تحلیل و طراحی الگوریتم (فاز اول)

نیم سال ۴۰۴۲

دانشکده‌ی مهندسی کامپیوتر – آزاد شیراز

مدرس: دکتر اسکندری

طراحان: رضا رضایی ، نیما سلطانی ، بابک جانفزا

زمان تحویل: ۲۳ اردیبهشت

- پروژه انفرادی ویا نهایتا ۲ نفره است.
- فقط در موارد ذکرشده مجاز به استفاده از کتابخانه‌های آماده هستید.
- هر گروه باید کد پروژه خود را در مهلت اعلام شده ارسال نماید.
- علاوه بر گزارش پروژه، در انتهای ترم یک ارائه و دفاع نیز از این پروژه گرفته خواهد شد. مهلت ارسال پاسخ تا ساعت ۲۳:۵۹ روز مشخص شده است.
- همکاری و هم فکری شما در انجام پروژه مانع ندارد اما پاسخ ارسالی هر گروه حتما باید توسط خود او نوشته شده باشد.
- در صورت هم فکری و یا استفاده از هر منابع خارج درسی، نام هم فکران و آدرس منابع مورد استفاده برای حل سوال مورد نظر را ذکر کنید.

1 مقدمه

در این پروژه به صورت عملی از مفاهیم الگوریتم‌های حریصانه(Greedy) ، برنامه‌نویسی پویا (Dynamic Programming)، الگوریتم‌های گراف و کدگذاری هافمن استفاده می‌شود. سناریوی پروژه برگرفته از مجموعه **فرار از فاکس ریور** است؛ جایی که مایکل اسکوفیلد باید با استفاده از الگوریتم‌های صحیح، مسیر فرار خود را پیدا کند.

شما در **۷ ایستگاه عملیاتی** باید الگوریتم‌های مشخص شده را از صفر (from scratch) پیاده‌سازی کنید.

2 توضیح تکمیلی

این فاز از پروژه را در ۷ مرحله قرار است تکمیل کنید، ولی قبل از هرچیزی ابتدا باید پروژه را از طریق کوئرا دانلود کنید و آن را از حالت فشرده خارج کنید.
به موارد زیر توجه کنید:

- از تغییر دادن نام کلاس ها و همچنین قرار دادن فایل های پروژه خود در دایرکتوری های مختلف خودداری کنید. در انتها این فاز، پروژه شما با اجرای کد autograder نمره دهی خواهد شد و این کد از نام کلاس های مشخص شده استفاده می‌کند و همچنین فرض می‌کند که تمام فایل ها در کنار این فایل قرار گرفته‌اند.
- سه دسته تابع در کدها وجود دارد:

  - دسته‌ی اول تابع هایی هستند که با کامنت TODO مشخص شده‌اند. این قسمت ها باید توسط شما تکمیل شوند.
  - دسته‌ی دوم تابع هایی هستند که با کامنت PROVIDED (keep) مشخص شده‌اند. این قسمت ها معمولا

helper method هایی هستند که برای اجرای بهتر و راحت تر کد مورد نیاز است. بهتر است از تغییر آن ها خودداری کنید.

  - دسته سوم تابع های هستند که کامنت هایی شبیه DO NOT CHANGE یا leave as-is مشخص شده‌اند.

از تغییر دادن این توابع بپرهیزید چرا که اجرای کد autograder را با مشکل مواجه می‌کند.

- پس از اتمام پروژه حتما کد autograder را خودتان نیز یک بار اجرا کنید تا از درست همه چیز مطمئن شوید. از

تغییر این کد خودداری کنید چرا که برای نمره دهی این کد به صورت بدون تغییر در کنار پروژه شما جایگزین و اجرا می شود.

- شما می‌توانید هر تعدادی helper method که خواستید به کلاس ها اضافه کنید یا هر بخش که DO NOT CHANGE نخورده است را تغییر دهید. فقط در نهایت از درستی عملکرد autograder اطمینان حاصل کنید.

