به نام خدا

دانشگاه اصفهان

مستند پروژه ساختمان داده

دکتر فاطمی

میترا عمرانی 993613047

شقایق شهبازی 993613040

پذیرش مشتری:

در این بخش ورودی های لازم از جمله مشتریان رستوران و همچنین نقشه رستوران دریافت خواهد شد. به این منظور کلاسی تحت عنوان customer برای ذخیره کردن اطلاعات هر مشتری در نظر گرفته شده است. از جمله نام، نام غذای سفارش داده شده، مدت زمان آماده سازی غذا و مدت زمانی که خوردن غذا می شود.

تحویل غذا توسط گارسون:

در این بخش نقشه رستوران دریافت و ذخیره خواهد شد سپس طول کمترین مسیری را که می توان در طی آن به تمامی میزها غذاها را رساند چاپ خواهد شد.در واقع برای انجام این بخش از حل مسأله TSP و الگوریتم BFS استفاده شده است. **(بخش امتیازی)**

همچنین علاوه بر بخش امتیازی بخش غیرامتیازی نیز به طور کامل پیاده سازی شده است. توابع مهم این بخش عبارتند از:

- **print_path()** : در این تابع پس از پیداکردن کوتاه ترین مسیر ، مسیر موردنظر با رنگ متفاوت در نقشه چاپ خواهد شد. (مربوط به بخش غیرامتیازی)
- **find_shortest_path()** : در این تابع از الگوریتم BFS استفاده شده است که کوتاه ترین مسیر را پیدا کرده و آن را ذخیره می کند. (مربوط به هردوبخش امتیازی و غیرامتیازی)
  - **مرتبه زمانی : O(V+E)**
- **calculate_Min_distn()** : تابعی که در آن کوتاه ترین فاصله میان تمامی نودها محاسبه خواهد شد. برای این کار درواقع از تابع find_shortest_path() استفاده می شود. (مربوط به بخش امتیازی)
- **Tsp()** : این تابع پیاده سازی مشابه حل مسأله TSP به کمک dynamic programming است و یک تابع بازگشتی است . که درواقع این مسأله راه دیگری هم دارد که با محاسبه تمام جایگشت ها صورت می گیرد اما مرتبه زمانی آن O((N-1)!) است. در حالیکه با روش dynamic programming از نظر زمانی به صرفه تر است . (مربوط به بخش امتیازی)
  - **مرتبه زمانی : O(N^2*2^N)**

آشپزخانه:

در این بخش نام غذا و سپس پیش نیازهای آن وارد شده و تعدادی عملیات بر روی آن انجام می شود. که برای این کار یک کلاس به نام Food در نظر گرفته شده که در آن نام غذا، پیش نیازها (به فرم درخت با کمک لیست همسایگی)، و مدت زمانی طول کشیده شده است تا غذا با توجه به پیش نیازهایش آماده شود، آمده است. توابع مهم این بخش عبارتند از:

- **TopologicalSort()** : این تابع به کمک لیست همسایگی اولویت های هر غذا، آنها را به ترتیب اولویتشان مرتب و ذخیره خواهد کرد.
  - **مرتبه زمانی : O(V+E)**
- **AddRelation()** : این تابع رابطه جدیدی بین دو پیش نیاز یک غذا ایجاد خواهد کرد . اما به منظور ایجاد نشدن دور در گراف پیش نیازهای غذا، ابتدا به کمک تابع ThereIsPath() بررسی خواهد کرد که از پیش نیاز دوم به پیش نیاز اول که نودهای گراف هستند ، مسیری وجود نداشته باشد. در صورتی که مسیری وجود نداشت این ارتباط جدید را اضافه می کند در غیراینصورت خطای مربوطه را چاپ خواهد کرد.
  - **مرتبه زمانی پیدا کردن مسیر : O(V+E)**
- **DeleteFood()** : این تابع نام غذای را دریافت کرده و آن را از لیست تمام غذاهای موجود حذف خواهد کرد.
  - **مرتبه زمانی : O(N)**
- **hasMaxPrereq()** : این تابع نام غذایی را که در بین تمام غذاهای موجود بیشترین تعداد پیش نیاز را دارد، چاپ خواهد کرد.
- **MaxMinTimePrepareFood ()** : این تابع نام غذاهایی را که کمترین و بیشترین زمان آماده سازی را داشته اند چاپ خواهد کرد.

مهمانی:

در این بخش چینش مهمان ها همانند درخت AVL است بنابراین یک کلاس به نام Node در نظر گرفته شده که علاوه بر ویزگی هایی که نودهای درخت AVL دارند، نام هر فرد و شماره نوبت آن شخص هم درنظر گرفته

شده است. و نیز یک کلاس دیگر هم به نام AVLTree تعریف شده که توابع مربوط به درخت AVL و اشاره گری به ریشه آن، در آن وجود دارند. توابع مهم این بخش عبارتند از:

- **Insert()** : در این تابع هر مهمان جدیدی که وارد میشود با توجه به قوانین درخت AVL و شماره نوبت آن فرد به درخت اضافه خواهد. برای این بخش در واقع از الگوریتم اضافه کردن نود جدید به درخت AVL بر مبنای نوبت هرفرد استفاده شده است.
  - **مرتبه زمانی : O(logN)**
- **deleteNodeByTurn()** : در این تابع هر مهمانی که قصد خارج شدن را دارد به کمک الگوریتم حذف از درخت AVL و با توجه به نوبت آن فرد، حذف خواهد شد.
  - **مرتبه زمانی : O(logN)**
- **deleteNodeByName()** : تابعی برای حذف مهمان ها با دریافت نام آنها. درواقع این تابع نام هر مهمان را دریافت کرده و به کمک تابع SearchByName() نوبت آن مهمان را به کمک الگوریتم DFS پیدا کرده و به عنوان خروجی میدهد.پس از یافتن نوبت فرد موردنظر آن را به کمک تابع deleteNodeByTurn() از درخت حذف خواهد کرد.
  - **مرتبه زمانی : O(NlogN)**
- **SearchByTurn()** : این تابع با دریافت نوبت فرد با توجه به الگوریتم سرچ در درخت BST آن را پیدا می کند و نام آن را چاپ خواهد کرد.
  - **مرتبه زمانی : O(logN)**
- **ShowTree()** : این تابع درخت را به کمک ریشه آن چاپ خواهد کرد. در این تابع برای زیبا و خوانا چاپ شدن درخت از الگوریتم BFS استفاده شده و آن را سطر به سطر پیمایش کرده و ذخیره میکند و درنهایت درخت را چاپ خواهد کرد. **(بخش امتیازی)**