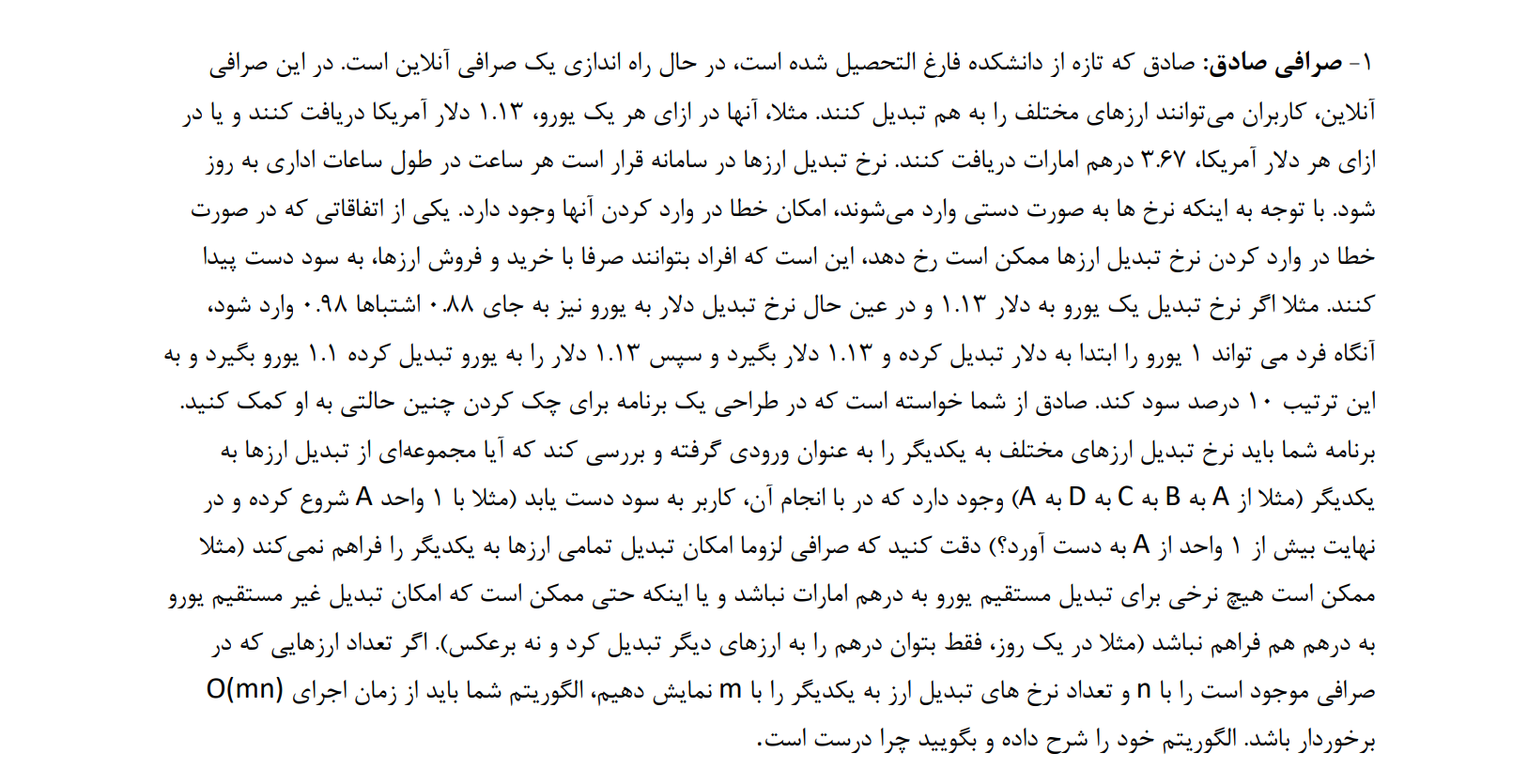
به نام خدا

تمرین چهارم درس ساختمان داده‌ها و الگوریتم‌ها

چمران معینی

۹۹۳۱۰۵۳

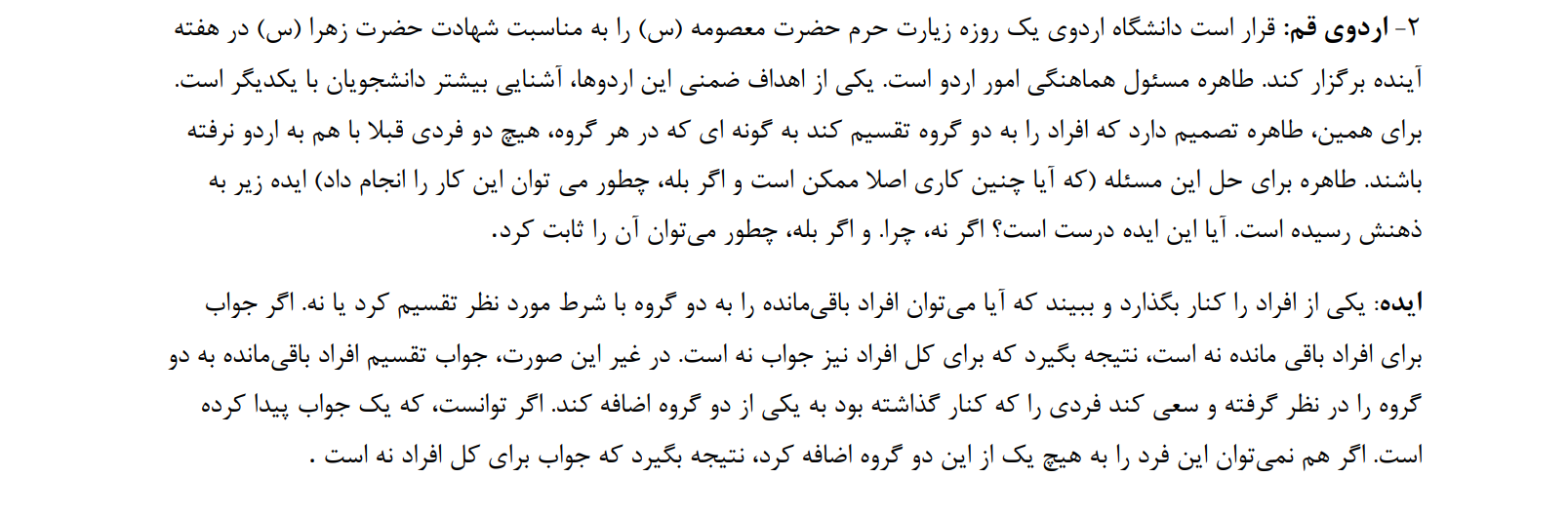


اگر بخواهیم همه‌ی تبدیل‌های مختلف را محاسبه کنیم، بسیاری از محاسبات بارها و بارها تکرار می‌شوند. برای مثال در تبدیل «یورو به درهم به دلار»، و «یورو به درهم به دینار»، و تبدیل «یورو به درهم به ریال»، بارها تبدیل یورو به درهم را محاسبه می‌کنیم، که در مقیاس‌های بزرگ، این تکرارهای زیاد بسیار زمان‌گیر می‌شوند، پس به نظر می‌رسد برنامه‌نویسی پویا می‌تواند در یافتن الگوریتم مناسب، به ما کمک کند.

یک آرایه‌ی دو بعدی به نام dp با سایز n\*n می‌سازیم تا مقادیر محاسبه شده را در آن ذخیره کنیم. خانه‌ی ستون i ام و سطر j ام، نشان‌دهنده‌ی این است که با تبدیل i به j ، چند واحد از آن را خواهیم داشت. پس ابتدا می‌توانیم یکی از قطرها را (که در تمام آن i == j است) را برابر با ۱ قرار دهیم.

