حوری دهش 9821413

سوال 1:

برای حل این مسئله می‌توانیم از الگوریتم فلوید-وارشال (Floyd-Warshall) استفاده کنیم. این الگوریتم یک راه حل کامل برای مسئله کوتاه ترین مسیرها در گراف است و برای گراف های با تعداد گره های کم و بزرگ مناسب می باشد.

فرض می کنیم که هر دو اسکله در ابتدا وجود داشته و ما هزینه بین هر دو اسکله را داریم. سپس با استفاده از الگوریتم فلوید-وارشال، برای هر دو اسکله i و j، هزینه کمینه بین آن‌ها را محاسبه می‌کنیم به عبارت دیگر، فاصله کمینه بین هر دو نقطه را برای تمام جفت نقاط محاسبه می‌کنیم. برای این منظور، یک آرایه سه بعدی D به اندازه n × n × n تعریف می‌کنیم که در آن، D[i][j][k] برابر با کمینه هزینه برای رسیدن از اسکله i ام به اسکله j ام با استفاده از اسکله های 1 تا k است. اولین و دومین ابعاد آرایه نشان دهنده دو نقطه‌ی مقصد و مبدا هستند، در حالی که سومین ابعاد آرایه برای فرضیه‌ی پایه الگوریتم فلوید-وارشال به کار رفته است. حال، با استفاده از یک حلقه، مقدارهای آرایه D را به صورت تدریجی به‌روز می‌کنیم. بدین صورت که با شروع از k=1، مقادیر آرایه را به‌روز می‌کنیم، سپس با افزایش مقدار k، برای تمام جفت‌های i و j، هزینه کمینه بین i و j با استفاده از اسکله‌های 1 تا k+1 را محاسبه می‌کنیم و مقدار متناظر در آرایه D را به‌روز می‌کنیم.

در نهایت، مقدار D[1][n][n] برابر با هزینه کمینه برای رسیدن از اسکله 1 ام به اسکله n ام با استفاده از تمامی اسکله هاست.

برای محاسبه D[i][j][k] می توانیم از رابطه زیر استفاده کنیم:

D[i][j][k] = min(D[i][j][k-1], D[i][k][k-1] + D[k][j][k-1])

هزینه مستقیم بین دو اسکله هم میشود:

D[i][j][0] = aij

زمان اجرای این الگوریتم از $O(n^3)$ می باشد.

سوال 2:

برای حل این مسئله، می توانیم از روش برنامه نویسی پویا استفاده کنیم. اگر تعداد تاس ها n و تعداد وجه ها m باشد، می توانیم یک آرایه دو بعدی dp با اندازه (n + 1) × (x + 1) تعریف کنیم که در آن dp[i][j] تعداد حالاتی را نشان می دهد که مجموع تاس ها برابر با j و از i تاس استفاده شده است. برای محاسبه dp[i][j]، می توانیم از فرمول زیر استفاده کنیم:

dp[i][j] = dp[i-1][j-1] + dp[i-1][j-2] + ... + dp[i-1][j-m]

این فرمول به این معنی است که برای محاسبه dp[i][j]، باید تمام حالاتی که از یک تاس کمتر است و مجموع تاس های آنها برابر با j-1، و تمام حالاتی که از دو تاس کمتر است و مجموع تاس های آنها برابر با

j-2، و به همین ترتیب تا حالاتی که از m تاس کمتر هستند و مجموع تاس های آنها برابر با j-m را با هم جمع کنیم.

حالت پایه ما dp[0][0] است که برابر با 1 است، چون برای مجموع صفر تاس، تنها یک حالت وجود دارد (عدم وجود تاس).

در نهایت، تعداد حالات ممکن برای مجموع تاس های برابر با x، برابر است با:

dp[n][x]

زمان اجرای این الگوریتم O(nxm) است.

سوال 3:

برای یافتن اندازه بزرگترین زیرماتریس مربعی که تمام عناصر آن یک هستند می‌توانیم از الگوریتمی با رویکرد برنامه نویسی پویا استفاده کرد. در این الگوریتم از یک ماتریس دو بعدی به نام dp استفاده می‌شود که در آن dp[i][j] بیانگر اندازه بزرگترین زیرماتریس مربعی با سمت بالا و چپ خود در سطر i و ستون j است که تمام عناصر آن یک هستند.

