کلاسBFSDepth2Checker

معرفی کلاس

کلاس BFSDepth2Checker شامل متدی برای بررسی وجود ارتباط بین دو دانشگاه در یک گراف با حداکثر دو گام (عمق ≤ ۲) است. این کلاس از الگوریتم جستجوی سطحی (BFS) با محدودیت عمق استفاده می‌کند.

جزئیات کلاس

پکیج: مشخص نشده (پکیج پیش‌فرض)

سطح دسترسیpublic :

متدها

isReachableWithin2Steps

بررسی می‌کند که آیا بین دو دانشگاه با حداکثر دو گام ارتباط وجود دارد یا خیر.

پارامترها:

start (String):

نام دانشگاه مبدأ (نود شروع)

target (String):

نام دانشگاه مقصد (نود هدف)

paths (List<UniPaths>):

لیستی از تمام مسیرهای موجود در گراف که به صورت شیءهای UniPaths تعریف شده‌اند

مقدار بازگشتی:

boolean:

true در صورت وجود ارتباط بین start و target با حداکثر دو گام.

false در غیر این صورت.

شرح عملکرد:

ساخت گراف:

یک گراف بدون جهت از لیست مسیرهای ورودی ساخته می‌شود.

هر مسیر به گراف در هر دو جهت اضافه می‌شود تا بدون جهت بودن گراف حفظ شود.

آماده‌سازی:BFS

یک صف برای پیمایشBFS ، یک مجموعه برای نگهداری نودهای بازدید شده، و یک نگاشت برای ثبت عمق هر نود از نود شروع، مقداردهی می‌شوند.

نود شروع (start) به صف اضافه شده و عمق آن صفر در نظر گرفته می‌شود.

اجرای:BFS

پیمایش BFS به صورت سطح‌به‌سطح انجام می‌شود.

پیمایش در موارد زیر متوقف می‌شود:

اگر نود هدف (target) پیدا شود.

اگر عمق فعلی به ۲ برسد (حداکثر گام مجاز).

نودهایی با عمق ۲ دیگر بررسی نمی‌شوند.

نتیجه‌گیری:

اگر نود هدف در حداکثر دو گام پیدا شود، trueبرگردانده می‌شود.

در غیر این صورت، falseبازگشت داده می‌شود.

کلاسGraphPanel

معرفی کلاس

کلاس GraphPanel یک پنل گرافیکی Swing است که برای نمایش و مدیریت گراف دانشگاه‌ها طراحی شده است. این کلاس قابلیت‌های زیر را ارائه می‌دهد:

نمایش گراف دانشگاه‌ها با یال‌های جهت‌دار

رسم نمودار حرارتی (Heatmap) برای نمایش میزان استفاده از مسیرها

امکان افزودن یال‌های جدید با درگ و دراپ

نمایش درخت پوشای کمینه(MST)

سیستم رزرو و پیشنهاد مسیر هوشمند

انیمیشن حرکت دانشجویان روی مسیرها

تقسیم‌بندی منطقه‌ای گراف و محاسبه MST مقیاس‌پذیر

ویژگی‌های کلاس

ثابت‌ها

NODE_RADIUS:

شعاع گره‌ها در نمایش گراف

HEATMAP_MARGIN:

حاشیه‌های پنجرهHeatmap

فیلدهای مهم

paths:

لیست تمام مسیرهای بین دانشگاه‌ها

universityPositions:

مکان‌های دانشگاه‌ها روی پنل

universities:

لیست دانشگاه‌ها

mstEdges:

یال‌های درخت پوشای کمینه

animations:

لیست انیمیشن‌های دانشجویان در حال حرکت

reservations:

صف اولویت‌دار رزروها

usageCount:

شمارنده استفاده از مسیرها برایHeatmap

heatmapDialog:

پنجره نمایشHeatmap

متدهای اصلی

isReachableWithin2Steps

بررسی وجود ارتباط بین دو دانشگاه با حداکثر دو گام.

recordUsage

ثبت استفاده از یک مسیر برای به‌روزرسانی.Heatmap

setupTopButtons

تنظیم دکمه‌های بالای پنل شامل:

بررسی ارتباط دو دانشگاه

نمایشMST

پیشنهاد مسیر هوشمند

نمایش لیست رزروها

نمایشHeatmap

نمایش MST منطقه‌ای

setupMouseListeners

تنظیم شنودگرهای موس برای:

