نتجه می‌توان بر روی حل مسئله در قطعه زمان‌های کوچکتر تمرکز کرد

**۸.۷.۶** دو ماتریس  $A$  و  $B$  داریم. می‌دانیم ضرب آنها در حالت عادی زمان  $O(n^3)$  نیاز دارد. الگوریتمی ارائه دهید که ضرب این دو ماتریس را با زمان  $O(n^{\log_2 7})$  دهد.

راه حل:

هر کدام از دو ماتریس را به  $4 \times n/2$  قسمت  $n/2 \times n/2$  در  $n/2 \times n/2$  تقریب می‌کنیم. در این صورت ضرب دو ماتریس به شکل زیر می‌شود:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}_A \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}_B = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}_C$$

که برای محاسبه‌ی حاصل ضرب نیاز به محاسبه‌ی  $\wedge$  ضرب است و جمع‌هایی که در مجموع اندازه‌ی جمع دو ماتریس  $n \times n$  است. طبق قضه‌ی اصلی اردر این روش  $O(n^2) = O(n^2) + O(n^2)$  برای جمع زدن) که برابر است با  $O(n^3)$

در نتیجه اینجا هم نیاز است تا تعداد ضرب‌ها را کاهش دهیم. (از زمان خواسته شده هم پیداست که باید به ۷ ضرب با اندازه‌ی  $n/2 \times n/2$  برسیم!) حاصل‌های  $r_1$  تا  $r_7$  را (که هر کدام یک ضرب دارند) به این صورت تعریف می‌کنیم:

$$r_1 = a(f-h)$$

$$r_2 = (a+b)h$$

$$r_3 = (c+d)e$$

$$r_4 = d(g-e)$$

$$r_5 = (a+d)(e+h)$$

$$r_6 = (b-d)(g+h)$$

$$r_7 = (a-c)(e+f)$$

در نتیجه داریم :

$$c_{11} = ae + bg = r_5 + r_4 - r_2 + r_1$$

$$c_{12} = ce + dg = r_3 + r_4$$

$$c_{21} = af + bh = r_1 + r_2$$

$$c_{22} = cf + dh = r_1 + r_5 - r_3 - r_7$$

---

زمان هم با قضه‌ی اصلی از رابطه‌ی  $T(n) = 7T(n/2) + O(n^2)$  بدست می‌آید.

**۸.۷.۷ آرایه‌ای به طول  $n$  داریم. با زمان  $O(n)$  عددی را پیدا کنید که بیش از  $n/2$  بار در آرایه تکرار شده است.**

راه حل:

هربار با پیدا کردن یک pivot به اندازه‌ی مانگن اعداد آرایه را به دو قسمت تقسیم می‌کنیم. با  $O(n)$  روشی داریم که اعداد بزرگتر از pivot انتخاب شده بعد از آن و اعداد کوچکتر یا مساوی قبل از آن قرار بگیرند. مث در مرحله‌ی اول واضح است مسئله را باید در قسمت

بزرگتر ادامه دهیم. در نهایت به دنبال حالتی هستیم که آرایه به طول  $k$  به دو زیرمسئله به اندازه های  $0$  و  $k$  تقسیم شود (که یعنی همه ای اعداد برابرند) و  $n/k >= 2$  باشد. تحلیل زمان هم مشابه مسئله quicksort انجام می شود.

---

۸.۷.۸ فرض کنید یک درخت جستجوی دودویی روی اعداد  $a_1, a_2, \dots, a_n$  داده شده. الگوریتمی طراحی کنید که با استفاده از این درخت، دنباله ای مرتب شده ای این اعداد را به عنوان خروجی تولید کند. نشان دهید زمان اجرای الگوریتم شما  $O(n)$  است. توجه کنید که درخت ورودی لزوماً متوازن نیست. (راهنمایی: درخت چپ و راست را به شکل بازگشتی پردازش کنید و خطی بودن زمان اجرا را با استقرار نشان دهید. اگر غیربازگشتی ترجیح مسدهید، طراحی الگوریتم خطی غیربازگشتی نیز ساده است.)

راه حل:

میدانیم که با پیمایش DFS روی BST به آرایه مرتب مرسیم، برای این کار نیز درخت را به دو زیر درخت چپ و راست تقسیم میکنیم و مساله را روی هر کدام از زیر درخت ها حل میکنیم.

inorder (root)

{

if (root == null)

    print root

    inorder (root -> left)

    print root

    inorder (root -> right)

}

چون در این الگوریتم هر گره دققاً یکبار دیده مشود هزینه الگوریتم  $O(n)$  مشود.

---

1. گراف  $G$  داده شده است، تعداد کوتاه‌ترین مسیرهای موجود در این گراف را بیابید.

