محمد ملائی، محسن محمودآبادی، داوود نصرتی امیرآبادی

۱ تمرین

۱.۱ الف

الگوریتمی که در صورت سوال به عنوان نمونه و ورودی داده شده است تنها به یک تغییر جزئی در مقدار خروجی خود دارد تا به جواب بخش الف تمرین دست پیدا کند. در واقع چون الگوریتم اندیس نویسه ی اول رشته را برمی گرداند، با داشتن طول نویسه می توان به آخرین نویسه ی رشته دست پیدا کرد و اندیس آن را به عنوان خروجی بازگرداند.

Algorithm 1 Last-Index

Input: An array T[0...n-1] of n characters representing a text and an array $P[0...m-1]$ representing a pattern

Output: The index of last character in text that ends a matching substring or -1 if the search is unsuccessful

```
for i = 0 to n − m do
    j = 0
    while j < m and P[j] = T[i + j] do
        j = j + 1
    if j = m then
        return i + m − 1
return -1
```

۲.۱ ب

با توجه به خواسته ی مسئله، ابتدا به اندازه ی طول الگو به سمت راست حرکت کرده و سپس یکی یکی به سمت چپ حرکت می کنیم. در پایان در صورت عدم موفقیت آمیز بودن جستجو، از نویسه ی بعدی پیمایش را آغاز می کنیم.

Algorithm 2 RTL-Check

Input: An array T[0...n-1] of n characters representing a text and an array $P[0...m-1]$ representing a pattern

Output: The index of last character in text that ends a matching substring or -1 if the search is unsuccessful

```
for i = 0 to n − m do
    j = m - 1
    while j > 0 and P[j] = T[i + j] do
        j = j - 1
    if j = 0 then
        return i
return -1
```

۳.۱ پ

در اینجا متغیری به نام *count* تعریف می کنیم و بار هر بار شناسایی کامل الگو، یک واحد آن را افزایش می دهیم. در ضمن، پس از یافتن الگو به طور کامل، به اندازه ی طول الگو به سمت جلو حرکت می کنیم و پیمایش را از آنجا آغاز می نماییم تا از یافتن الگوهای دارای اشتراک جلوگیری کرد.

Algorithm 3 Count

Input: An array T[0...n-1] of n characters representing a text and an array $P[0...m-1]$ representing a pattern

Output: Number of pattern occurrences in text

```
i = 0
count = 0
while i < n - m do
    j = m - 1
    while j > 0 and P[j] = T[i + j] do
        j = j - 1
    if j = 0 then
        i = i + m
        count = count + 1
return count
```

۴.۱ ت

با توجه به اینکه کارایی زمانی الگوریتم باید O(n) باشد ، باید نهایتا از یک حلقه استفاده کنیم که تمام کاراکتر های متن را فقط یک بار بررسی می کند. در این الگوریتم، متن و الگو همزمان و با یک متغیر (i) بررسی می شوند و با هر بار مقایسه j افزایش می یابد که نشانگر تطابق آن بخش از رشته و الگوست که رسیدن آن به اندازه الگو به معنی یافتن زیر رشت های یکسان با الگو است. پس از یافتن تطابق مکان آخرین کاراکتر این زیر رشته به عنوان خروجی الگوریتم برگردانده می شود.

Algorithm 4 Linear-Index

Input: An array T[0...n-1] of n characters representing a text and an array $P[0...m-1]$ representing a pattern

Output: The index of last character in text that ends a matching substring or -1 if the search is unsuccessful

```
j = 0
for i = 0 to n do
    if T[i] = P[j] then
        j = j + 1
    else
        j = 0
    if j = m then
        return i
return -1
```

۲ تمرین

۱.۲ الف

از آنجایی که تعداد اعمال انتساب و تقسیم با هم برابر است تعداد تکرار عمل انتساب را در نظر می گیریم برای محاسبات تعداد مراحل انجام شده و از آنجا که $m \ mod \ n$ می تواند حداکثر نصف n باشد و در بدترین حالت کارایی زمانی برابر است با :

$$\gcd(m,n) = \gcd(n, \frac{n}{2}) = \gcd(\frac{n}{2}, \frac{n}{4}) = ... = \gcd(\frac{n}{n^{n-2}}, \frac{n}{n^{n-1}})$$

که $\frac{n}{n^{n-1}} = 1$ و حلقه ی while به پایان می رسد. پس داریم:

$$n-1 = \log_2^n \Rightarrow n = \log_2^n + 1 \Rightarrow \lim_{n\to\infty} \frac{\log_2^n + 1}{\log_2^n} = 1 \Rightarrow \log_2^n + 1 \in O(\log_2^n)$$

۲.۲ ب

برای ثابت کردن این رابطه بازگشتی، ما می‌توانیم از اصل استقرا استفاده کنیم. برای این کار، ابتدا باید ثابت کنیم که این رابطه بازگشتی برای دو عدد صحیح مثبت درست است. سپس، فرض کنید که این رابطه بازگشتی برای $n-1$ عدد صحیح مثبت درست است. بنابراین،

$$\begin{aligned}
\gcd(a_1, a_2, \ldots, a_n) &= \gcd(\gcd(a_1, \ldots, a_{n-1}), gcd(a_n)) && (۱)\\
&= \gcd(\gcd(\gcd(a_1, \ldots, a_{n-2}), a_{n-1}), a_n) && (۲)\\
&= \ldots && (۳)\\
&= \gcd(\gcd(...(\gcd(a_1, a_2), a_3), a_3), \ldots, a_n) && (۴)
\end{aligned}$$

که این رابطه بازگشتی بدون توجه به ترتیب اعداد ورودی، همیشه درست است.
الگوریتم پایین به محاسبه gcd دو عدد می پردازد.