درگ و دراپ برای ایجاد یال جدید

انتخاب گره‌ها

paintComponent

متد اصلی رسم که موارد زیر را ترسیم می‌کند:

یال‌های گراف (به صورت خطوط مستقیم یا منحنی)

گره‌های دانشگاه‌ها (با رنگ‌بندی منطقه‌ای)

انیمیشن دانشجویان

یال‌های برجسته شده

showHeatmapDialog

نمایش پنجره Heatmap جداگانه با رنگ‌بندی بر اساس میزان استفاده.

showSuggestionDialog

نمایش دیالوگ پیشنهاد مسیر و رزرو هوشمند.

showReservationDialog

نمایش لیست رزروها با امکان حرکت دانشجویان.

computeAndShowPartitionedMST

محاسبه و نمایش MST به صورت منطقه‌ای و سراسری.

متدهای کمکی

createCurve

ایجاد منحنی بین دو نقطه برای نمایش یال‌های موازی.

drawArrowو drawArrowOnCurve

رسم پیکان جهت‌دار روی یال‌ها.

interpolateColor

میان‌یابی رنگ برای گرادیان Heatmap.

buildPathPoints

ساخت نقاط مسیر برای انیمیشن حرکت دانشجو.

کلاس‌های داخلی

AnimatedStudent

مدیریت انیمیشن حرکت دانشجو روی مسیرها.

مستندات کلاسGraphPartitioner

معرفی کلاس

کلاس GraphPartitioner یک ابزار کمکی برای تقسیم‌بندی و مدیریت گراف دانشگاه‌ها بر اساس مناطق جغرافیایی است. این کلاس الگوریتم‌هایی برای تقسیم‌بندی گراف به بخش‌های مختلف و محاسبه درخت پوشای کمینه (MST) به صورت مقیاس‌پذیر ارائه می‌دهد.

ویژگی‌های کلاس

متدهای اصلی

partitionNodesByRegion

کاربرد: تقسیم‌بندی گره‌های گراف (دانشگاه‌ها) بر اساس موقعیت جغرافیایی

ورودی: لیست دانشگاه‌ها(<List<Universities)

خروجی <<Map<String, List<Universities :که کلید آن نام منطقه و مقدار آن لیست دانشگاه‌های آن منطقه است

partitionEdgesByRegion

کاربرد: تقسیم یال‌های گراف به دو دسته درون‌منطقه‌ای و بین‌منطقه‌ای

ورودی:

نقشه مناطق) خروجی(partitionNodesByRegion

لیست تمام یال‌های گراف

خروجی <<Map<String, List<UniPaths :با دو کلید:

"intra":

یال‌های درون‌منطقه‌ای

"inter":

یال‌های بین‌منطقه‌ای

computeInterRegionMST

کاربرد: محاسبه یال‌های بین‌منطقه‌ای که مناطق را با کمترین هزینه به هم متصل می‌کنند

ورودی:

نقشه مناطق

لیست یال‌های بین‌منطقه‌ای

خروجی: لیست یال‌های انتخابی برای اتصال مناطق

الگوریتم: از الگوریتم Kruskal با ساختار Union-Find استفاده می‌کند

computeCompleteMST

کاربرد: محاسبه MST کامل با ترکیبMST های منطقه‌ای و یال‌های بین‌منطقه‌ای

ورودی:

نقشه مناطق

لیست تمام یال‌های گراف

خروجی: لیست یال‌های MST نهایی

متدهای کمکی

getNodeRegion

کاربرد: پیدا کردن منطقه یک دانشگاه خاص

ورودی:

نام دانشگاه

نقشه مناطق

خروجی: نام منطقه یا null اگر پیدا نشد

<u>جزئیات پیاده‌سازی</u>

الگوریتم‌های استفاده شده

تقسیم‌بندی منطقه‌ای: گره‌ها و یال‌ها بر اساس موقعیت جغرافیایی تقسیم می‌شوند

Kruskal

با :Union-Find برای محاسبه MST با پیچیدگی زمانی(O(E log E

بررسی پوشش کامل: اطمینان از اینکه تمام گره‌ها در MST نهایی پوشش داده شده‌اند

ساختارهای داده

:Union-Findبرای مدیریت مجموعه‌های مجزا در الگوریتمKruskal

Hash Maps: برای نگاشت سریع گره‌ها به مناطق و بالعکس

کلاسGraphUtils

<u>معرفی کلاس</u>

کلاس GraphUtils یک کلاس کمکی است که عملیات مختلف مربوط به گراف دانشگاه‌ها را انجام می‌دهد. این کلاس شامل توابعی برای محاسبه هزینه‌ها، به‌روزرسانی گراف، مدیریت ظرفیت مسیرها و ساخت ماتریس هزینه برای حل مسئله فروشنده دوره‌گرد (TSP) می‌باشد.