2. الگوریتم فلويد-وارشال را به گونه‌ای تغییر دهید و روشی ارائه کنید که پس از اجرای فلويد-وارشال بتوان علاوه بر طول کوتاه‌ترین مسیر بین رئوس  $v$  و  $u$  خود مسیر را هم پیدا کرد. زمان اجرای این الگوریتم را تحلیل کنید.

3. تعریف ضرب دو ماتریس  $C = AB$  به شکل زیر است:

$$C_{ij} = \sum_{k=1}^m A_{ik}B_{kj}$$

- گراف بدون وزن و بدون جهت  $G$  با ماتریس مجاورت  $M$  را در نظر بگیرید، عدد واقع در  $(M^K)_{ij}$  برابر با چه مقداری است؟
- آیا می‌توانید تعریف ضرب ماتریس را برای ماتریسهای مربعی به گونه‌ای تغییر دهید که اگر  $W$  ماتریس وزنهای یک گراف وزن‌دار چهتدار باشد هد واقع در  $(W^n)_{ij}$  طول کوتاه‌ترین مسیر از راس  $i$  به راس  $j$  باشد؟
- فرض کنید الگوریتمی داریم که حاصل ضرب  $t(n)$  باز تعریف شده در قسمت قبل را در زمان  $\theta(t(n)\log n)$  حساب می‌کند. الگوریتمی با زمان اجرای بین تمام رئوس یک گراف ارائه دهید.

4. گراف  $G$  داده شده است، با استفاده از الگوریتم فلوید-وارشال وجود دور منفی در  $G$  را بررسی کنید.
5. گراف  $G$  داده شده است، طول کوتاه‌ترین دوری در  $G$  که تعدادی فرد یال دارد را بیابید.

۱. الگوریتم فلوید-وارشال را اینگونه تغییر میدهیم: در هر گام الگوریتم علاوه بر طول کوتاهترین مسیر پیدا شده تعداد مسیرهای با این طول را نیز نگه میداریم (در ماتریس  $\text{count}$ ). برای گام به روزرسانی اینگونه عمل میکنیم: اگر طول مسیری که از  $a$  به  $b$  از طریق  $v$  میرود کوتاهتر از مقدار فعلی بود،  $\text{count}[a][b] = \text{count}[a][v] \times \text{count}[v][b]$  قرار میدهیم، اگر طول مسیر جدید مساوی مقدار قبلی بود،  $\text{count}[a][b] = \text{count}[a][b] + \text{count}[a][v] \times \text{count}[v][b]$  قرار میدهیم.

۲. الگوریتم فلوید-وارشال را اینگونه تغییر میدهیم: علاوه بر ماتریس طول کوتاهترین مسیر، یک ماتریس دیگر به نام  $\text{parent}[u][v]$  نیز نگه میداریم. محتوای  $\text{parent}[u][v]$  برابر راسی با بیشترین شماره است که کوتاهترین مسیر از  $u$  به  $v$  از آن میگذرد. مقادیر اولیه این ماتریس به شکل زیر است:

$$\text{parent}[u][u] = -1$$

$$\text{parent}[u][v] = u, \text{ if } w[u][v] \neq \text{inf}$$

برای گام به روز رسانی اگر مسیر گذرنده از راس فعلی ( $v$ ) کوتاهتر از مسیر بدست آمده تا این مرحله بود،  $\text{parent}[a][b] = v$  قرار میدهیم. برای بدست آوردن کوتاهترین مسیر از راس  $u$  به  $v$  به صورت بازگشتی عمل میکنیم: ابتدا کوتاهترین مسیر از  $u$  به  $\text{parent}[u][v]$  را طی میکنیم، سپس کوتاهترین مسیر از  $v$  به  $\text{parent}[v][w]$  را طی میکنیم و ... حالت پایانی زمانی است که  $\text{parent}[u][v] = -1$  باشد.

۳. تعریف ضرب دو ماتریس  $C = AB$  به شکل زیر است:

$$C_{ij} = \sum_{k=1}^m A_{ik}B_{kj}$$

. عدد واقع در  $(M^n)_{ij}$  برابر است با تعداد گشتهای به طول  $n$  بین روسوس  $i$  و  $j$ . اثبات را با استقرا انجام میدهیم. درستی پایه استقرا  $1 = n$  واضح است. برای گام استقرا داریم

$$(M^n)_{ij} = \sum_{k=1}^m (M^{n-1})_{ik} M_{kj}$$

يعني مقدار  $(M^n)_{ij}$  برابر است با حاصل جمع تعداد گشتهای به طول  $1 - n$  از راس  $i$  به راس  $k$  ضربدر تعداد گشتهای به طول  $1$  از راس  $k$  به راس  $j$  به ازای تمام  $k$  های ممکن. و این مقدار برابر است با تعداد گشتهای به طول  $n$  بین روسوس  $i$  و  $j$ .

