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

Week 01 Answers

پاسخنامه تکالیف هفته ۰۱

پاسخ سوال اول (۱۰ امتیاز):

فایل A1.py را اجرا کنید

توضیحات فنی سوال: جستجوی درِ گمشده(Bubble Sort)

این مسئله برای درک عمیق یکی از ابتدایی‌ترین و در عین حال کلیدی‌ترین الگوریتم‌های مرتب‌سازی یعنی Bubble Sort طراحی شده است. در سناریوی کارخانه هیولاها، هدف این است که دانشجو با مفهوم «مقایسه جفت‌به‌جفت» و نحوه بهینه‌سازی یک الگوریتم با مرتبه زمانی O(n^2) آشنا شود.

ساختمان داده‌ها و متغیرهای کلیدی:

- **doors: یک لیست پایتونی حاوی اعداد صحیح که ارتفاع درهای انبار را نشان می‌دهد. پایتون به دلیل ماهیت داینامیک لیست‌ها، اجازه جابجایی (Swap) بسیار سریع را فراهم می‌کند.**
- **swapped (Flag): مهم‌ترین متغیر برای بهینه‌سازی. این متغیر بولی (Boolean) در ابتدای هر گذر (Pass) مقدار False می‌گیرد. اگر در طول یک پیمایش کامل، حتی یک جابجایی رخ دهد، مقدار آن به True تغییر می‌کند.**
- **n - i - 1: این عبارت در حلقه داخلی تعیین‌کننده محدوده مقایسه‌هاست. با توجه به اینکه در هر مرحله بزرگترین عنصر به انتهای لیست منتقل شده است، نیازی به بررسی مجدد عناصر مرتب شده در انتهای لیست نیست.**

منطق الگوریتم و مراحل اجرا:

1. **مقایسه همسایگی : الگوریتم از ابتدای لیست شروع کرده و هر دو عنصر مجاور را با هم مقایسه می‌کند. اگر عنصر فعلی از بعدی بزرگتر باشد، جای آن‌ها عوض می‌شود.**

2. **اثر حبابی : با تکرار این کار، بلندترین در (بزرگترین عدد) در اولین دور به آخرین جایگاه لیست می‌رسد. در دور دوم، دومین عدد بزرگ به جایگاه یکی مانده به آخر می‌رسد و این روند ادامه می‌یابد.**
3. **بهینه‌سازی (Early Exit): اگر در یک دور کامل، هیچ جابجایی صورت نگیرد (یعنی swapped کماکان False بماند)، به این معنی است که لیست از قبل مرتب شده است. در این حالت الگوریتم بلافاصله متوقف می‌شود تا از اجرای تکرارهای بیهوده جلوگیری کند.**

تحلیل پیچیدگی زمانی:

- **Worst Case (بدترین حالت): زمانی که لیست کاملاً معکوس باشد، پیچیدگی O(n^2) خواهد بود.**
- **Best Case (بهترین حالت): به لطف استفاده از پرچم swapped در این کد، اگر لیست از قبل مرتب باشد، پیچیدگی زمانی به O(n) کاهش می‌یابد که یک بهینه‌سازی بسیار مهم است.**

نکات مفید:

استفاده از map : دستور map(int, ...) تابع int را روی تک‌تک اعضای لیست حاصل از split اجرا می‌کند که از نظر عملکردی بسیار بهینه است.

- **جابجایی :(Swap) در پایتون بدون نیاز به متغیر کمکی (temp) و تنها با دستور a, b = b, a می‌توان جای دو عنصر را عوض کرد.**

پاسخ سوال دوم (۵۰ امتیاز):

فایل A2.py را اجرا کنید

توضیح سوال: معمای بیل سایفر(Selection Sort)

این مسئله برای آموزش الگوریتم مرتب‌سازی انتخابی طراحی شده است. برخلاف *Bubble Sort* که عناصر بزرگ را با جابجایی‌های متعدد به انتهای لیست هُل می‌دهد، این الگوریتم تمرکزش بر یافتن (Select)کوچکترین عنصر و قرار دادن آن در ابتدای لیست است.

ساختمان داده‌ها و منطق الگوریتم:

- **دو زیرلیست فرضی :در هر لحظه، لیست ما به دو بخش تقسیم می‌شود:**
  1. **بخش مرتب‌شده (سمت چپ) :که در ابتدای کار خالی است و مرحله‌به‌مرحله پر می‌شود.**
  2. **بخش نامرتب (سمت راست) :که الگوریتم در آن به دنبال عدد مینیمم می‌گردد.**
- **متغیر :min_idx این متغیر کلیدی‌ترین بخش کد است. در هر دور، فرض می‌کنیم اولین عنصرِ بخش نامرتب، کوچکترین است. سپس با پیمایش بقیه لیست، اگر عددی کوچکتر پیدا کردیم، ایندکس آن را در min_idxنگه می‌داریم.**
- **کمترین تعداد جابجایی :ویژگی اصلی Selection Sort این است که در هر دور حلقه اصلی، حداکثر یک Swap انجام می‌دهد. این ویژگی زمانی که هزینه نوشتن در حافظه (Memory Write)بالا باشد، بسیار مفید است.**