- استفاده از کتابخانه‌های آماده برای پیاده سازی متدها و توابع غیر مجاز است و باید تمام بخش ها را from scratch پیاده سازی کنید.
- توجه کنید که آن چیزی که در نهایت برای ارزیابی پروژه شما بررسی می‌شود، کارکرد آن در بازیابی درست اسناد است؛ به همین علت فارغ از انجام تمام مراحل ذکر شده در ادامه، از بازگرداندن اسناد درست با یک کوئری مشخص اطمینان حاصل کنید.

حال برای تکمیل پروژه کافی است مراحل که در ادامه آورده شده است را به درست و به ترتیب انجام دهید.

3 مرحله صفر –دریافت فایل‌های پروژه

فایل‌های اولیه پروژه شامل:

```
project/
├── coin_change.py
├── matrix_chain.py
├── lis_lcs.py
├── huffman.py
├── graph_paths.py
├── knapsack.py
├── climb_stairs.py
├── autograder.py
└── utils.py (PROVIDED)
```

4 مرحله اول – چالش باج(Coin Change)

🗁 **فایل** coin_change.py :

4.1 greedy_coin_change

پیاده‌سازی روش حریصانه برای کمترین تعداد سکه (سکه‌های استاندارد).

```
def greedy_coin_change(coins: List[int], amount: int) -> Tuple[int, List[int]]:
    """
    TODO: حریصانه روش پیاده‌سازی
    coins: سکه‌ها لیست (شده مرتب نزولی صورت به)
    amount: نظر مورد مبلغ
    return: (شده انتخاب سکه‌های لیست, سکه‌ها تعداد)
    """
```

راهنما:

- از بزرگترین سکه شروع کنید و تا جایی که ممکن است بردارید.
- اگر amount به صفر نرسید، یعنی با این سکه‌ها نمی‌توان ساخت (در سکه‌های استاندارد این اتفاق نمی‌افتد).

4.2 dp_coin_change

پیاده‌سازی الگوریتم Coin Change با برنامه‌نویسی پویا (برای سکه‌های غیراستاندارد).

```
def dp_coin_change(coins: List[int], amount: int) -> int:
    """
    TODO: پیاده‌سازی DP
    return: سکه تعداد کمترین ،صورت این غیر در  -۱
    """
```

راهنما:

- از آرایه dp به طول $amount + 1$ استفاده کنید.
- مقدار اولیه $dp[0] = 0$ و بقیه inf.
- رابطه بازگشتی: $dp[i] = min(dp[i], dp[i - coin] + 1)$

5 مرحله دوم – چالش تخریب (Matrix Chain Multiplication)

🗀 **فایل** matrix_chain.py **:**

```
def matrix_chain_order(dims: List[int]) -> Tuple[int, str]:
    """
    TODO: ماتریس‌ها زنجیره‌ای ضرب پیاده‌سازی
    dims: صورت به ماتریس‌ها ابعاد [p0, p1, p2, ..., pn]
    return: (پرانتزگذاری ترتیب ,ضرب‌ها تعداد کمترین)
    """
```

راهنما:

- تعداد ماتریس‌ها = $len(dims) - 1$
- از دو ماتریس $m[i][j]$ و $s[i][j]$ استفاده کنید.
- رابطه:

$$m[i][j] = min(m[i][k] + m[k+1][j] + dims[i-1] * dims[k] * dims[j])$$

6 مرحله سوم – چالش هزارتو (LIS , LCS)

🗀 **فایل** lis_lcs.py **:**

6.1 longest_increasing_subsequence

```
def longest_increasing_subsequence(arr: List[int]) -> Tuple[int, List[int]]:
    """
    TODO: پیاده‌سازی LIS با DP
    return: (طول, دنباله خود)
    """
```

6.2 longest_common_subsequence

```
def longest_common_subsequence(a: str, b: str) -> int:
    """
    TODO: پیاده‌سازی LCS با DP
    return: طول مشترک زیردنباله طولانی‌ترین
    """
```

راهنما LCS :

- از ماتریس $dp[m+1][n+1]$ استفاده کنید.
- رابطه:

اگر $a[i-1] == b[j-1]$: $dp[i][j] = dp[i-1][j-1] + 1$

در غیر این صورت :

$$dp[i][j] = max(dp[i-1][j], dp[i][j-1])$$

7 مرحله چهارم – چالش پیغام مخفی (Huffman Coding)

🗀 **فایل** huffman.py :

```
def huffman_encode(text: str) -> Tuple[Dict[str, str], str]:
    """
    TODO: هافمن کدگذاری پیاده‌سازی
    return: (کاراکتر دیکشنری -> کد باینری, کد شده رشته)
    """
```

راهنما:

- از min-heap (اولویت) استفاده کنید.
- گره‌ها شامل char, freq, left, right هستند.
- برای تولید کدها از DFS روی درخت استفاده کنید.