. تعریف زیر را در نظر میگیریم:

$$C_{ij} = \min_{k=1}^m \{A_{ik} + B_{kj}\}$$

ادعا میکنیم طبق این تعریف مقدار موجود در  $(W^m)_{ij}$  شامل طول کوتاهترین گشتی با حداقل  $m$  یال از راس  $i$  به  $j$  است. حکم را به استقرا ثابت میکنیم. برای  $1 = m$  حکم بدیهی است. برای گام استقرا داریم:

$$(W^m)_{ij} = \min_{k=1}^n \{(W^{m-1})_{ik} + W_{kj}\}$$

يعني مقدار  $(W^m)_{ij}$  برابر است با حداقل طول گشتی که از  $i$  با گشتی با حداقل  $m$  یال به یک راس دیگر مثل  $k$  میرود و سپس با یک یال از  $k$  به  $j$  میرود (اگر  $j = k$  مقدار همان طول کوتاهترین گشت به طول حداقل  $1 - m$  یال میشود). مقداری که ذکر برابر با مقدار ادعا شده است.

. کافیست مقدار  $W^{\lceil log n \rceil}$  را حساب کنیم که با  $\lceil log n \rceil$  بار به توان ۲ رساندن مکرر محاسبه میشود.

۴. ادعا میکنیم گراف دور منفی دارد اگر و تنها اگر پس از اجرا الگوریتم فلوید-وارشال اگر مقدار  $distance[v][v]$  به ازای حداقل یک راس منفی باشد. یک سمت ادعا واضح است، در صورتی که چنین راسی موجود باشد گراف دور منفی دارد. از سمت دیگر، یک دور منفی در گراف را در نظر میگیریم. دنباله روسوس موجود در این دور را  $v_1 \dots v_k$  مینامیم به گونه‌ای که  $v_k$  راس با بیشترین شماره است. هنگامی که در زمان اجرای الگوریتم فلوید-وارشال راس میانی برابر  $v_k$  است، مقدار  $distance[v_1][v_{k-1}]$  حداکثر برابر طول مسیر  $v_1 \dots v_{k-1}$  است، سپس با بهروزرسانی از طریق  $v_k$  مقدار  $distance[v][v]$  با  $distance[v][v]$  توجه به منفی بودن طول دور یک عدد منفی میشود.

۵. ابتدا طول کوتاهترین مسیرهای با زوج یال و فرد یال را جفت راس پیدا میکنیم. برای این کار الگوریتم فلوید-وارشال را اینگونه تغییر میدهیم: بجای نگه داشتن یک ماتریس شامل طول کوتاهترین مسیرهای بین روسوس تا مرحله فعلی، یک ماتریس شامل طول کوتاهترین مسیر با زوج یال و یک ماتریس شامل طول کوتاهترین مسیر با فرد یال نگه میداریم، برای اشاره به این دو ماتریس به ترتیب  $sop[u][v]$  و  $sep[u][v]$  را استفاده میکنیم. مقادیر اولیه این دو ماتریس به شکل‌های زیرند:

$$sop[u][v] = w[u][v] \quad , if \ u \neq v, sop[u][u] = \inf$$

$$sep[u][v] = \inf \quad , if \ u \neq v, sep[u][u] = \cdot$$

برای گام بروز رسانی اینگونه عمل میکنیم:

$$sop[a][b] = \min(sop[a][b], sep[a][v] + sop[a][v], sop[a][v] + sep[a][v])$$

$$sep[a][b] = \min(sep[a][b], sep[a][v] + sep[v][b], sop[a][v] + sop[v][b])$$

سپس طول کوتاهترین دور فرد به شکل زیر پیدا میشود:

$$\min_{u,v,w \in V(G)} (sop[u][v] + sop[v][w] + sop[w][u], sep[u][v] + sep[v][w] + sop[w][u])$$

6. موارد موجود در گروه الف را به مورد مرتبط با آنها در گروه ب وصل کنید.

گروه الف)

- (1) الگوریتم دایکسترا
- (2) الگوریتم بلمن فورد
- (3) الگوریتم فلوبید وارشال

گراف ب)

- Dynamic Programming (1
- backtracking(2
- greedy algorithm(3

جواب:

دایکسترا یک الگوریتم حریصانه (greedy algorithm) است.  
بلمن فورد و فلوبید وارشال هر دو Dynamic Programming هستند.