تحلیل پیچیدگی زمانی:

- **همیشه O(n^2): این الگوریتم چه لیست مرتب باشد، چه معکوس و چه تصادفی، همیشه باید برای پیدا کردن کمینه، کلِ بخش نامرتب را بگردد. بنابراین بر خلاف Bubble Sort (نسخه بهینه)، این الگوریتم حالت Best Case سریع‌تری ندارد.**

پاسخ سوال سوم (۳۰امتیاز):

فایل A3.py را اجرا کنید

توضیحات سوال: فرار از باغ‌وحش(Counting Sort)

این مسئله برای آموزش الگوریتم مرتب‌سازی شمارشی (Counting Sort) طراحی شده است. برخلاف الگوریتم‌های مرتب‌سازی مبتنی بر مقایسه (مانند Merge Sort یا Bubble Sort)، این الگوریتم از ماهیت عددی داده‌ها استفاده می‌کند. در سوال ماداگاسکار، دو چالش اصلی یعنی **پایداری (Stability)**و **مدیریت اعداد منفی** هدف قرار گرفته‌اند.

ساختمان داده‌ها و متغیرهای کلیدی:

- **min_val و :max_val** برای تعیین محدوده (Range) تغییرات کدهای اولویت استفاده می‌شوند. از آنجایی که در سناریو ذکر شده برخی حیوانات کدهای منفی دارند، پیدا کردن کوچکترین عدد برای محاسبه "جابجایی (Offset) "الزامی است.
- **count_arr(آرایه شمارش) :**یک لیست به طول max - min + 1 . هر ایندکس این آرایه معادل یکی از کدهای اولویت است. به دلیل وجود اعداد منفی، ایندکس i در این آرایه نشان‌دهنده عدد i + min_val در لیست اصلی است.
- **output_arr :** آرایه‌ای هم‌اندازه با ورودی که نتیجه نهایی و مرتب‌شده را در خود جای می‌دهد تا آرایه اصلی دستخوش تغییر ناگهانی نشود.

منطق الگوریتم و مراحل اجرا:

۱. **یافتن بازه :**ابتدا باید بدانیم کدهای اولویت در چه محدوده‌ای هستند. اگر کوچکترین عدد -5 باشد، ما آن را به عنوان مبدأ (صفر) در نظر می‌گیریم.

۲. **شمارش :(Frequency Count)** در این مرحله، تعداد تکرار هر کد اولویت را می‌شماریم.

۳. **تجمعی کردن :(Cumulative Sum)** این مرحله کلیدِ جایگذاری صحیح است. با جمع کردن مقادیر قبلی در آرایه شمارش، متوجه می‌شویم که هر عدد باید در چه بازه ایندکسی از آرایه نهایی قرار بگیرد.

۴. **پیمایش معکوس :(Backward Iteration)** برای حفظ **پایداری** (اینکه حیواناتی با اولویت یکسان، به ترتیب ورودشان در صف باقی بمانند)، لیست اصلی را از انتها به ابتدا می‌پیماییم. این کار باعث می‌شود آخرین تکرار یک عدد، در آخرین جایگاه اختصاص‌یافته به آن در خروجی قرار بگیرد.

چراCounting Sort ؟

- **بهینگی زمانی :**وقتی بازه اعداد (k) نسبت به تعداد آن‌ها (n) کوچک باشد، پیچیدگی زمانی O(n+k)است که از O(n log n) بسیار سریع‌تر است.
- **سناریوی کاربردی :**در سیستم‌های اولویت‌بندی با کدهای محدود (مثل اولویت‌های ۱ تا ۱۰)، این بهترین انتخاب است.

نکات پیاده‌سازی در پایتون:

- استفاده از input().split()به ما این امکان را می‌دهد که تعداد نامشخصی ورودی را به صورت پویا از کاربر دریافت کنیم.

استفاده از **List Comprehension**برای تبدیل سریع رشته‌های ورودی به اعداد صحیح (int)، کد را کوتاه‌تر و خواناتر (Pythonic)می‌کند.

سوال چهارم (۷۰امتیاز): پروتکل "جعبه سیاه" بانکی 💸 🏦

خلاصه پروتکل »جعبه سیاه «بانکی و الگوریتم Bucket Sortبرای تشخیص کلاهبرداری به شرح زیر است:

💡 مفهوم و منطق اصلی

این سیستم تراکنش‌ها را بر اساس شناسه‌های رشته‌ای (مثل "007") در ۱۰ سطل (از ۰ تا ۹) دسته‌بندی کرده و سپس هر سطل را با قوانین متفاوتی مرتب می‌کند.