گفته شده که m نرخ تبدیل عرض داریم. باید تمام آن‌ها را حداقل یک بار محاسبه و سپس ذخیره کنیم. اگر یک نرخ تبدیل، تبدیل ارز i به ارز j را به ما نشان دهد، در dp[i][j] آن را ذخیره می‌کنیم. سپس به سراغ j ها می‌رویم و هرجا که نرخ تبدیلی بود، دوباره آن را تبدیل می‌کنیم و در سطر مربوط در همان ستون می‌نویسیم. در هر ستون i ، آن‌قدر این کار را ادامه می‌دهیم که به j == i‌ برسیم و اگر نتیجه مقداری غیر از ۱ بود، پس ایرادی در مقادیرمان وجود دارد.

یک راه حل دیگر می‌تواند این باشد که آرایه‌ی dp را m\*m تعریف کنیم و در ستون اول از هر سطر، یکی از تبدیل‌ها را بنویسیم. ایده‌ی کلی این است که در هر یک از سطرها، مسیری که با تبدیلِ نوشته شده در ستون اول شروع شده است را، آن‌قدر ادامه بدهیم که شاید دوباره به همان ارزی که ابتدا داشتیم رسیدیم. اگر مقدار ۱ بود، که یعنی ایرادی در مقادیر داده شده، وجود دارد که باید اصلاح شود، اما اگر تمام سطرها، به ۱ رسیدند، یعنی مقادیرِ داده شده صحیح هستند.



در ایده‌ی مطرح شده، آن‌قدر افراد کنار گذاشته می‌شوند که یا تمام می‌شوند (که یعنی جواب نه است) یا نهایتا تعدادی باقی می‌مانند که می‌توان آن‌ها را در دو گروه قرار داد، و پس از آن یکی یکی افراد کنار گذاشته را بررسی می‌کنیم و به یکی از این دو گروه اضافه می‌کنیم.

تا هنگامی که افرادی که اضافه می‌کنیم، قابلیتِ اضافه شدن را داشته باشند (یعنی گروهی باشد که در آن هیچ آشنایی نداشته باشند) پیش می‌رویم. نهایتا یا همه‌ی افراد گروه بندی می‌شوند، یا این که در این میان به موردی می‌رسیم که در هر دو گروه آشنا دارد، که در این صورت پاسخ نه خواهد بود.

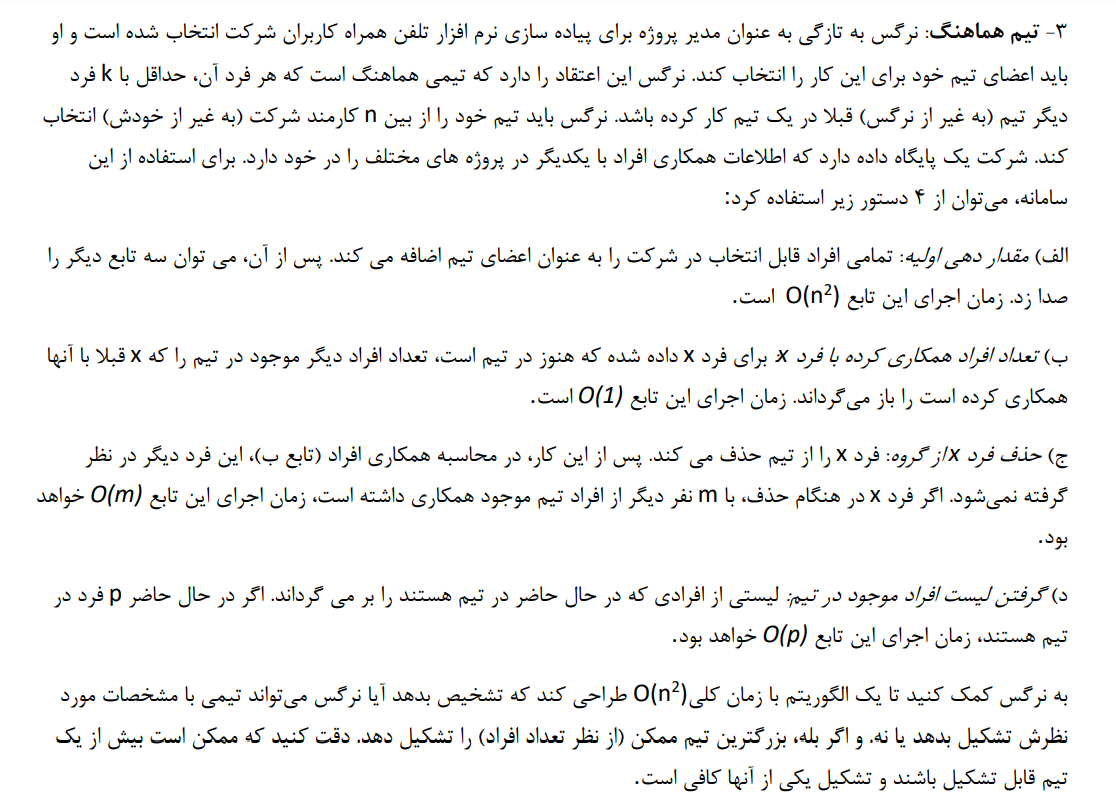
این الگوریتم لزوما درست نیست، برای رد آن یک مثال نقض می‌آوریم.