7. یک گراف جهتدار  $G = (V, E)$  بما داده شده که در ان از هر یال  $E \ni (U, V)$  همراه یک مقدار  $R(U, V)$  که یک عدد حقیقی در محدوده  $[1, 0]$  است میباشد این عدد دهنده قابلیت اطمینان از یک کانال ارتباطی از راس  $U$  به  $V$  است با فرض استقلال این احتمالات و اینکه  $R(V, U) = R(U, V)$  احتمال شکست را مشخص کند یک الگوریتم کارآمد برای یافتن قابل اطمینان ترین مسیر بین هر دو راس داده شده چیست؟

برای حل این سوال نیاز داریم مسیری با مینیمم ضرب یالها را بیابیم فرض کنیم  $S$  مبدا و  $T$  راس مقصد باشد

$$P = \arg \max \prod_{i=0}^k r(v_{i-1}, v_i)$$

این سوال به راحتی به یک مسئله کوتاهترین مسیر از یک رأس تبدیل میشود. بیش از ان لازم است وزن هر راس را به  $\log r(u, v)$  تبدیل کنیم از انجا که لگاریتم یکنواختی را تغییر نمیدهد و منفی لگاریتم در این وزنهای جدید باعث میشود مسئله بجای کوچک کردن به بزرگ کردن تبدیل شود در نتیجه

$$P = \arg \min \sum_{i=1}^k w(v_{i-1}, v_i)$$

کم وزنترین مسیر را در گراف اولیه خواهد داد که با استفاده از Dijkstra قابل حل خواهد بود و در  $O(E \log V)$  قابل انجام است

8. راه دیگر برای بازسازی کوتاه ترین مسیر در الگوریتم فلوبید وارشال با استفاده از مقادیر  $\Phi_{ij}^{(k)}$  است برای هر  $i, j, k = 1, 2, 3, \dots, n$  مقدار  $\Phi_{ij}^{(k)}$  بیشترین عدد داده شده بین راسهای میانی کوتاهترین مسیر بین راس او زاست و همه راسها از مجموعه  $\{1, 2, \dots, k\}$  میباشد یک فرمول بازگشتی از فلوبید وارشال ارایه دهید که مقادیر  $\Phi_{ij}^{(k)}$  را محاسبه و چاپ کند و ماتریس  $\Phi_{ij}^{(k)}$  را بعنوان ورودی بگیرد.

ابتدا باید بدانیم که  $\Phi_{ij}^{(k)}$  پدر راس زدر کوتاهترین مسیر از  $i$  تا  $j$  را ذخیره میکند

$$\phi_{ij}^{(k)} = \begin{cases} NIL & \text{if } k = 0 \\ k & \text{if } k > 0 \text{ and } d_{ik}^{k-1} + d_{kj}^{k-1} < d_{ij}^{k-1} \\ \phi_{ij}^{(k)} & \text{if } k > 0 \text{ and } d_{ik}^{k-1} + d_{kj}^{k-1} \geq d_{ij}^{k-1} \end{cases}$$

---

**Algorithm 1:** FLOYD-WARSHALL(W)

---

```

1  n = W.rows;
2  D(0) = W;
3  Φ(0) = NIL;
4  for k = 1 to n do
5      let D(k) = (dij(k)) be a new n × n matrix;
6      let Φ(k) = (φij(k)) be a new n × n matrix;
7      for i = 1 to n do
8          for j = 1 to n do
9              if dik(k-1) + dkj(k-1) < dij(k-1) then
10                 dij(k) = dik(k-1) + dkj(k-1);
11                 φij(k) = k;
12             end
13             else
14                 dij(k) = dij(k-1);
15                 φij(k) = φij(k-1);
16             end
17         end
18     end
19 end

```

---

---

**Algorithm 2: PRINT-ALL-PAIRS-SHORTEST-PATH( $\Phi$ , i, j)**


---

```

1 if  $i == j$  then
2   | print  $i$ ;
3 end
4 else if  $\phi_{ij} == NIL$  then
5   | print "no path from "  $i$  " to "  $j$  " exists".;
6 end
7 else
8   | PRINT-ALL-PAIRS-SHORTEST-PATH( $\Phi$ , i,  $\phi_{ij}$ );
9 end

```

---

9. فرض کنیم  $G = (V, E)$  یک گراف وزن‌دار با تابع وزنهای مثبت باشد به تعریف  $w: E \rightarrow \{1, 2, \dots, w\}$  برای مقادیر مثبت  $w$  است و فرض کنیم هیچ دو راسی کوتاهترین مسیر با وزن یکسانی را تا راس مبدا یندارند. حال فکر کنید گرافی بدون وزن به نام  $(V' \cup V', E')$  را تعریف کنیم که از جا گذاری هر یال  $e \in E$  با  $w(e) = 1$  باشد در مجموعه ریوس بدست می‌باشد  $G'$ . چند راس دارد؟ حال فرض کنید که در سیاه کردن ریوس برایر با اراده دایکسترا در خارج کردن انها از صف اولویت اعمال شده بر  $G'$  است.