۳.۲ پ

Algorithm 5 gcd(a, b)

Input: a, b are two integers to find gcd
Output: Greatest Common Devisor of a and b

```
while y != 0 do
    x, y = y, (x mod y)
return x
```

الگوریتم پایین با کمک الگوریتمی که در بالا معرفی شد، در هر مرحله در صورت وجود لیستی با طول بیش از ۲، به محاسبه ی gcd <u>عضو اول لیست</u> و <u>باقی اعضای لیست</u> می پردازد و مقدار حاصل را بر می گرداند.

Algorithm 6 super_gcd($A[a_0, a_1, \ldots, a_{n-1}]$)

Input: A: A list of n integers
Output: gcd of all the elements of A

```
if len(a) == 2 then
    return gcd(a[0], a[1])
else
    return gcd(a[0], super_gcd(A[a_1, a_2, ..., a_{n-1}]))
```

با توجه به اینکه کارایی الگوریتم gcd برابر با $\log_2(n)$ است و هربار یک عدد از لیست ما کاسته می شود، و این کار n بار انجام می شود، پس کارایی زمانی الگوریتم ما از مرتبه ی $n\log_2(n)$ می باشد. یا به عبارتی:

$$
\begin{aligned}
M(2) &= \log_2(n) && (۵)\\
M(n) &= && (۶)\\
&= \log_2(n) + M(n-1) && (۷)\\
&= 2\log_2(n) + M(n-2) && (۸)\\
&= i\log_2(n) + M(n-i) && (۹)\\
&= (n-1)log_2(n) && (۱۰)\\
\rightarrow M(n) &\in \Theta(n.log_2(n)) && (۱۱)
\end{aligned}
$$

۳ تمرین

۱.۳ الف

برای محاسبه جواب این مسئله به روش ساده اندیشانه، ابتدا تمام زیر مجموعه های مجموعه S را ساخته و سپس بزرگترین زیرمجموعه ای که در شرط مسئله صدق کند را به عنوان جواب مسئله انتخاب می کنیم یعنی :

$$Space = \{A \in S |\ \nexists 1 \leq j \leq n : A_j \leq A_i \leq F_j\}$$

بزرگترین عضو مجموعه Space از نظر تعداد عضو، جواب مسئله خواهد بود.

Algorithm 7 GreatestSubset

Input: Sets S = $\{s_1, s_2, ..., s_n\}$ and F = $\{f_1, f_2, .., f_n\}$
Output: Set A = $\{a_1, a_2, ..., a_m\}$ which is a subset of S

```
A = {}
for X = {x_1, x_2, .., x_q} in Subsets of {1, 2, ..., n} do
    if q > m then
        iEqual = True
        for i = 1 to q do
            jEqual = True
            for j = 1 to q do
                if x_j ≠ x_i And s_{x_j} ≤ s_{x_i} ≤ f_{x_j} then
                    jEqual = False
                    break
            if jEqual = False then
                iEqual = False
                break
        if iEqual = True then
            A = X
return A
```

۲.۳ ب

برای پیدا کردن جواب این مسئله به 2^n زیر مجموعه از S نیازمندیم که برای بررسی شرط مسئله برای هرکدام از آنها، دو حلقه For که هر یک سه مقایسه انجام می دهند. بنابراین با فرض اینکه هر کارایی زمانی ساخت هر زیرمجموعه $\Theta(1)$ باشد داریم :

$$M(n) = \sum_{i=1}^{n} \binom{n}{i} \sum_{j=1}^{i} \sum_{k=1}^{n} 3 \quad (۱۲)$$

$$= \sum_{i=1}^{n} \binom{n}{i} \sum_{j=1}^{i} 3n \quad (۱۳)$$

$$= \sum_{i=1}^{n} \binom{n}{i} 3(i)(n) \quad (۱۴)$$

$$= 3n \sum_{i=1}^{n} (i) \binom{n}{i} \quad (۱۵)$$

$$= 3n^2 * 2^{n-1} \quad (۱۶)$$

$$\Rightarrow M(n) \in O(n^2 * 2^n) \quad (۱۷)$$

۴ تمرین

۱.۴ الف - ساختار داده

با اجرای الگوریتم و دادن دنباله به عنوان ورودی تابع get_data_structe ، خروجی یک ماتریس $n \times n$ خواهد بود که درایه ij آن کوچکترین مقدار محدوده بین i و j خواهد بود.

```python
def get_min_of_slice(list, start, end):
    minimum = list[start]
    for k in range(start, end + 1):
        minimum = list[k] if list[k] < minimum else minimum
    return minimum

def get_data_structure(list):
    list_size = len(list)
    Matrix = [[0] * list_size for _ in range(list_size)]
    for i in range(0, len(list)):
        for j in range(i + 1):
            min_ij = get_min(list, j, i)
            Matrix[j][i] = min_ij
            Matrix[i][j] = min_ij
    return Matrix
```