ویژگی‌های کلاس

<u>متدهای اصلی</u>

calculateCost

کاربرد: محاسبه فاصله اقلیدسی بین دو دانشگاه به عنوان هزینه یال

ورودی: دو شیءUniversities

خروجی: فاصله به صورت عدد صحیح

updateGraphAfterAddingUniversity

کاربرد: افزودن یال‌های جدید به گراف هنگام اضافه شدن دانشگاه جدید

ورودی:

دانشگاه جدید

لیست تمام دانشگاه‌ها

لیست مسیرهای موجود

توضیح: کم‌هزینه‌ترین یال بین دانشگاه جدید و سایر دانشگاه‌ها را پیدا و اضافه می‌کند

getMinCapacityAlong

کاربرد: یافتن کمترین ظرفیت باقیمانده در یک مسیر

ورودی: لیست یال‌ها

خروجی: کمترین مقدار ظرفیت باقیمانده

incrementCapacity

کاربرد: افزایش ظرفیت یک یال پس از حرکت دانشجو

ورودی: شیءUniPaths

buildCostMatrix

کاربرد: ساخت ماتریس هزینه برای حل مسئلهTSP

ورودی:

لیست دانشگاه‌های انتخابی

لیست تمام مسیرها

وزن زمان و هزینه

خروجی: ماتریس هزینه دو بعدی

متدهای کمکی

buildCostMatrix() اورلود شده

نسخه ساده‌تر با وزن‌های پیش‌فرض (50% زمان، 50% هزینه)

جزئیات پیاده‌سازی

الگوریتم‌های استفاده شده

محاسبه فاصله اقلیدسی: برای تعیین هزینه اولیه یال‌ها

الگوریتم دایجسترا: برای یافتن کوتاه‌ترین مسیر در ساخت ماتریس هزینه

مدیریت اولویت: با استفاده از PriorityQueue برای یافتن کم‌هزینه‌ترین یال

ساختارهای داده

PriorityQueue:

برای مدیریت یال‌ها بر اساس هزینه

ماتریس دو بعدی: برای نگهداری هزینه‌های مسیرهای مختلف

مستندات کلاس main

معرفی کلاس

کلاس main نقطه شروع برنامه "سامانه هوشمند حمل و نقل دانشگاهی" است که رابط کاربری گرافیکی را ایجاد و مدیریت می‌کند. این کلاس مسئولیت‌های زیر را بر عهده دارد:

ایجاد و مدیریت پنجره اصلی برنامه

نمایش منوها و صفحات مختلف

مدیریت داده‌های دانشگاه‌ها و مسیرها

ذخیره و بارگذاری اطلاعات

ویژگی‌های کلاس

فیلدهای اصلی

universities:

لیست دانشگاه‌ها

paths:

لیست مسیرهای بین دانشگاه‌ها

universityPositions:

مکان‌های دانشگاه‌ها روی گراف

graphPanel:

پنل نمایش گراف

mainPanel:

پنل اصلی با قابلیت تعویض صفحات

cardLayout:

مدل نمایش کارتی برای صفحات مختلف

متدهای اصلی

main

نقطه شروع برنامه

ایجاد پنجره اصلی و تنظیمات اولیه

راه‌اندازی رابط کاربری

createMainMenu

ایجاد صفحه منوی اصلی با گزینه‌های:

ساخت گراف

مسئلهTSP

ذخیره/بارگذاری داده‌ها

خروج

generateRandomUniversitiesAndPaths

تولید داده‌های تستی شامل دانشگاه‌ها و مسیرهای تصادفی

saveUniversitiesAndPaths

ذخیره داده‌های دانشگاه‌ها و مسیرها در فایل متنی

loadUniversitiesAndPaths

بارگذاری داده‌ها از فایل ذخیره شده

updateSystemAfterDataChange

به‌روزرسانی سیستم پس از تغییرات داده

createBuildGraphPage

ایجاد صفحه مدیریت گراف با قابلیت‌های:

افزودن/حذف دانشگاه

افزودن/حذف مسیر

نمایش گراف

معماری رابط کاربری

برنامه از یک معماری کارتی استفاده می‌کند که شامل صفحات زیر است:

منوی اصلی

صفحه ساخت و مدیریت گراف

صفحه مسئلهTSP

مدیریت داده‌ها

داده‌ها در لیست‌های universities و paths نگهداری می‌شوند

امکان ذخیره و بازیابی در قالب فایل متنی وجود دارد

تغییرات داده‌ها بلافاصله در رابط کاربری اعمال می‌شود

نکات فنی

از JFrame برای پنجره اصلی استفاده شده

CardLayoutبرای مدیریت صفحات مختلف به کار رفته

داده‌های تستی به صورت تصادفی قابل تولید هستند

تمام عملیات با پیام‌های مناسب به کاربر اطلاع داده می‌شود

مستندات کلاسMSTCalculator

معرفی کلاس

کلاس MSTCalculator مسئول محاسبه درخت پوشای کمینه (Minimum Spanning Tree) با استفاده از الگوریتم Primاست. این کلاس برای گراف‌های غیرجهت‌دار با یال‌های وزندار طراحی شده است.

ویژگی‌های کلاس

متد اصلی

computeMST

کاربرد: محاسبه درخت پوشای کمینه با الگوریتمPrim

ورودی:

universities:

لیست گره‌ها (دانشگاه‌ها)

allPaths:

لیست یال‌های موجود بین دانشگاه‌ها

خروجی: لیست یال‌های تشکیل‌دهندهMST

پیچیدگی زمانی O(E log V) :که E تعداد یال‌ها و V تعداد گره‌ها است

الگوریتم و منطق

مقداردهی اولیه:

مجموعه visited برای نگهداری گره‌های پردازش شده

صف اولویت (PriorityQueue) برای انتخاب یال با کمترین وزن

شروع الگوریتم:

انتخاب یک گره شروع به صورت تصادفی (اولین گره در لیست)

افزودن تمام یال‌های متصل به گره شروع به صف اولویت

مرحله اصلی:

تا زمانی که تمام گره‌ها پردازش نشده‌اند:

انتخاب یال با کمترین وزن از صف اولویت

اگر یال انتخاب شده یک گره جدید را به MST اضافه کند:

افزودن یال به نتیجه نهایی

افزودن گره جدید به مجموعهvisited

افزودن یال‌های متصل به گره جدید به صف اولویت

نکات فنی

از PriorityQueue برای انتخاب کارآمد یال با کمترین وزن استفاده شده است

برای جلوگیری از ایجاد حلقه، فقط یال‌هایی که یک گره جدید را به MST اضافه می‌کنند انتخاب می‌شوند

در صورت خالی بودن لیست دانشگاه‌ها، لیست خالی برگردانده می‌شود

کاربرد

این کلاس در بخش‌های مختلف سیستم مانند نمایش MST منطقه‌ای و سراسری در GraphPanel استفاده می‌شود.

مستندات کلاسReservation

معرفی کلاس

کلاس Reservation نماینده یک رزرو در سیستم حمل و نقل دانشگاهی است. این کلاس اطلاعات مربوط به رزرو مسیر توسط دانشجویان را نگهداری می‌کند و قابلیت مرتب‌سازی بر اساس زمان رزرو را دارد.

ویژگی‌های کلاس

فیلدها

bookingTimestamp:

زمان ثبت رزرو (برحسب میلی‌ثانیه)

studentName:

نام دانشجوی رزروکننده

origin:

دانشگاه مبدأ

dest:

دانشگاه مقصد

pathEdges:

لیست یال‌های مسیر رزرو شده

متدهای اصلی

سازنده

پارامترها: نام دانشجو، مبدأ، مقصد و لیست یال‌های مسیر

زمان رزرو به صورت خودکار با زمان فعلی سیستم تنظیم می‌شود

متدهای دسترسی(Getters)

getBookingTimestamp():

زمان رزرو

getStudentName():

نام دانشجو

getOrigin():

مبدأ

getDest():

مقصد

getPathEdges():