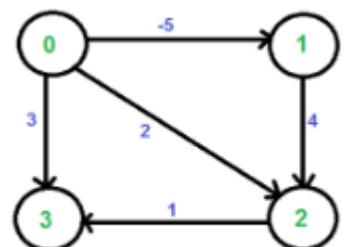
میدانیم  $G'$  دارای مجموعه ای از ریوس  $v$  است برای هر راس  $(u, v)$  در  $G$  به دلیل اینکه یالهای  $(u, v)$  با وزن واحد در  $G'$  وجود دارد در نتیجه راس داخلی در  $G'$  وجود خواهد داشت. در نتیجه مجموع ریوس داخلی در  $G'$   $(u, v)-1$

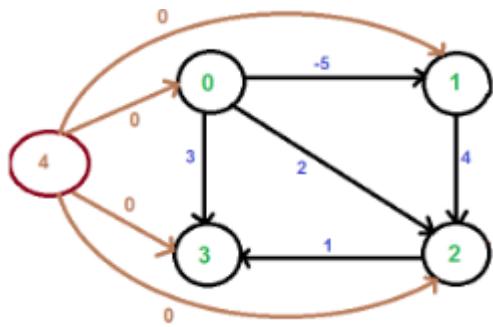
$$|V'| = \sum_{(u,v)} (w(u,v) - 1) = \sum_{(u,v)} w(u,v) - |E|$$

$$|v \cup v'| = |V| + |V'| = \sum_{(u,v) \in E} w(u,v) + |V| - |E|$$

حال فرض کنید  $v$  از صف اولویت اکسیرکت شده صفت  $Q$  باشد. با علم به اینکه هیچ دو راسی وزنهای کوتاهترین مسیرشان یکسان نیست در نتیجه  $d[u] < d[v]$  در  $G'$  بدليل اینکه هر راس  $(u', v')$  در  $G'$  به اندازه  $w(u', v')$  بزرگ شده اند در  $BFS$  ان در مرحله  $discovery$  آن  $u$  ویزیت شده و در مرحله  $d[v]$  ویزیت شده است در نتیجه بعد از  $u$  ویزیت شده بعارت دیگر در  $v$  بعد از  $BFS$   $v$  پس عمق  $v$  از  $u$  بیشتر است که نشان میدهد  $d[v] > d[u]$  در نتیجه در دایکسترا نیز پس از  $u$  خارج می‌شود.

10. الگوریتم جانسون را بر گراف زیر اعمال کنید:





راس 4 را اضافه میکنیم.

.1

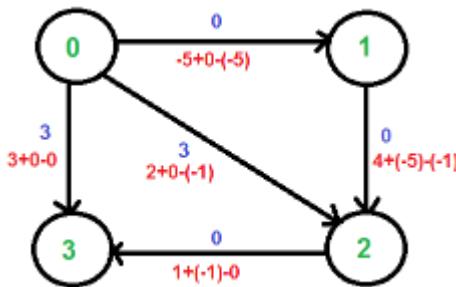
با بلمن فورد کمترین فاصله راس 4 را از همه ریوس حساب میکنیم و همه ریوس را به اپدیت میکنیم سپس روی هر راس دایکسترا اعمال  $w(u, v) = w(u, v) + h[u] - h[v]$ .

میکنیم

هزینه نهایی:

$O(V^2 \log V + VE)$  است.

.2



11. اگر  $G_s$  یک گراف احتمالات باشد محتملترین مسیر بین هر دو راس را با یک الگوریتم تعیین کنید:

1. راههای با وزن صفر به  $G_s$  اضافه میکنیم تا گراف کامل شود

2. ماتریس  $[W_{ij}] = W$  را ماتریس وزن یالها تعیین میکنیم و وزن یالها در ان ذخیره میشود.

3. ماتریس پدر  $[t_{ij}] = T$  را با تعداد یکسان ردیف و ستون میسازیم و به این شکل مقدار دهی میکنیم :

$$t_{ij} = \begin{cases} 0, & i = j \\ 1, & i \neq j \end{cases}$$

هر راس رو اپدیت میکنیم (relax) بشکل زیر:

$$t_{ij} = t_{jk} \text{ انگاه } w_{ij} < w_{ik} \cdot w_{kj}$$

$$w_{ij} = w_{ik} \cdot w_{kj}$$

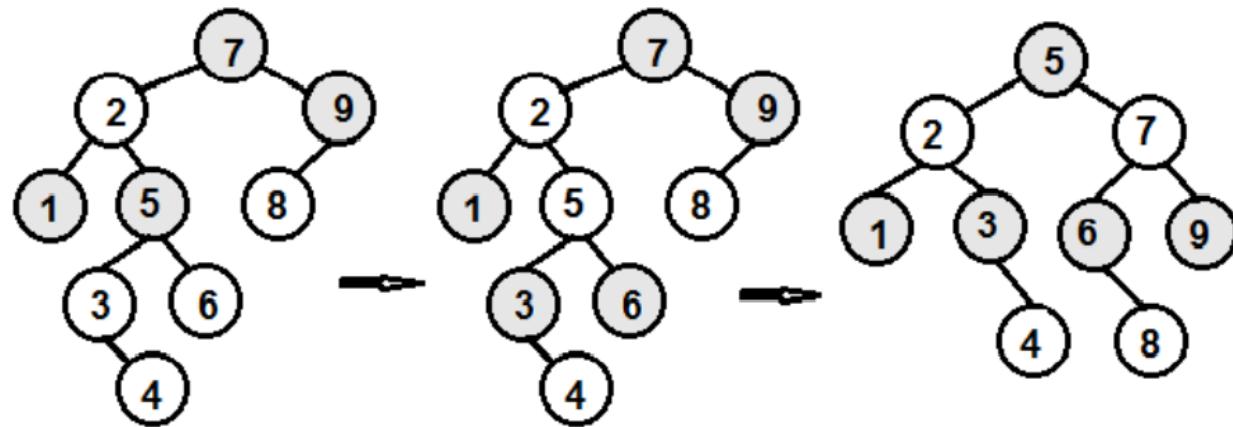
ماتریس نهایی  $W$  ماتریس احتمال از محتملترین دنباله اتفاقات است ( $w_{ij}$  احتمال محتمل ترین دنباله است که از اتفاق  $i$  شروع و به  $j$  ختم میشود)