Listing 1: Python Implementation

۲.۴ ب - تحلیل کارایی

M(n) را تعداد مقایسه های الگوریتم در نظر می گیریم و داریم :

$$
\begin{aligned}
M(n) &= \sum_{i=0}^{n}\sum_{j=0}^{i}\sum_{k=j}^{i} 1 && (۱۸)\\
&= \sum_{i=0}^{n}\sum_{j=0}^{i} i-j+1 && (۱۹)\\
&= \sum_{i=0}^{n}\left(\sum_{j=0}^{i} i - \sum_{j=0}^{i} j + \sum_{j=0}^{i} 1\right) && (۲۰)\\
&= \sum_{i=0}^{n}\left(i^2 - \frac{i(i+1)}{2} + (i+1)\right) && (۲۱)\\
&= \sum_{i=0}^{n} \frac{1}{2}i^2 + \frac{1}{2}i + 1 && (۲۲)\\
&= \frac{1}{2}\sum_{i=0}^{n} i^2 + \frac{1}{2}\sum_{i=0}^{n} i + \sum_{i=0}^{n} 1 && (۲۳)\\
&= \frac{1}{12}n(n+1)(2n+1) + \frac{1}{2}n(n+1) + n + 1 && (۲۴)\\
&= \frac{1}{6}n^3 + \frac{3}{4}n^2 + \frac{19}{12}n + 1 && (۲۵)\\
&\Rightarrow M(n) \in \Theta(n^3) && (۲۶)
\end{aligned}
$$

۵ تمرین

الگوریتم را به سه بخش تقسیم می‌کنیم. ابتدا لیست m تایی ورودی را به لیست مجاورت تبدیل کرده و سپس به جستجوی عمیق در گراف پرداخته، در نهایت با اجرای پیمایش عمیق به یافتن جواب می‌پردازیم.

برای ساخت لیست مجاورت، یک آرایه‌ی n عضوی می‌سازیم. در اندیس 0 ام آرایه، آرایه هایی به شکل $[v, e]$ قرار می‌دهیم که در آن v نشان دهنده کامپیوتری است که کامپیوتر صفرم به آن متصل است و e زمانی است که در آن، کامپیوتر صفرم به کامپیوتر v ام متصل می‌شود.

برای انجام این کار، در لیست ورودی پیمایش می‌کنیم.

Algorithm 8 graph_adjacency(connections)

Input: a connection list, containing m lists of $[C_i,\ C_j,\ t_k]$
Output: An Adjacency list of the graph

```
m = len(connections)
adjacency = [[ ] for _ in range(m)]
for C_i, C_j, t_k in graph do
    adjacency[C_i - 1].append([C_j − 1, t_k])
    adjacency[C_j - 1].append([C_i − 1, t_k])
return adjacency
```

الگوریتم زیر، نحوه‌ی انجام پیمایش عمیق در لیست مجاورت را نشان می‌دهد. لازم به ذکر است که در هر مرحله از انجام پیمایش، لازم است تا سه شرط مورد بررسی قرار گیرند.

۱. راسی که قصد مشاهده آن را داریم در لیست راس های آلوده نباشد.

۲. زمان ارتباط با راس بعدی، بیشترمساوی زمان شیوع ویروس و کمترمساوی زمان پایان بررسی باشد.

۳. زمان ارتباط با راس بعدی، حتما بیشترمساوی زمان ارتباط راس قبلی با راس فعلی باشد.

Algorithm 9 dfs(graph, start_node, last_traveled_time, infected_devices)

Input: *graph*: The adjacency list of the graph, *start_node*: The start of search, *last_traveled_time*: The time at which the *start_node* has been infected, *infected_computers*: The list of infected computers by the time of the search
Output: Returns None, but modifies the *infected_computers* list

```
for neighbor in graph[start_node] do
    neighbor_id, neighbor_time = neighbor
    con_1 = neighbor_id not in infected_computers
    con_2 = x ≤ neighbor_time ≤ y
    con_3 = neighbor_time ≥ last_traveled_time
    if con_1 and con_2 and con_3 then
        infected_computers.add(neighbor_id)
        dfs(graph, neighbor_id, neighbor_time)
```

در نهایت با ایجاد یک لیست از کامپیوتر های آلوده، بررسی می‌کنیم که آیا کامپیوتر j ام در لیست وجود دارد یا خیر.

Algorithm 10 is_virus_there$graph, C_i, C_j, x, y$)

Input: *graph*: The adjacency list of the graph, C_i: The first computer infected, C_j: The target computer, x: The time of first infection, y: The end of check time

Output: True if C_j is infected, False otherwise

```
infected_computers = [ ]
dfs(graph, C_i, 0, infected_computers)
C_j in infected_computers
```