لیست یال‌های مسیر

getFullPathString()

کاربرد: نمایش مسیر کامل به همراه ظرفیت باقیمانده هر یال

فرمت خروجی"A->B(3)->C(1)->D(4)" :

getRemainingCapacity()

کاربرد: محاسبه کمترین ظرفیت باقیمانده در طول مسیر

خروجی: کمترین مقدار ظرفیت باقیمانده بین تمام یال‌های مسیر

compareTo()

کاربرد: مقایسه رزروها بر اساس زمان ثبت

استفاده: برای مرتب‌سازی در صف اولویت‌دار

toString()

کاربرد: نمایش ساده اطلاعات رزرو

فرمت خروجی: "نام دانشجو: مبدأ ← مقصد"

نکات فنی

کلاس Comparable را پیاده‌سازی می‌کند تا امکان مرتب‌سازی بر اساس زمان رزرو فراهم شود

از StringBuilder برای ساخت رشته نمایش مسیر استفاده شده است

متد getRemainingCapacity از Stream های جاوا برای یافتن حداقل ظرفیت استفاده می‌کند

کاربرد

این کلاس در سیستم رزرو و مدیریت صف سفرهای دانشگاهی استفاده می‌شود و توسط کلاس GraphPanel برای نمایش و مدیریت رزروها مورد استفاده قرار می‌گیرد.

مستندات کلاسTSPPage

معرفی کلاس

کلاس TSPPage یک پنل گرافیکی Swing است که مسئله فروشنده دوره‌گرد (TSP) را برای سیستم حمل و نقل دانشگاهی پیاده‌سازی می‌کند. این کلاس امکان محاسبه و نمایش بهینه‌ترین مسیر بازدید از چندین دانشگاه را فراهم می‌سازد.

ویژگی‌های کلاس

فیلدهای اصلی

graphPanel:

پنل نمایش گراف

universities:

لیست دانشگاه‌ها

paths:

لیست مسیرهای بین دانشگاه‌ها

lastOptimalOrder:

ترتیب بهینه آخرین محاسبهTSP

universityList:

لیست دانشگاه‌های قابل انتخاب

resultArea:

ناحیه نمایش نتایج

lastCostMatrix:

ماتریس هزینه آخرین محاسبه

متدهای اصلی

initUI

تنظیم رابط کاربری اصلی شامل:

پنل انتخاب دانشگاه‌ها

پنل نتایج

پنل دکمه‌ها

پنل بازگشت

createSelectionPanel

ایجاد پنل انتخاب دانشگاه‌ها با قابلیت انتخاب چندگانه

createResultPanel

ایجاد پنل نمایش نتایج محاسباتTSP

createButtonPanel

ایجاد پنل دکمه‌های عملیاتی شامل:

محاسبه مسیر

نمایش گراف

نمایش ماتریس هزینه

calculateTSP

محاسبه مسیر بهینه با استفاده از ماتریس هزینه و الگوریتمTSP

displayResults

نمایش نتایج محاسبه به صورت فرمت‌بندی شده

visualizePath

نمایش گرافیکی مسیر بهینه روی گراف

showCostMatrixDialog

نمایش ماتریس هزینه به صورت جدول رنگی

متدهای کمکی

updateUniversityList

به‌روزرسانی لیست دانشگاه‌های قابل انتخاب

highlightPathOnGraph

هایلایت مسیر بهینه روی گراف اصلی

showGraphInNewPanel

نمایش گراف در پنجره جدید با مسیر هایلایت شده

الگوریتم‌ها و منطق

محاسبه ماتریس هزینه: با استفاده ازGraphUtils.buildCostMatrix

حل مسئله :TSP با کلاسTSPSolver

نمایش نتایج: به صورت متن و گراف

مدیریت داده‌ها: ذخیره آخرین محاسبات برای نمایش مجدد

نکات فنی

از CardLayout برای مدیریت صفحات استفاده شده

رابط کاربری به صورت واکنش‌گرا طراحی شده

امکان نمایش ماتریس هزینه با رنگ‌بندی متمایز

قابلیت نمایش مسیر بهینه هم به صورت متنی و هم گرافیکی

کاربرد

این کلاس در سیستم اصلی برای حل مسئله مسیریابی بین چندین دانشگاه استفاده می‌شود و به کاربران امکان می‌دهد بهینه‌ترین مسیر بازدید از دانشگاه‌های انتخابی را محاسبه و مشاهده کنند.