## نمونه سوالات مربوط به بخش 9.2

9.2.1 عناصر 4,6,3,1,8,1,2,9,7 را به ترتیب در یک درخت قرمز-سیاه تهی درج کنید. (با ذکر مراحل)

راه حل :

3 مرحله‌ی پایانی:



9.2.2 نشان دهید در هر درخت قرمز-سیاه ارتفاع هر راس حداقل 3 برابر کوتاهترین فاصله از این راس به یکی از برگها است.

راه حل :

کوتاهترین فاصله  $x$  تا برگ :  $d(x)$  ، ارتفاع :  $h(x)$

$2*d \geq h(x) \geq d(x)$  در غیر این صورت حداقل 2 راس قرمز مجاور مشوند . می دانیم  $d(x)$  بنابراین  $h(x)/2$

9.2.3 دو درخت قرمز-سیاه با اندازه‌های  $n$  و  $m$  داریم، چگونه این دو درخت را در زمان بهینه با هم ادغام کنیم؟

راه حل :

زمان  $O(m+n)$

اگر درختها را به صورت *inorder* پیمایش کنیم میتوانیم عناصر هر کدام از آنها را به صورت مرتب شده در  $O(2)$  لیست در  $O(m)+O(n)$  به دست آوریم سپس دو لیست به دست آمده را با هم *merge* میکنیم به کمک آنها در  $O(m+n)$  یک درخت جدید بسازیم. روش ساختن درخت جدید :

عنصر میانه لیست را به دست میآوریم و به عنوان ریشه درخت جدید قرار میدهیم و زیردرخت چپ و راست را به همین شکل به صورت بازگشتی به دست میآوریم :  $T(n+m) = O(1) + 2T(n+m/2) \rightarrow \text{overall} : O(n+m)$

---

9.2.4 الگوریتمی بنویسید که یک BST را رنگ کند تا قرمز-سیاه شود یا تشخیص دهد این کار ممکن نیست.

راه حل :

3. For all vertexes like n starting by the root:

```
5.   if n is root,  
6.       color(n) = black  
7.       bh(n)= ⌈ h(n)/2 ⌉  
8.   else if p(n) is red,  
9.       color(n) = black  
10.      bh(n)= bh(p(n)).  
11.  
12.   else (p(n) is black)  
13.       if d(n) < bh(p(n)), then  
14.           error "shortest path was too short"  
15.       else if d(n) = bh(p(n)) then  
16.           Color(n) = black  
17.       else (d(n) > bh(p(n)))  
18.           Color(n) = red  
19.       either way,  
20.           bh(n)= bh(p(n)) - 1
```

n <- سیاه ارتفاع bh(n)

n <- ارتفاع h(n)

n <- والد p(n)

رنگ ریشه را برابر سیاه قرار میدهیم و آن را برابر با نصف ارتفاع آن قرار میدهیم(عنی  $bh$  را برابر حداقل مقدار ممکن قرار میدهیم). به ازای هر راس  $n$  اگر پدر  $n$  قرمز باشد رنگ  $n$  سیاه میشود ( $bh(n)$  برابر با  $bh$  پدرش میشود). اگر پدر  $n$  سیاه باشد:

$bh(n)$  برابر با  $bh(p(n))-1$  میشود. اگر ارتفاع  $n$  از سیاهارتفاع پدرش باشد، نمیتوان درخت را رنگآمیزی کرد. اگر ارتفاع  $n$  دقیقاً برابر سیاهارتفاع پدرش باشد،  $n$  را سیاه میکنیم ( $n$  و تمام رئوسی که در مسیر  $n$  تا برگ قرار دارند باید سیاه شوند). اگر ارتفاع  $n$  بیشتر از سیاهارتفاع پدرش باشد، رنگ آن را قرمز میکنیم.

(برای یک درخت ممکن است چندین رنگآمیزی مختلف وجود داشته باشد)

---

9.2.5 آیا امکان دارد که تمام گره های یک درخت قرمز - سیاه به رنگ سیاه باشد؟

راه حل :

---

9.2.6 درخت قرمز - سیاه ای مثال بزنید که AVL نباشد!

