430.329: Introduction to Algorithm **Spring 2025**

Final Project

(Due on June 13, 2025 11:59PM KST)

Instructor: Jonghyun Choi (jonghyunchoi@snu.ac.kr)
TAs: Byeonghwi Kim and Taeheon Kim

1 Overview

The final project is to implement the Traveling Salesman Problem (TSP) and improving the cost while reducing the computational cost. The TSP problem requires you to find the shortest possible tour that visits each city exactly once and returns to the starting city. Although the finding the optimal solution of TSP is NP-Hard, many exact and heuristic algorithms are widely used in practice.

This project wants you to do the following :

1. **Implementation**. Implement algorithms that are correct for all inputs with n **(graph node size)** ≤ 10 and at least one scalable heuristic capable of addressing $n \geq 1000$. You may use any AI tools to get help for implementation.
2. **Empirical evaluation.** Measure the tour cost and run-time. We will use these two metrics to rank your submissions.

2 Minimum Requirements

Here are minimal requirements of this project. Successfully implementing them will give you the full score for the base case.

1. **2-step greedy solver.** Implement a heuristic TSP algorithm, 2-step greedy solver, which iteratively finds the next adjacent vertex with the minimum distance from its adjacent vertex.
   Specifically, given a set of nodes, V, and a starting node, v_{start}, we start the algorithm by defining a set of visited and unvisited nodes, V_{visited} and $V_{\text{unvisited}}$ respectively, and set the currently visited node v_{current} to the starting node as in Equation 2.1 :

$$V_{\text{visited}} \leftarrow \{v_{\text{start}}\} \quad V_{\text{unvisited}} \leftarrow V - \{v_{\text{start}}\}, \quad v_{\text{current}} = v_{\text{start}}. \tag{2.1}$$

   We then iteratively finds the next, adjacent node, v_{next} with the minimum distance, $d(v_{\text{next}}, w)$, from its adjacent node, w as in Equation 2.2 :

$$v_{\text{next}} = \underset{v}{\operatorname{argmin}}\, d(v_{\text{current}}, v) + d(v, w), \quad v, w \in V_{\text{unvisited}}, \quad v \neq w, \tag{2.2}$$

   where $d(\cdot, \cdot)$ denotes a distance function. Note $d(u, u) = \infty$ if u and v is not adjacent.
   Once the next node is determined, we add/remove the next node to the visited/unvisited set, $V_{\text{visited}}/V_{\text{unvisited}}$, as in Equation 2.3 :

$$V_{\text{visited}} \leftarrow V_{\text{visited}} \cap \{v_{\text{next}}\}, \quad V_{\text{unvisited}} \leftarrow V_{\text{unvisited}} - \{v_{\text{next}}\}. \tag{2.3}$$

   We repeat Equation 2.2 and Equation 2.3 until $V_{\text{visited}} = V$.
2. **Testing.** Get the correct cost for given 15 Test cases using 2-step greedy solver.

3 Open-Ended Competition

2-step greedy solver is not an optimal solution. It is not the fastest algorithm, it cannot guarantee to find the optimal (minimum) cost, and it may fail to find valid tour route for sparsely connected graph. **If you want to get more score beyond the base score, you need to improve your heuristic algorithm for both optimality of the cost and the efficiency of the algorithm.**

1. Implement creative ideas to improve rewards and reduce computation. The rank in the leaderboard is determined by the combined measure of the rewards and computational cost.
2. You can refer to the following methods for improvement
   (a) Dynamic programming with bit masking (URL)
   (b) Branch and bound (URL)

4 How to Get, Edit & Run the Base Project Code (using 'Elice' Platform in eTL)

1. **Launch the online IDE.**
   (a) Go to eTL → 2025-1 Introduction to Algorithms (001)
   (b) Go to **Coding Class** → **Go to coding class** (코딩수업 → 코딩수업 바로가기). You should be able to see our course automatically.
   (c) Go to **Traveling Salesman Problem Project** → **TSP Solver Implementation**

   You will see `main.cpp` and example graph (*.txt) files. *Do all development inside this browser IDE.* Your edits are auto-saved to the cloud. (You can do implementation locally if you want.)
2. **Build & test your code in the online IDE.** Press the **Run** (실행) button. The built-in script compiles your code and executes it on every `tests/*.txt` graph, printing each tour cost and runtime.
3. **Submit your code in the online IDE to compare with Ground-Truth cost.** Press the **Submit** (제출) button. The built-in script compiles your code, executes it, and compare with the ground-truth cost for each graph, printing the correctness for each graph. (You should take a screenshot for the printed log on Elice and put it on your report.)