8مرحله پنجم – چالش مسیریابی (گراف)

🗀 **فایل** graph_paths.py :

8.1 dijkstra

```
def dijkstra(graph: List[List[Tuple[int, int]]], src: int, dest: int) -> Tuple[int, List[int]]:
    """
    TODO: دیکسترا پیاده‌سازی
    graph: صورت به مجاورت لیست [(همسایه, وزن), ...]
    return: (فاصله کوتاه‌ترین, بهینه مسیر)
    """
```

floyd_warshall 8.2

```
def floyd_warshall(graph: List[List[int]]) -> List[List[int]]:
    """
    TODO: فلوید پیاده‌سازی – وارشال
    graph: وزن ماتریس (یال وجود عدم برای inf)
    return: فاصله‌ها کوتاه‌ترین ماتریس
    """
```

راهنما:

- سه حلقه تو در تو k, i, j :
- $dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])$

9 مرحله ششم – چالش غنیمت (Knapsack)

🗀 **فایل** knapsack.py **:**

fractional_knapsack 9.1

```
def fractional_knapsack(weights: List[int], values: List[int], capacity: int) -> float:
    """
    TODO: حریصانه روش
    return: حمل قابل ارزش حداکثر (اعشاری)
    """
```

zero_one_knapsack 9.2

```
def zero_one_knapsack(weights: List[int], values: List[int], capacity: int) -> int:
    """
    TODO: برای پویا برنامه‌نویسی 0/1 Knapsack
    return: ارزش حداکثر
    """
```

راهنما 1 / 0 :

- $dp[w] = max(dp[w], dp[w - weight] + value)$
- از حلقه معکوس روی ظرفیت استفاده کنید.

10 مرحله هفتم – چالش خروج (پله و فیبونانچی)

🗀 **فایل** climb_stairs.py **:**

```
def climb_stairs(n: int) -> int:
    """
    TODO: تعداد راه‌های رسیدن به پله n (هر پله 1 یا 2 قدم)
    return: تعداد راه‌ها (mod 1e9 + 7)
    """
```

راهنما:

- روابط فیبوناچی $dp[i] = dp[i-1] + dp[i-2]$
- برای حافظه کم، فقط دو متغیر نگه دارید.

11 سوالات نظری (باید در گزارش پاسخ دهید)

۱. در مسئله سکه، چرا روش حریصانه برای سکه‌های استاندارد بهینه است اما برای سکه‌های غیراستاندارد نه؟
۲. چرا الگوریتم هافمن با وجود حریصانه بودن، همیشه جواب بهینه می‌دهد؟
۳. تفاوت اصلی برنامه‌نویسی پویا (DP) و روش حریصانه (Greedy) چیست؟ حداقل دو مثال بزنید.
٤. چرا در Knapsack 1/0 روش حریصانه (بر اساس ارزش به وزن) شکست می‌خورد؟

12 ارزیابی پروژه

در این بخش از پروژه شما باید از درستی عملکرد پروژه خود اطمینان حاصل کنید. برای راحتی شما در ارزیابی درست پروژه کافی است که فایل autograder را اجرا کنید و مطمئن شوید تمام موارد passed می‌شوند و هیچ failed ای اتفاق نمی‌افتد. توجه کنید که این فایل فقط خروجی نهایی شما را نمی‌سنجد بلکه تمام بخش های پروژه را بررسی می‌کند.

نحوه ران کردن در صورتی که از cmd ران می کنید :

```
python autograder.py
```

خروجی مورد انتظار:

```
Testing coin_change... PASSED
Testing matrix_chain... PASSED
Testing lis_lcs... PASSED
Testing huffman... PASSED
Testing graph_paths... PASSED
Testing knapsack... PASSED
Testing climb_stairs... PASSED
All tests passed!
```