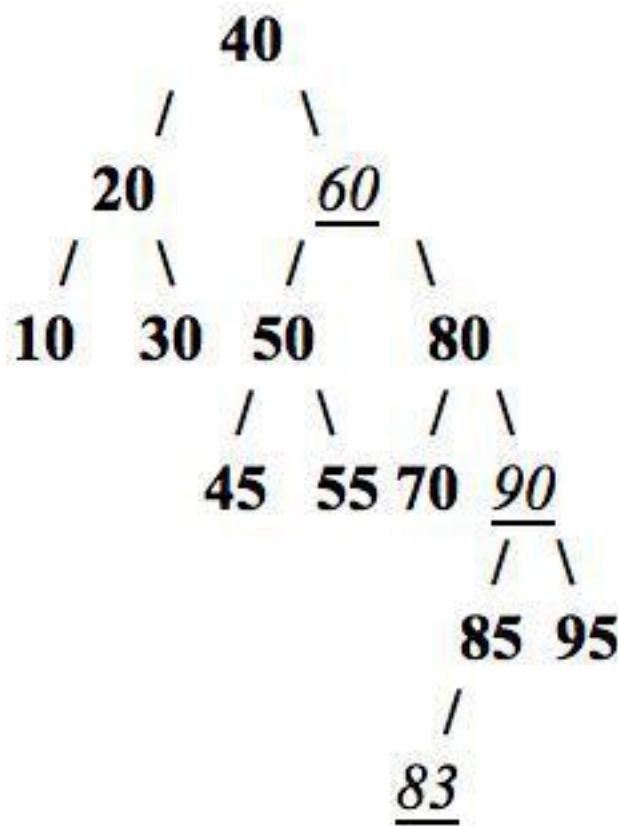
راه حل :

---

9.2.7 حداقل ارتفاع درخت قرمز - سیاه با ۱۴ گره جقدر است؟ مثالی از درختی قرمز - سیاه با ۱۴ گره بکشید که به حداقل ارتفاع رسیده باشد.

راه حل :

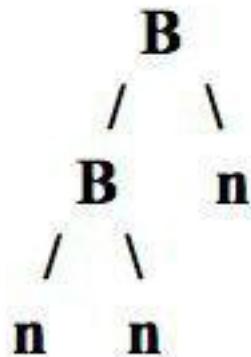
ماکسیمم ارتفاع ۵ است ا توجه به راهنمایی سوال یک درخت با black-height ۲ میتواند دقیقاً ارتفاع ۵ داشته باشد اگر که به صورت تناوبی آن را تا عمیق ترین گره قرمز و سیاه کنیم.



9.2.8 چرا نمیتواند یک درخت قرمز-سیاه ، یک گره مشکی دقیقاً یک فرزند سیاه و هیچ فرزند قرمز داشته باشد؟

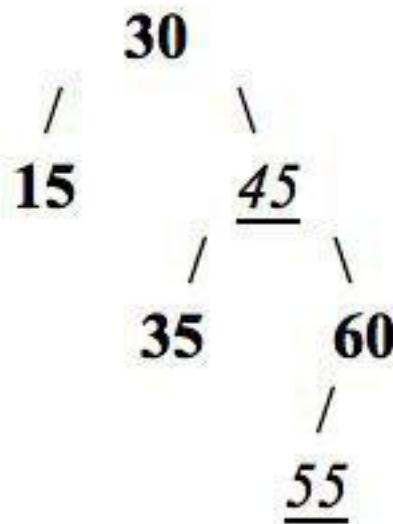
راه حل:

برای اینکه black depth گره های خارجی گره های سیاه نمیتواند برابر black depth بقیه ی گره های خارجی آن گره سیاه باشد با توجه به شکل زیر سیاه ها را با B و گره های خارجی را با n نشان داده شده.



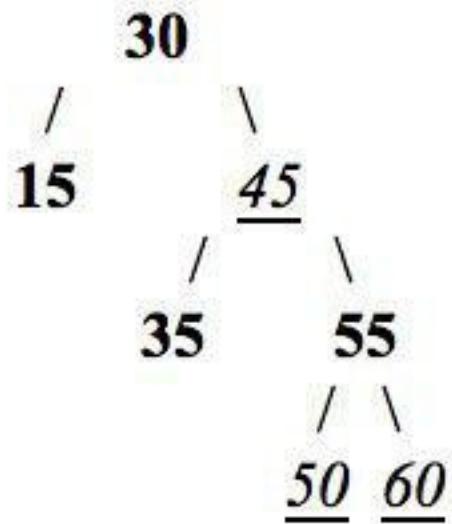
---

9.2.9 عدد ۵۰ را به درخت قرمز-سیاه زیر اضافه کنید .