🛠قوانین توزیع(Bucketing Rules)

برای هر شناسه تراکنش، ابتدا مجموع ارقام آن محاسبه می‌شود:

1. **اگر مجموع ارقام زوج باشد :شماره سطل = رقم اول (از سمت چپ).**
2. **اگر مجموع ارقام فرد باشد :شماره سطل = رقم آخر (از سمت راست).**

⚖️قوانین مرتب‌سازی(Sorting Rules)

پس از توزیع تمام تراکنش‌ها، محتوای هر سطل به ترتیب زیر مرتب می‌شود:

- سطل‌های ۰ تا ۴ (کم‌خطر) :مرتب‌سازی صعودی (Ascending).
- سطل‌های ۵ تا ۹ (پرخطر) :مرتب‌سازی نزولی (Descending).
-
- 💻 نکات فنی و پیاده‌سازی
- حفظ ماهیت رشته‌ای :شناسه‌ها نباید به intتبدیل شوند، چون هویت صفرهای پشت عدد (مثل "040")باید حفظ شود.
- توابع کلیدی:
  - استفاده از ()isdigitبرای محاسبه مجموع ارقام.
  - استفاده از reverse=Trueبرای مرتب‌سازی سطل‌های ۵ تا ۹.
  - استفاده از ()extendبرای ترکیب نهایی سطل‌ها در یک لیست واحد.

تحلیل پیچیدگی

- پیچیدگی زمانی :در حالت میانگین O(n log(n/k)) که بسیار سریع‌تر از الگوریتم‌های سنتی است.
- پیچیدگی مکانی O(n) :جهت ذخیره‌سازی داده‌ها در سطل‌ها.

مثال کوتاه:

ورودی521 :

۱ .مجموع ارقام: 8=1+2+5 (زوج).

۲ .قانون زوج: انتخاب رقم اول ←سطل ۵.

۳ .چون سطل ۵ پرخطر است، با سایر اعضای این سطل به صورت نزولی مرتب می‌شود.

پاسخ سوال پنجم (۱۰۰ امتیاز) :

راهنمای کامل: طراحی سامانه مرتب‌سازی هوشمند تراکنش‌های کلان

Radix Sortچیست؟

فرض کنید می‌خواهید یک دسته کارت با اعداد ۳ رقمی را مرتب کنید. یک راه این است که کارت‌ها را دوتا دوتا مقایسه کنید، اما راه ساده‌تری هم وجود دارد:

مرحله ۱ :کارت‌ها را بر اساس رقم یکان در ۱۰ دسته (۰ تا ۹) قرار دهید.

مرحله ۲ :دسته‌ها را به ترتیب جمع کنید و حالا بر اساس رقم دهگان دسته‌بندی کنید.

مرحله ۳ :دوباره جمع کنید و بر اساس رقم صدگان دسته‌بندی کنید.

تمام! کارت‌ها مرتب شدند بدون اینکه حتی یک‌بار دو عدد را با هم مقایسه کنید.

یک مثال ساده

فرض کنید این اعداد را داریم:

[170, 45, 75, 90, 802, 24, 2, 66]

مرحله ۱ - مرتب‌سازی بر اساس یکان:

رقم ۰[170, 90] :

رقم ۲[802, 2] :

رقم ۴[24] :

رقم ۵[45, 75] :

رقم ۶[66] :

نتیجه[170, 90, 802, 2, 24, 45, 75, 66] :

مرحله ۲ - مرتب‌سازی بر اساس دهگان:

رقم ۰[802, 2] :

رقم ۲[24] :

رقم ۴[45] :

رقم ۶[66] :

رقم ۷[170, 75] :

رقم ۹[90] :

نتیجه[802, 2, 24, 45, 66, 170, 75, 90] :

مرحله ۳ - مرتب‌سازی بر اساس صدگان:

رقم ۰[2, 24, 45, 66, 75, 90] :

رقم ۱[170] :

رقم ۸[802] :

نتیجه[2, 24, 45, 66, 75, 90, 170, 802] :

مرتب شد!

چرا از Counting Sort استفاده می‌کنیم؟

در هر مرحله ازRadix Sort ، ما فقط با ارقام ۰ تا ۹ کار داریم Counting Sort .برای چنین حالتی عالی است چون:

- خیلی سریع است
- ترتیب نسبی اعداد را حفظ می‌کند (پایداری)

پایداری یعنی چه؟

فرض کنید دو عدد ۱۲۱ و ۲۲۱ داریم. هر دو رقم یکان ۱ دارند. اگر در ورودی ۱۲۱ قبل از ۲۲۱ بود، بعد از مرتب‌سازی یکان هم باید ۱۲۱ قبل از ۲۲۱ بماند.

اگر این ترتیب حفظ نشود، مراحل بعدی خراب می‌شوند و جواب نهایی اشتباه درمی‌آید.

مبنا (Base) چیست؟

مبنا تعیین می‌کند هر بار چند رقم را بررسی کنیم:

تعداد دسته‌ها	هر بار چند رقم؟	مبنا
۱۰ دسته	۱ رقم	۱۰
۱۰۰ دسته	۲ رقم	۱۰۰
۱۰۰۰ دسته	۳ رقم	۱۰۰۰

برای اعداد بزرگ‌تر، مبنای بزرگ‌تر انتخاب می‌کنیم تا تعداد مراحل کمتر شود.

سرعت الگوریتم

برای یک میلیون عدد ۱۰ رقمی:

- Quick Sort: حدود ۲۰ میلیون عملیات

- **Radix Sort: حدود ۴ میلیون عملیات**

یعنی Radix Sort حدود ۵ برابر سریع‌تر است!