الگوریتم پیشرفته

حل تمرین سری ۲

نیلوفر وفا

۴۰۱۴۲۲۱۹۶

1.1

.رویکرد غیرهوشمندانه brute-force به این مسئله به این صورت است که هم هی حالتهای ممکن را امتحان کنیم. یعنی برای هر مهره در ابتدای ردیف، آن را انتخاب کنیم و سپس مسئله را به صورت بازگشتی بر روی ر دیف باقیمانده حل کنیم. این روش بهینه نیست زیرا تعداد حالتها به صورت نمایی با افزایش تعداد مهره ها افزایش مییابد و زمان اجرای آن بسیار زیاد م یشود.

2.1 .یک استراتژی هوشمندانه برای پاسخ بهینه به این مسئله، استفاده از الگوریتم داینا میک است. میتوان از یک جدول دوبعدی برای ذخیره سازی امتیازهای حاصل شده در هر بازه از ردیف استفاده کرد. دراین الگوریتم، با شروع از ابتدای ر دیف، به صورت بازگشتی امتیازهای حاصل شده را محاسبه میکنیم. برای هر حالت، میتوانیم یکی از دو انتخاب را داشته با شیم:
انتخاب اولین مهره یا انتخاب آخرین مهره. با انتخاب هرکدام از آنها، امتیاز به دست آمده و مجموع امتیازهای بدست آمده در ادامه ردیف را به عنوان ورودی به بازگشت استفاده میکنیم. سپس از بین دو حالت بالا، حالتی را انتخاب م یکنیم که امتیاز بیشتری دارد.
در نهایت، با ادامه این فرآیند تا انتهای ردیف، امتیاز حاصل شده و همچنین دنباله حرکات مربوطه به دست میآید.

3.1 .پیچیدگی زمانی الگوریتم داینام یک ارائه شده برابر با O(n^2) است. چرا که برای هر حالت در ردیف، حداکثر دو حالت)انتخاب اولین مهره و انتخاب آخرین مهره(را در نظر میگیریم و امت یازهای حاصل شده را در جدول دوبعدی ذخیره م یکنیم. تعداد کل خانه ها در جدول برابر با n^2 است.

۲-

حل مربوط به انتخاب بهینه ترین مکانهای توقف در طول مسیر است که به صورت حریصانه قابل حل است. الگو ریتم حریصانه ارائه شده در ادامه قابل استفاده است:

۱- خواندن ورودی:

مقدار n را برابر با حداکثر فاصله که میتوانید با پر شدن باک ما شین طی کنید قرار دهید.

خواندن مختصات جایگاههای سوخت از نقشه راه.

۲- محاسبه فواصل بین جایگا ه های سوخت:

محاسبه فاصله بین هر جفت جایگاه سوخت با استفاده از فرمول فاصله دو نقطه در فضای دوبعدی مانند فاصله اقل یدسی.

۳- الگو ریتم حر یصانه:

مقدار پیشفرض برای تعداد توقفها (count) را صفر قرار دهید.

ایجاد یک لیست خالی برای ذخیره مکانهای توقف.(stops)

تا زمانی که باک ما شین پر نشده است:

پیدا کردن نز دیکترین جایگاه سوخت)مکان جایگاه(به موقعیت فعلی.

اگر فاصله جایگاه سوخت به موق عیت فع لی از باقی فاصلهی باک کمتر یا مساوی n است:

اضافه کردن مکان جایگاه به لیست توقفها.

افزایش مقدار count به ازای توقف جدید.

کاهش مقدار باک با توجه به فاصله بین موقعیت فعلی و مکان جایگاه.

تغییر موقعیت فعلی به مکان جایگاه.

در غیر ا ینصورت، قطعه کد را تمام ک ن ید.

۴-چاپ خروجی:

چاپ تعداد کل توق فها.(count)

چاپ مختصات مکانهای توقف به ترتیب آنها در لیست توقفها.

الگوریتم حریصانه بهینه ترین مکانهای توقف را با توجه به فاصله بین جایگاههای سوخت و مکان فعلی شما انتخاب میکند. با اجرای این الگور یتم، م یتوانید توقفهایی را پیدا کنید که فاصله آنها تا موقعیت فعلی شما کمتر یا مساوی باقیماند هی باک ما شین (n) باشد، بدین ترتیب تعداد توقفها را به حداقل ممکن م یرسانید و تعداد دفعات پر کردن باک را کاهش میدهید.

با اجرای الگوریتم حر یصانه، م یتوا نید به صورت کارا و بهینه مسیر خود را برنامه ریزی کنید تا توقف های لازم را در مکانهای مناسب داشته با شید و تعداد توقفها و مسافت طی شده را به حداقل برسانید.

۳-

در این حالت، طول جادهها به صورت توانی از ۲ تعریف شده است، که اگر طول جاده ۸ کیلومتر باشد، برابر است با ۳ به توان ۲ کیلومتر. بر اساس این تو ضیحات، اسکریپت پا یتون زیر را برای محاسبه مجموع حداقل فواصل بین هر جفت شهر و چاپ خروجی به صورت دودویی ارائه م یدهیم.

با توجه به ، خروجی باید 100100 باشد که معادل عدد ده دهی 68 است.

در این اسکریپت، از الگو ریتم فلوید -وارشال برای پیدا کردن حداقل فواصل بین تمام جفت شهرها استفاده شده است. توسط این الگور یتم، فواصل کمینه بین تمام جفت شهرها به روزرسانی میشوند و در نهایت مجموع این فواصل کمینه محاسبه میشود.

الگوریتم مورد استفاده در این مسئله برای محاسبه مجموع حداقل فواصل بین هر جفت شهر به صورت زیر است:

خواندن ورودی: ابتدا تعداد شهرها و جادهها را از ورودی خوانده و آن‌ها را در متغیرهای مناسب ذخیره می‌کنیم.

ایجاد نقشه فواصل: برای ایجاد نقشه فواصل بین شهرها، یک ماتریس با اندازه N × N ایجاد می‌کنیم و آن را با مقدار بی‌نهایت (نمادی برای

فاصله بی‌نهایت) پر می‌کنیم. سپس در این ماتریس، فواصل مستقیم بین جفت شهرها را با استفاده از اطلاعات جاده‌ها به صورت مستقیم وارد می‌کنیم.

به‌روزرسانی فواصل: سپس با استفاده از الگوریتم فلوید-وارشال، ماتریس فواصل را به‌روزرسانی می‌کنیم. الگوریتم فلوید-وارشال به ترتیب تمام جفت شهرها را به عنوان گره میانی در نظر می‌گیرد و فواصل را از طریق این گره‌ها به‌روزرسانی می‌کند.

محاسبه مجموع حداقل فواصل: با جمع کردن تمام مقادیر در ماتریس فواصل، مجموع حداقل فواصل بین هر جفت شهرها را محاسبه می‌کنیم.

تبدیل به باینری: نهایتاً مقدار محاسبه شده را به صورت باینری (دودویی) تبدیل کرده و آن را چاپ می‌کنیم.

در واقع، این الگوریتم ابتدا با استفاده از اطلاعات جاده‌ها نقشه فواصل را ایجاد می‌کند و سپس با استفاده از الگوریتم فلوید-وارشال آن را به‌روزرسانی می‌کند تا در نهایت مجموع حداقل فواصل را محاسبه کند.