فرض کنید ما سه نفر داریم، علی و محمد و صادق.

در نظر می‌گیریم که علی با محمد دوست باشد. همچنین علی با صادق دوست باشد. اما محمد و صادق با یکدیگر دوست نیستند.

فرض کنید ما ابتدا علی را کنار بگذاریم، سپس محمد و صادق را به دو گروه تقسیم می‌کنیم. حال آیا می‌توانیم علی را به یکی از گروه‌ها اضافه کنیم؟ خیر، زیرا در هر دو گروه دوستانی دارد. به این ترتیب الگوریتم مطرح شده، پاسخ نه می‌دهد، در حالی که می‌توان «علی» را در یک گروه و «محمد و صادق» را در گروه دیگر قرار داد تا به مطلوبِ سوال برسیم.

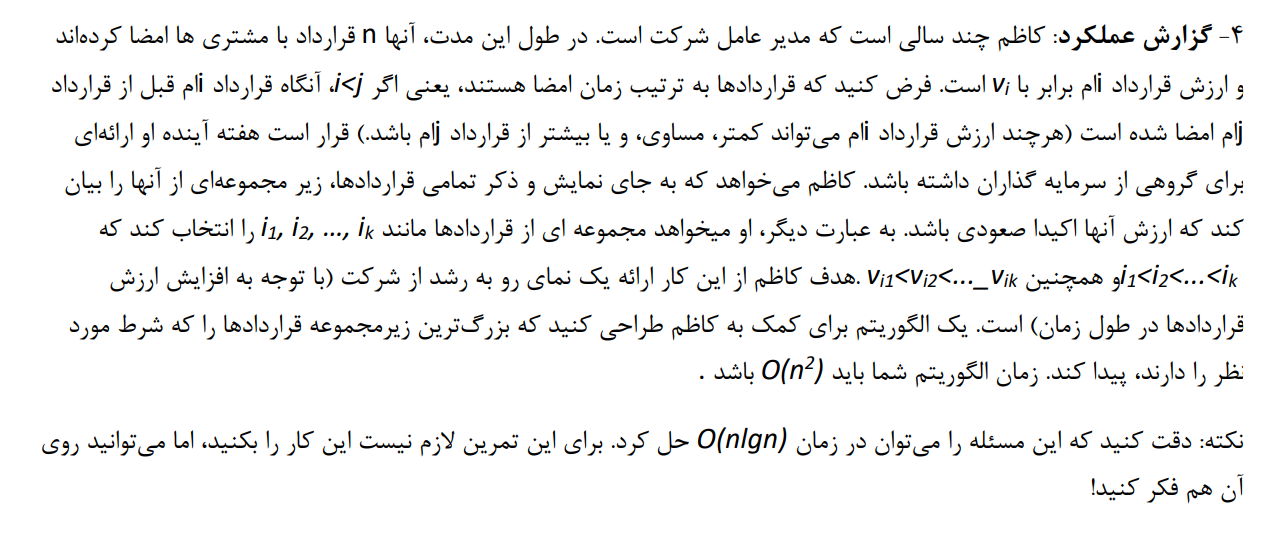
ایده‌ی مطرح شده در صورت سوال، اگر جواب بله برگرداند، جوابش صحیح است، اما ممکن است گاهی که جواب بله است، به اشتباه نه بازگرداند.



