تحلیل و طراحی الگوریتم (فاز دوم)

نیم سال ۴۰۴۲

دانشکده‌ی مهندسی کامپیوتر – آزاد شیراز

مدرس: دکتر اسکندری

طراحان:

امیر حسین همتی ، حمید رضا نامجومنش،

ویانا حسین بیگی ، مریم میدانشاهی ،

علی نیکوان ، رضا رضایی ،نیما سلطانی،

بابک جانفزا

زمان تحویل: ۰۷ خرداد

- 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 فاز دوم پروژه **فرار از فاکس ریور**، تیم فرار وارد لایه‌های پیشرفته‌تری از سیستم امنیتی زندان می‌شود. مایکل اسکوفیلد باید مسائل زیر را در **۶ ایستگاه عملیاتی** حل کند:

۱. **بهینه‌سازی ادغام قفل‌های زنجیره‌ای** (Knuth Optimization)

۲. **جستجوی سریع الگو در لاگ‌های امنیتی** (Rabin-Karp)

۳. **یافتن طولانی‌ترین زیردنباله صعودی فرکانس لیزرها** (LISدر O(n log n))

٤. **مسئله کوله‌پشتی با وزن بالا** (FPTAS)

٥. **یافتن گلوگاه شبکه با بیشینه شار و کمینه برش** (Edmonds-Karp)

٦. **محاسبه کوتاه‌ترین مسیر بین تمام جفت‌نقاط** (Floyd-Warshall)

همه پیاده‌سازی‌ها باید از صفر (from scratch) انجام شود، اما استفاده از کتابخانه‌های استاندارد زیر برای اهداف کمکی مجاز است:

توضیح	کاربرد مجاز	کتابخانه
فقط برای مشخص کردن نوع داده‌ها	Type hints (List, Tuple, Dict, etc.)	typing
ساختار داده صف برای پیمایش	deque برای BFS در Edmonds-Karp	collections
فقط برای عملیات ساده	توابع پایه ریاضی (.inf, floor, etc)	math
برای مسائل heap-based	فقط در صورت نیاز و با ذکر دلیل	heapq

2 توضیح تکمیلی

این فاز از پروژه را در ٦ 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 پیاده‌سازی مستقیم الگوریتم‌ها غیر مجاز است و باید تمام بخش‌ها را از صفر (from scratch) پیاده‌سازی کنید. کتابخانه‌های typing، collections.deque و math (تنها برای اهداف کمکی) مجاز هستند.
- 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/
├── knuth_optimization.py
├── rabin_karp.py
├── lis_fast.py
├── fptas_knapsack.py
├── edmonds_karp.py
├── floyd_warshall.py
├── autograder.py
```

└── utils.py (PROVIDED)

4 مرحله اول – چالش قفل‌های زنجیره‌ای(Knuth Optimization)

🗁 **فایل** knuth_optimization.py :

```
def optimal_merge_cost(codes: List[int]) -> int:
    """
    TODO: پیاده‌سازی Knuth Optimization برای بهینه ادغام کدها
    codes: کدها طول لیست (مرتب‌شده)
    return: ادغام کل هزینه کمترین
    """
```

راهنما:

- هزینه ادغام دو کد = مجموع طول آن‌ها.
- از رابطه بازگشتی DP با بهینه‌سازی Knuth (تکیه بر خاصیت quadrangle inequality) استفاده کنید.
- پیچیدگی نهایی: $O(n^2)$ به جای $O(n^3)$.

5مرحله دوم – چالش لاگ‌های امنیتی(Rabin-Karp)

🗁 **فایل** rabin_karp.py :

```
def rabin_karp_search(text: str, pattern: str) -> List[int]:
    """
    TODO: کارپ – رابین الگوریتم با الگو جستجوی
    return: تطابق شروع ایندکس‌های لیست (0 – based)
    """
```

راهنما:

- مقدار base = ۲۵۶، modulus = ۱_۰۰۰_۰۰۰_۰۰۷ (یک عدد اول بزرگ)
- تابع هش رولینگ:

- hash = (hash * base + new_char) % mod
- فقط در صورت برابر بودن هش، مقایسه مستقیم انجام شود.

6 مرحله سوم – چالش سنسورهای لیزری (O(n log n)در LIS)

🗀 **فایل** lis_fast.py **:**

