# SG Travel Planner

An Android Travel App By Du Li (1002187), Gao Yiming (1002180), Yin Yuanzhi (1002206)

## App description

SG Travel Planner is an Android travel app for travelers in Singapore as a tour guide and planning assistant. It is able to locate popular tourist attractions, save addresses for certain places, and do daily itinerary planning with a specified budget.

## UI Design

The opening screen of our app is simple and clean, with a text input field at the center for users to type in the location they want to view. We use bottom navigation view to navigate between the activities, the Itinerary Planner, MY Favorite, and Settings. The contents of the bar at the bottom can be populated by specifying a menu resource file. Each menu item title, icon and enabled state will be used for displaying bottom navigation bar items.

After you enter the google map interface, you can also search other places by using the search bar, the user can easily find the tourist attractions they want with Robust Spell Checking implemented. Our Maps fragment uses Google Maps API. It also shows the address and postal code of the location, and can move around the map to look for landmarks or nearby train stations. When clicking on the pointer on the map, there will be an implicit intent that allows you to jump to Google Maps. By clicking the Set Favorite button, you can then add different locations to My Favorite so that you can view the places later. This feature is implemented by storing data in an SQLite Database, and displaying the favorite places in a RecyclerView.

When you click the Itinerary Planner button at the bottom and enter the interface, you will be able to choose several tourist attractions from the location list and specify a budget. The App will then calculate the fastest way based on the budget specified and display in an alert dialog. This includes deciding the order of seeing the attractions, and starts and ends at the original position (for now, it is Marina Bay Sands).

And when you come to Settings, the user can select their preference. You will be able to adjust the font size, change to night mode, or choose from brute force and fast algorithm for itinerary planning.

## Itinerary Planning

For Itinerary Planning, we use a JSON file to store map data, and designed two algorithms to calculate the fastest path.

***Exhaustive Search:***
The itinerary planning involved a recursive algorithm that generates all possible permutations of the given list of attractions with different transports. After picking out the paths that satisfy the budget requirements, it finds the one that takes the least time with corresponding transports.

***Fast Approximate Solver:***
We use simulated annealing algorithm to build the fast solver. Start with a random tour through the selected locations, choose two locations on the tour randomly, and then reverse the portion of the tour that lies between them. If the new tour is better than the existing tour, accept it. If the candidate tour is worse than the existing tour, still maybe accept it, according to some probability. The probability is calculated using function

$P = e^{(\text{current travel time} - \text{new travel time})/\text{temperature}}$.

Then repeat this step for many times, lowering the temperature a bit at each iteration (initial temperature = 100, cooling rate = 0.1), until we get to a low temperature and arrive at a minimum. The transport for each sub-path is also selected randomly using similar method as selecting a new tour.

***Conclusion:***
The Brute Force method is more accurate, as it lists all the possible paths and chooses the fastest one, while the fast algorithm may not be able to get the optimal result as it chooses paths randomly and is not able to compare all the paths. And certainly, the running time of the fast algorithm is almost always less than brute force (except when there is only one path), as it does not need to list all the possible paths.

## Robust Spell Checker

Our Robust Spell Checker compares the length of longest common subsequence and substring present both in correct names in our data file and the name typed in. The weight for longest common subsequence is 0.6, and substring is 0.4. First, we find the length of longest common subsequence & substring of each pair of words, then choose the location name corresponding to the largest value after weighting to correct typo. If the correct name is a phrase, we will calculate similarity for every word in the phrase. And if the input matches any of the abbreviation of the locations (initial letters of the name), the locator will also point out the right place.