OCTOBER 2023

First Project Report

TRAVEL SALESMAN PROBLEM

M.Amin HosseinNiya

Presented to:
Dr. Bourbour

دانشگاه تربیت مدرس
Tarbiat Modares University

PART 1

جزئیات هر دو پیاده‌سازی خود را توضیح دهید. به خصوص توضیح دهید که چگونه عبارات سطح بالا و انگلیسی در شبه کد را به صورت بهینه پیاده سازی کرده‌اید.

الف) رویکرد نزدیکترین همسایه (Nearest Neighbor)

شبه کد ارائه‌شده در کتاب به شرح زیر است:

NearestNeighbor(P)
    Pick and visit an initial point p_0 from P
    $p = p_0$
    $i = 0$
    While there are still unvisited points
        $i = i + 1$
        Select p_i to be the closest unvisited point to p_{i-1}
        Visit p_i
    Return to p_0 from p_{n-1}

پیش از شروع الگوریتم، نیاز است فایل ورودی که شامل تعداد نقاط و مختصات نقاط است، خوانده‌شود. بنابراین در اولین بخش کد این مرحله را پیاده‌سازی کردم. پس از آن لیستی تهیه کردم که نقاط ملاقات‌شده را در آن ذخیره کنم. اولین نقطه‌ی فایل ورودی را به عنوان نقطه‌ی شروع در نظر گرفتم. این نقطه را ملاقات‌شده فرض کردم. همچنین ابتدا طول تور را برابر صفر قرار دادم تا در طول اجرای الگوریتم آپدیت شود. پس از آن مراحل الگوریتم را مطابق شبه کد پیاده‌سازی کردم. بدین ترتیب که برای هریک از نقاط ملاقات‌نشده، فاصله‌ی اقلیدسی دیگر نقاط ملاقات‌نشده تا آن محاسبه می‌شوند و نزدیکترین آن‌ها به عنوان نقطه‌ی بعدی در نظر گرفته می‌شود. حالا فاصله‌ی این دو نقطه به طول تور اضافه می‌شود و اندیس نقطه‌ی جدید نیز به لیستی که ترتیب نقاط را ذخیره می‌کند افزوده می‌شود. بدین‌ترتیب همه‌ی عبارات سطح بالا و انگلیسی در شبه کد، معادلی در کد پیدا می‌کنند و الگوریتم بدون مشکل اجرا می‌شود.

این الگوریتم در فایل Nearest_Neighbor.py نوشته شده‌است. همچنین برای ارائه‌ی توضیحات دقیق‌تر در جای‌جای کد کامنت‌های مناسب را قرار داده‌ام تا مطالعه‌ی آن راحت‌تر شود.

PART 1

ب) رویکرد جست‌وجوی کامل (Exhaustive Search)

شبه کد ارائه‌شده در کتاب به شرح زیر است:

```
OptimalTSP(P)
    d = ∞
    For each of the n! permutations P_i of point set P
        If (cost(P_i) ≤ d) then d = cost(P_i) and P_min = P_i
    Return P_min
```

مشابه حالت قبل، ابتدا فایل ورودی را خواندم. به کمک تابع permutations در کتابخانه‌ی itertools، تمامی جایگشت‌های ممکن برای ۵ نقطه (به عبارت بهتر، برای تعداد نقاط ورودی) را به دست آوردم. سپس برای هریک از این جایگشت‌ها، طول تور را محاسبه کردم و ترتیب نقاط را ذخیره کردم. ترتیب نقاط در جایگشت دارای کمترین طول را در نهایت به عنوان ترتیب بهینه اعلام کردم. بدین‌ترتیب تمام اصطلاحات انگلیسی و سطح بالای موجود در شبه کد (مثلاً n!) معادل مناسبی در کد پیدا کردند.

این الگوریتم در فایل Exhaustive_Search.py این‌بار نیز کد را با کامنت‌های متعدد برای مطالعه و بررسی آسان‌تر کرده‌ام.