برای محاسبه این ماتریس از رابطه زیر استفاده می‌کنیم:

dp[i][j] = min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]) + 1 if matrix[i][j] == 1 else 0

در این رابطه اگر عنصر ماتریس در مختصات (i, j) برابر یک باشد، اندازه زیرماتریس مربعی با سمت بالا و چپ خود، برابر با حداقل اندازه زیرماتریس مربعی با سمت بالا، سمت چپ و قطر از یک خانه کمتر با برابر با یک خواهد بود. اما اگر عنصر ماتریس در مختصات (i, j) برابر صفر باشد، اندازه زیرماتریس مربعی با سمت بالا و چپ خود برابر صفر خواهد بود.

با پر کردن سطر و ستون اول ماتریس dp به صورت تکیه گاه، می‌توانیم این ماتریس را به صورت تکمیلی پر کرد. سپس بزرگترین عدد در ماتریس dp معادل اندازه بزرگترین زیرماتریس مربعی با تمام عناصر یک است.

زمان اجرای این الگوریتم از $O(n^2)$ می باشد.

سوال 4:

برای پیدا کردن اندیس های s، r، q و p با شرط مشخص شده، می‌توانیم از روش برنامه نویسی پویا با استفاده از یک جدول کمک گرفت. این الگوریتم در زمان خطی O(n) و با استفاده از فضای O(n) انجام می‌شود، که n طول ارایه A است.

فرض می کنیم مقدار max_val در ابتدا برابر A[0] باشد. سپس یک جدول n در 4 را تعریف می‌کنیم به این صورت که در ردیف i، ستون j نشان دهنده اندیسی است که انتهای بازه‌ی مورد نظر برای آن در این ارایه برابر با i است و s، r، q و p نیز به ترتیب با 1، 2، 3 و 4 مشخص می‌شوند.

حال، برای پر کردن جدول، از فرمول زیر استفاده می‌کنیم:

DP[i][j] = max{ DP[i-1][j] , DP[i-1][j-1] + A[i] - A[i-j+1] (j>1) }

در این فرمول، DP[i][j] بیانگر مقدار s-r-q-p مورد نظر با انتهای بازه‌ی s برابر با i و فاصله ی s و r برابر با j است. اگر j برابر با 1 باشد، مقدار DP[i][j] برابر با DP[i-1][j] است چون تنها یک عنصر وجود دارد و در این صورت مقدار A[s] - A[r] + A[q] - A[p] برابر با A[i] - A[i] + A[q] - A[p] می‌شود. در غیر این صورت (j > 1)، مقدار DP[i][j] برابر با بیشترین مقداری است که می توان به ازای آن s، r، q و p را به دست آورد که شرط s > r > q > p را داشته باشند.

در نهایت، اندیس‌های s، r، q و p به دست آمده با شرط مورد نظر مقدار A[s] - A[r] + A[q] - A[p] را برابر با بیشترین مقدار ممکن می‌کنند.

سوال 6:

برای حل این مسئله، می توانیم از الگوریتم داینامیک زیر استفاده کنیم:

1. تعریف کردن یک جدول به ابعاد n × n به نام dp که dp[i][j] بیانگر مجموع کمینه طول وترهایی است که n ضلعی محدب با رئوس i و j را تقسیم به n-2 مثلث می‌کنند.

2. مقداردهی اولیه جدول dp به صورت زیر:

- dp[i][i+1]=0 (طول وتر بین دو راس متوالی برابر با صفر است)
- dp[i][i+2] = dist(i,i+1,i+2) (طول وتر بین دو راس از راس متوالی 2 فاصله دارد و این فاصله برابر با فاصله اقلیدسی بین این ۳ راس است)
- سایر خانه‌های جدول dp را با مقدار بی نهایت مقداردهی اولیه می‌کنیم (از آنجایی که هیچ وتری نمی‌تواند دو وتر دیگر را قطع کند، مقدار بی‌نهایت برای خانه هایی که وتر می شوند به کار میرود)

3. برای هر k از 3 تا n-1، به صورت تکراری، تمام جفت راس‌های ممکن با فاصله k را بررسی کرده و با استفاده از جدول dp، طول وتر جدید را محاسبه می‌کنیم. فرض می کنیم i و j دو راس باشند که k راس فاصله دارند. در این صورت، مقدار dp[i][j] به صورت زیر محاسبه می‌شود:

- dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j] + dist(i,k,j))

4. در نهایت، مجموع کمینه طول وترها برای تقسیم n ضلعی به n-2 مثلث بدون قطع کردن هیچ وتری، برابر با dp[1][n] خواهد بود.

زمان اجرای این الگوریتم از $O(n^3)$ می باشد.