مستندات کلاسTSPSolver

معرفی کلاس

کلاس TSPSolver مسئول حل مسئله فروشنده دوره‌گرد (TSP) با استفاده از الگوریتم برنامه‌نویسی پویا و روش ماسک بیتی است. این کلاس بهینه‌ترین مسیر بازدید از تمام دانشگاه‌های انتخاب شده با کمترین هزینه را محاسبه می‌کند.

ویژگی‌های کلاس

فیلدهای اصلی

m:

تعداد دانشگاه‌ها (گره‌ها)

cost:

ماتریس هزینه بین دانشگاه‌ها

dp:

جدول برنامه‌نویسی پویا برای ذخیره هزینه‌های محاسبه شده

parent:

ماتریس والد برای بازیابی مسیر بهینه

متدهای اصلی

سازنده

پارامتر: ماتریس هزینه بین دانشگاه‌ها

مقداردهی اولیه جداول dp و parent با مقادیر بی‌نهایت

solve

پیاده‌سازی الگوریتم TSP با برنامه‌نویسی پویا

پر کردن جدول dp با کمترین هزینه‌های مسیر

به‌روزرسانی ماتریس parent برای ردیابی مسیر

getOptimalPath

بازیابی مسیر بهینه از ماتریسparent

خروجی: لیست اندیس‌های دانشگاه‌ها به ترتیب بازدید

getOptimalCost

محاسبه کمترین هزینه کل برای بازدید از تمام دانشگاه‌ها

خروجی: هزینه بهینه به صورت عدد اعشاری

الگوریتم و منطق

مقداردهی اولیه:

هزینه شروع از هر دانشگاه صفر در نظر گرفته می‌شود

سایر مقادیر dp با بی‌نهایت مقداردهی می‌شوند

پر کردن جدول:dp

برای هر حالت ممکن (ماسک) و هر گره، کمترین هزینه محاسبه می‌شود

از مقادیر قبلی dp برای محاسبه حالت‌های جدید استفاده می‌شود

بازیابی مسیر:

با استفاده از ماتریسparent ، مسیر از آخرین گره به اولین گره ردیابی می‌شود

مسیر نهایی با معکوس کردن لیست بدست می‌آید

پیچیدگی

زمانی $O(n^2 * 2^n)$:که n تعداد دانشگاه‌ها است

فضایی $O(n * 2^n)$:برای ذخیره جداول dp وparent

نکات فنی

از ماسک بیتی برای نمایش زیرمجموعه‌های دانشگاه‌های بازدید شده استفاده می‌کند

برای گراف‌های کامل با $n \leq 20$ کارایی مناسبی دارد

در صورت عدم وجود مسیر، هزینه بی‌نهایت برگردانده می‌شود

کلاس UniPaths

هدف کلی

این کلاس نمایانگر مسیر (یال) بین دو دانشگاه است و اطلاعات مربوط به اتصالات بین دانشگاه‌ها را مدیریت می‌کند

ویژگی‌های کلاس

اطلاعات مسیر: شامل زمان شروع، زمان پایان، هزینه، ظرفیت و نام دانشگاه‌های مبدأ و مقصد

وضعیت مسیر: مشخص می‌کند آیا مسیر به صورت خودکار تولید شده یا دستی

مدیریت ظرفیت: پیگیری ظرفیت باقیمانده و رزروها

ویژگی‌های نمایشی: امکان هایلایت کردن مسیرهای خاص

متدهای مهم

DijkstraShortestPath

کاربرد: یافتن کوتاه‌ترین مسیر بین دو دانشگاه با الگوریتم دایکسترا

ورودی‌ها: لیست مسیرها، مبدأ، مقصد و مشخصه کاهش ظرفیت

خروجی: وضعیت یافتن مسیر (موفق/ناموفق)

ویژگی: در صورت نیاز ظرفیت مسیرهای انتخابی را کاهش می‌دهد

findShortestPathEdges

کاربرد: یافتن کوتاه‌ترین مسیر بدون در نظر گرفتن ظرفیت

ورودی‌ها: لیست مسیرها، مبدأ و مقصد

خروجی: لیست مسیرهای تشکیل‌دهنده کوتاه‌ترین مسیر

نکات فنی

از رابط Serializable برای ذخیره و بازیابی حالت شیء استفاده می‌کند

شامل متدهای استاندارد getter و setter برای دسترسی به ویژگی‌ها.