PART 2

پیچیدگی زمانی الگوریتم‌های خود در بدترین حالت را بر حسب n تعیین کنید.

الف) رویکرد نزدیکترین همسایه (Nearest Neighbor)

در الگوریتم Nearest Neighbor با یک نقطه دلخواه شروع می‌کنیم و به ترتیب نزدیکترین نقطه‌ی ملاقات‌نشده را انتخاب می‌کنیم تا همه‌ی نقاط بازدید شوند. این روند برای هر نقطه تکرار می‌شود. بنابراین برای هر نقطه، باید نزدیکترین نقطه‌ی بازدیدنشده را پیدا کنیم. برای محاسبه‌ی فاصله‌ها و یافتن کمترین فاصله، باید تمام نقاط بازدیدنشده‌ی باقیمانده را بررسی کن. این عملیات دارای پیچیدگی زمانی O(n) است.
از آنجایی که این فرآیند را برای هر نقطه تکرار می کنیم، پیچیدگی کلی زمانی برابر مقدار **O(n * n) = O(n^2)** است.

ب) رویکرد جست‌وجوی کامل (Exhaustive Search)

برای هر جایگشت، طول تور را با حرکت روی نقاط محاسبه می‌کنیم. این فرایند نیاز به n-1 بار محاسبه‌ی فاصله دارد. علاوه بر این، فاصله‌ی آخرین نقطه‌ی بازگشت تا نقطه‌ی شروع را نیز محاسبه می‌کنیم. بنابراین، تعداد کل دفعات محاسبه‌ی فاصله برابر n است.
از آنجایی که این مراحل را برای همه‌ی جایگشت‌ها تکرار می‌کنیم (که تعدادشان برابر n! است) و برای هر جایگشت n بار محاسبه‌ی فاصله را انجام می‌دهیم، پیچیدگی زمانی کلی **O(n * n!)** است.

PART 3

از یک تولید کننده اعداد تصادفی برای تولید ورودی الگوریتم‌های خود به ازای حداقل چهار n متفاوت استفاده کنید. مقادیر n برای هریک از دو رویکرد ممکن است نیاز باشد که متفاوت باشد. به ازای هر n، سه مرتبه برنامه را با ورودی یکسان اجرا کنید و میانگین زمان اجرا در این ۳ مرتبه را به عنوان زمان اجرا در نظر بگیرید. نتایج خود را در یک جدول نمایش دهید و توضیح دهید که n را چگونه انتخاب کردید.

در فایل input_creator.py کدی نوشته‌ام که مطابق خواسته‌ی سوال، فایل‌های متنی با فرمت مناسب را تولید می‌کند. اعداد تولیدشده شامل اعداد صحیح بین صفر و ۵۰ هستند.

الف) رویکرد نزدیکترین همسایه (Nearest Neighbor)

برای این الگوریتم n را برابر با اعداد ۳۰، ۴۰، ۵۰ و ۶۰ قرار دادم. الگوریتم را با هریک از این ورودی‌ها ۳ بار اجرا کردم و میانگین زمان اجرا را به عنوان زمان اجرای الگوریتم برای آن ورودی در جدول زیر قرار دادم:

n	زمان اجرا
30	0.0009973049163818360
40	0.0009961922963460286
50	0.0012662410736083984
60	0.0013295809427897136

PART 3

ب) رویکرد جست‌وجوی کامل (Exhaustive Search)

برای این الگوریتم n را برابر با اعداد ۵، ۶، ۷ و ۸ قرار دادم. الگوریتم را با هریک از این ورودی‌ها ۳ بار اجرا کردم و میانگین زمان اجرا را به عنوان زمان اجرای الگوریتم برای آن ورودی در جدول زیر قرار دادم:

n	زمان اجرا
5	0.0009972254435221355
6	0.0096317132314046 23
7	0.034587224324544 27
8	0.3031783103942871