5 How to Submit Your Work to the Leaderboard in Gradescope

You need to submit your code from Elice to Gradescope for Leaderboard system. Sadly, there is no *download* in Elice, so you need to create `.cpp` file locally, and then submit to Gradescope.

1. **Copy your final source from Elice.** Click anywhere inside the Elice editor pane containing `main.cpp`, then press :

   `Ctrl+A` → `Ctrl+C` (`Cmd+A` → `Cmd+C` on macOS)

   This places the entire file on the clipboard.
2. **Create a local `.cpp` file.**
   (a) **Windows 10/11**
      i. Open `notepad` (메모장).
      ii. Paste with `Ctrl+V`.
      iii. `File` → `Save As`(다른 이름으로 저장), set `Save as type: All files`, name the file `tsp.cpp`, and save it.
   (b) **macOS**
      i. Open Spotlight (`Cmd+Space`) → **TextEdit**.
      ii. `Format` → `Make Plain Text`.
      iii. Paste with `Cmd+V`.
      iv. `File` → `Save`, choose `tsp.cpp`, click `Use \.cpp"`.
   (c) **Linux (Ubuntu, Fedora, etc.)**
      i. Graphical way – gedit :
         A. Press the `Super` key, type `gedit`, and hit `Enter`.
         B. Paste with `Ctrl+V`.
         C. `File` → `Save As`, name the file `tsp.cpp`, and click `Save`.
      ii. Terminal way – vi / vim :

A. Open a terminal and run `$ vi tsp.cpp` or `$ vim tsp.cpp`.
B. Press `i` to enter *Insert* mode, then paste with `Shift+Ctrl+V`.
C. Press `Esc`, type `:wq`, and press `Enter` to save and quit.

3. **Upload the file to Gradescope.**
   (a) Visit https://www.gradescope.com, select the course and the assignment **TSP Leaderboard**.
   (b) Drag `tsp.cpp` onto the upload area or use the `Browse` button.
   (c) Wait for the autograder. A green check means your code compiled and ran on all hidden graphs.

   Note : We limit the maximum number of submissions by **7 times**. Only **the first 7 submissions** will be considered for the evaluation. Please plan your submissions carefully.

4. **Check your rank.** Refresh the page or open the **Leaderboard** tab. Gradescope orders entries by :
   i. **Valid tours** (desc) – how many test graphs produced a correct Hamiltonian tour,
   ii. **Total Cost** (asc) – lower is better,
   iii. **Runtime** (asc) – faster is better.

6 Evaluation Metric

1. **Tour validity check** (performed per test graph)
   1.i. starts and ends at the same vertex,
   1.ii. visits every vertex exactly once,
   1.iii. reported cost equals the sum of edge weights.

2. **Leaderboard keys**
   (a) **Valid** (*descending*)
   Number of graphs where the tour passed all checks.
   (b) **TotalCost** (*ascending*)
   Sum of your costs across *valid* tours. (*invalid* tours will add 1,000,000,000,000 to the total cost.)
   (c) **Runtime(ms)** (*ascending*)
   Accumulated wall-clock time for all graphs.

3. **Tie-breaking** Higher **Valid** wins first. If equal, the smaller **TotalCost** wins ; if still equal, the smaller **Runtime(ms)** wins.

7 Report

Your report should include following items. If you have other items to report additionally, it is absolutely fine.

1. Greedy 2-Step Algorithm
   (a) Explaining Your implementation of greedy 2-opt algorithm
   (b) Log print of your algorithm (Screenshot of Elice when you submit)
2. Your idea to improve the baseline
   (a) High-level intuition of your idea
   (b) Implementation of your idea
   (c) Log print of your algorithm (Screenshot of Gradescope showing validity, cost, and runtime of your algorithm)