کلاسUniversities

هدف کلی

این کلاس نماینده هر دانشگاه در گراف برنامه است و اطلاعات مربوط به موقعیت و مشخصات دانشگاه‌ها را مدیریت می‌کند.

ویژگی‌های کلاس

اطلاعات پایه: نام دانشگاه، موقعیت جغرافیایی (شمال، جنوب، شرق، غرب، مرکز)

زمان‌بندی: زمان شروع و پایان فعالیت دانشگاه

مختصات: موقعیت X و Y دانشگاه در صفحه نمایش

قابلیت سریال‌سازی: امکان ذخیره و بازیابی حالت شیء

متدهای مهم

generateNewUniversity

کاربرد: تولید یک دانشگاه جدید با موقعیت تصادفی در منطقه جغرافیایی مشخص

ورودی‌ها:

نام دانشگاه

موقعیت جغرافیایی

زمان شروع و پایان

لیست دانشگاه‌های موجود

ابعاد صفحه نمایش

ویژگی‌ها:

جلوگیری از همپوشانی با دانشگاه‌های موجود

توزیع دانشگاه‌ها در مناطق جغرافیایی مختلف

محدودیت فاصله حداقل بین دانشگاه‌ها

متدهای دسترسی(Getter/Setter)

متدهای استاندارد برای دسترسی و تغییر ویژگی‌های دانشگاه شامل:

نام دانشگاه

موقعیت جغرافیایی

مختصات X و Y

زمان شروع و پایان

نکات فنی

از رابط Serializable برای پشتیبانی از ذخیره و بازیابی حالت استفاده می‌کند

موقعیت‌یابی دانشگاه‌ها با در نظر گرفتن فاصله حداقل 80 پیکسل از یکدیگر انجام می‌شود

در صورت عدم یافتن موقعیت مناسب پس از 1000 تلاش، موقعیت تصادفی بدون بررسی فاصله ایجاد می‌کند

خلاصه پیچیدگی زمانی و حافظه الگوریتم‌های مهم در پروژه

فاز اول(Minimum Spanning Tree (MST :

الگوریتم:Prim

پیچیدگی زمانی (O((V+E) log V :با استفاده از) priority queue مین-هیپ(

پیچیدگی حافظه (O(V + E :برای ذخیره گراف

الگوریتم:Kruskal

پیچیدگی زمانی (O(E log V :با استفاده از ساختار داده Union-Find بهینه‌شده

پیچیدگی حافظه (O(E :برای ذخیره یال‌ها

BFSبرای بررسی اتصال:

پیچیدگی زمانی(O(V + E :

پیچیدگی حافظه(O(V :

فاز دوم: مسیر کوتاه‌ترین

الگوریتم:Dijkstra

پیچیدگی زمانی (O((V+E) log V :باmin-heap

پیچیدگی حافظه(O(V + E :

الگوریتم:*A

پیچیدگی زمانی: به هیورستیک بستگی دارد) در بدترین حالت مانند(Dijkstra

پیچیدگی حافظه (O(V :برای ذخیره گره‌های باز

فاز سوم: مسئله فروشنده دوره‌گرد(TSP)

برنامه‌نویسی پویا با:Bitmasking

پیچیدگی زمانی (O(n^2 * 2^n :برای n دانشگاه (حداکثر 10)

پیچیدگی حافظه(O(n * 2^n :

فاز چهارم: مقیاس‌پذیری

تقسیم گراف جغرافیایی:

پیچیدگی زمانی (O(V :برای تقسیم بر اساس موقعیت

پیچیدگی حافظه (O(V :برای ذخیره اطلاعات تقسیم‌بندی

اتصال زیرگراف‌ها:

پیچیدگی زمانی $O(k^2)$:که k تعداد زیرگراف‌ها است

پیچیدگی حافظه $O(k^2)$:برای ذخیره ماتریس اتصالات

نکات کلی:

تمام پیچیدگی‌ها برای بدترین حالت محاسبه شده‌اند.

پیچیدگی حافظه معمولاً شامل ذخیره گراف و ساختارهای داده کمکی است.

برای گراف‌های پر $(E \approx V^2)$، پیچیدگی‌ها می‌توانند افزایش یابند.