```
def lis_length_and_sequence(arr: List[int]) -> Tuple[int, List[int]]:
    """
    TODO: الگوریتم با صعودی زیردنباله طولانی‌ترین O(n log n)
    return: (دنباله خود ،طول)
    """
```

راهنما :

- از آرایه tails استفاده کنید:
- tails[i] = کوچکترین مقدار پایانی برای یک زیردنباله به طول i+1
- برای بازسازی دنباله، آرایه prev_index نگهداری کنید.

7 مرحله چهارم – چالش کوله‌پشتی ژنراتور (FPTAS)

🗀 **فایل** fptas_knapsack.py **:**

```
def fptas_knapsack(weights: List[int], values: List[int], capacity: int, eps: float) -> int:
    """
    TODO: الگوریتم تقریبی FPTAS برای ۰/۱ knapsack
    eps: مجاز نسبی خطای (۱۰٪ یعنی ۰.۱ مثلاً)
    return: ارزش حداکثر تقریبی مقدار
    """
```

راهنما:

- مقدار K = (eps * max_value) / len(values)
- مقادیر جدید: v'_i = floor(v_i / K)
- حل DP روی مقادیر مقیاس‌شده .
- خطای نهایی ≤ eps * OPT

8مرحله پنجم – چالش گلوگاه شبکه (Edmonds-Karp)

🗀 **فایل** edmonds_karp.py **:**

```
def edmonds_karp(capacity: List[List[int]], source: int, sink: int) -> int:
    """
    TODO: شار بیشینه برای کارپ – ادموندز الگوریتم پیاده‌سازی
    capacity: ظرفیت ماتریس (directed graph)
    return: شار بیشینه مقدار
    """
```

راهنما:

- هر بار با BFS یک مسیر افزایشی پیدا کنید.
- ظرفیت باقیمانده را در residual نگهداری کنید.
- مقدار جریان ارسال‌شده را به مسیر اضافه کنید.
- پیچیدگی: O(V * E²)

9 مرحله ششم – چالش جاده‌های جایگزین (Floyd-Warshall)

🗀 **فایل** floyd_warshall.py **:**

```
def all_pairs_shortest_paths(graph: List[List[float]]) -> List[List[float]]:
    """
    TODO: جفت‌رأس‌ها تمام بین مسیر کوتاه‌ترین برای وارشال – فلوید الگوریتم
    graph: مجاورت ماتریس (یال وجود عدم برای inf)
    return: فاصله‌ها کوتاه‌ترین ماتریس
    """
```

راهنما:

- سه حلقه
- for k in range(n):
- for I in range(n):
- for j in range(n):
- رابطه:
- $dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])$
- اگر $graph[i][j]$تهی باشد -> inf

11 سوالات نظری (باید در گزارش پاسخ دهید)

۱. چرا بهینه‌سازی Knuth باعث کاهش مرتبه زمانی از $O(n^3)$ به $O(n^2)$ می‌شود؟ خاصیت Monge یا quadrangle inequality چه نقشی دارد؟

۲. در الگوریتم رابین-کارپ، انتخاب modulus اول چه تأثیری در کاهش Collision دارد؟

۳. الگوریتم LIS با $O(n \log n)$ چگونه کار می‌کاد؟ تفاوت آن با نسخه $O(n^2)$ چیست؟

٤. FPTAS چه تضمینی برای خطای نسبی ارائه می‌دهد؟ چرا به آن «کاملاً چندجمله‌ای» می‌گویند؟

٥. قضیه بیشینه شار – کمینه برش چه رابطه‌ای با یافتن گلوگاه شبکه دارد؟

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 knuth_optimization... PASSED
Testing rabin_karp... PASSED
Testing lis_fast... PASSED
Testing fptas_knapsack... PASSED
Testing edmonds_karp... PASSED
Testing floyd_warshall... PASSED
All tests passed!
```