ابتدا تابع الف صدا می‌شود.

پس از آن، تابع ب را برای تمام اعضای دیگر صدا می‌کنیم و اگر مقداری که برای هر عضو برمی‌گرداند کمتر از k بود، تابع ج را برای آن فرد اجرا می‌کنیم. سپس تابع د را صدا می‌زنیم و روی لیستی که باز گردانده است، همین فرآیند را دوباره تکرار می‌کنیم.

یا این فرآیند آن‌قدر تکرار می‌شود که دیگر هیچ عضوی در تیم باقی نماند که در این صورت یعنی چنین چیزی ممکن نیست، یا به جایی می‌رسم که هیچ عضوری در طول یک فرآیند حذف نمی‌شود، که یعنی اعضای باقیمانده، بزرگ‌ترین گروه ممکن هستند.



اولین راه حلی که برای حل این سوال به نظر می‌رسد، این است که تمام زیرمجموعه‌های ممکن را بررسی کنیم و اگر صعودی بودند، طول آن‌ها را محاسبه کنیم و سپس از بین نتایج، ماکسیمم را به عنوان نتیجه اعلام کنیم، اما این روش را نمی‌توان در O(n2) انجام داد. با کمی دقت، متوجه می‌شویم که در این روش، بسیاری از مواقع برخی از محاسبات چند بار انجام می‌شوند. برای مثال دنباله‌ی زیر را در نظر بگیرید:

1, 2, 4, 3, 6, 7

در این دنباله، می‌دانیم که طولانی‌ترین دنباله‌ی اکیدا صعودی که از ۱ تا ۴ وجود دارد، به طول ۳ است، اما در محاسبه‌ی طول 1, 2, 4, 6, 7 و 1, 2, 3, 6, 7 ، دوبار این شمارش تکرار شده است. پس از برنامه نویسی پویا کمک می‌گیریم تا برای مثال هرگاه که خواستیم اعداد چهار و چهار به بعد را بررسی کنیم، بدانیم که بلندترین دنباله‌ی اکیدا صعودیِ قبل از ۴ ، به طولِ ۳ است.

می‌دانیم که هر عضو، حداقل زیرمجموعه‌ای اکیدا صعودی به طول یک دارد (مجموعه‌ای که تنها شامل خود عدد باشد) ، پس ابتدا آرایه‌ی dp را به طول n تعریف می‌کنیم و تمام اعضای آن را ۱ می‌گذاریم. قرار است که عضو i ام از این دنباله، نشان‌دهنده‌ی این باشد که طولانی‌ترین دنباله‌ی اکیدا صعودی که به این عضو ختم می‌شود، چه طولی دارد.

این مقدار برای اولین عضو همواره ۱ است، یعنی همیشه dp[0]=1

برای عناصر بعدی، یک واحد به بزرگ‌ترین dp ی قبلی‌شان که مقدار v اش از v عنصر مورد نظر کوچک‌تر باشد، اضافه می‌کنیم. برای ۲ ، این مقدار یکی بیشتر از dp ی ۱ خواهد بود، یعنی همان ۲ . برای ۴ این مقدار یکی بیشتر از دی‌پیِ ۲ خواهد بود، یعنی ۳ ، برای ۳ هم این مقدار یکی بیشتر ای دی‌پیِ یک خواهد بود، یعنی ۳

برای ۶ ، این مقدار برابر دی‌پیِ ۴ به علاوه‌ی یک خواهد بود، یعنی همان ۵ و برای هفت هم دی‌پی۶ به علاوه‌ی یک را خواهیم داشت که برابر ۵ می‌شود.