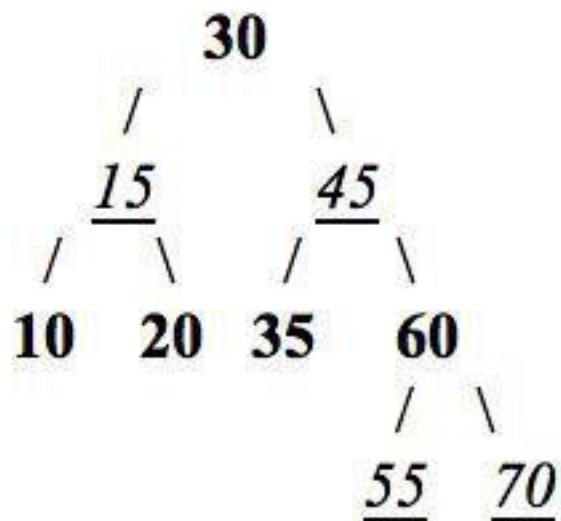
شكل 1

راه حل :



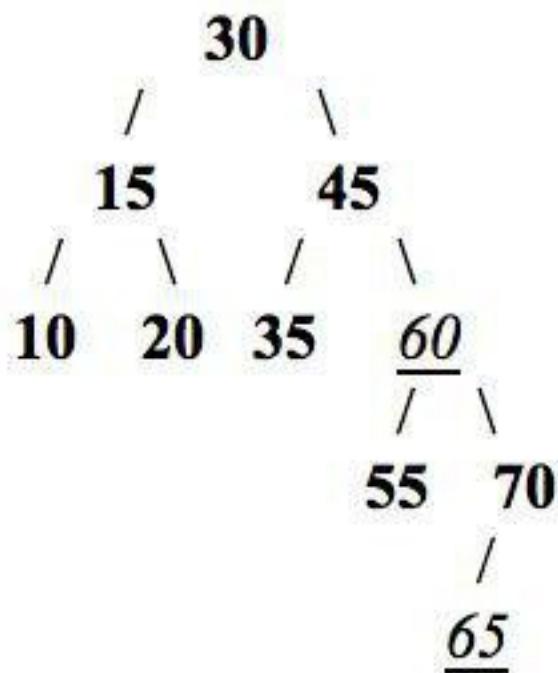
---

9.2.10 عدد 65 را به درخت قرمز-سیاه زیر اضافه کنید.



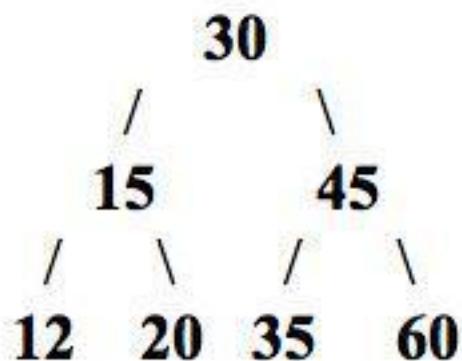
شكل 2

راه حل :



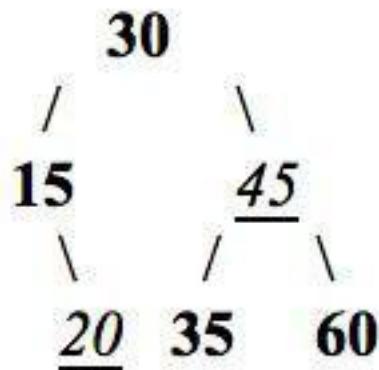
---

9.2.11 عدد ۱۲ را از درخت قرمز سیاه زیر پاک کنید و نتیجه را نشان دهید.



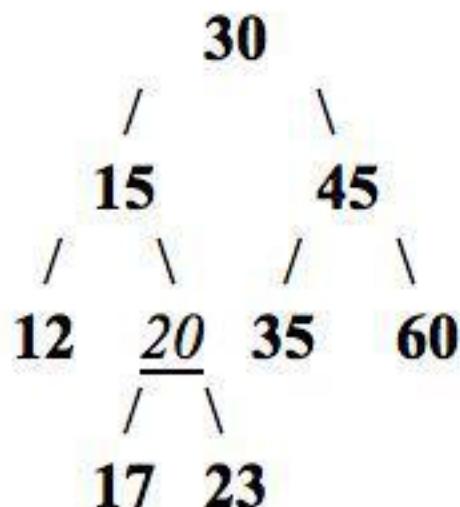
شكل 3

راه حل :



---

9.2.12 عدد ۱۲ را از درخت قرمز سیاه زیر پاک کنید و نتیجه را نشان دهید.



راه حل :

$$\begin{array}{c} 30 \\ / \quad \backslash \\ 20 \qquad 45 \\ / \quad \backslash \quad / \quad \backslash \\ 15 \ 23 \ 35 \ 60 \\ \backslash \\ \underline{17} \end{array}$$

---