CS 486/686: Assignment 1 (100 Points)

Instructors: Jesse Hoey & Victor Zhong
Due Date: Wednesday January 29th, 2025 at 11:59 pm (ET; Waterloo Time)

Instructions

- Submit any written solutions in a `.zip` file named `solution.zip` to the A1 Dropbox on `LEARN`. In the `solution.zip` file, include
  - A file named `writeup.pdf` that contains all your written solutions;
  - A file named `code.py` that contains your code.

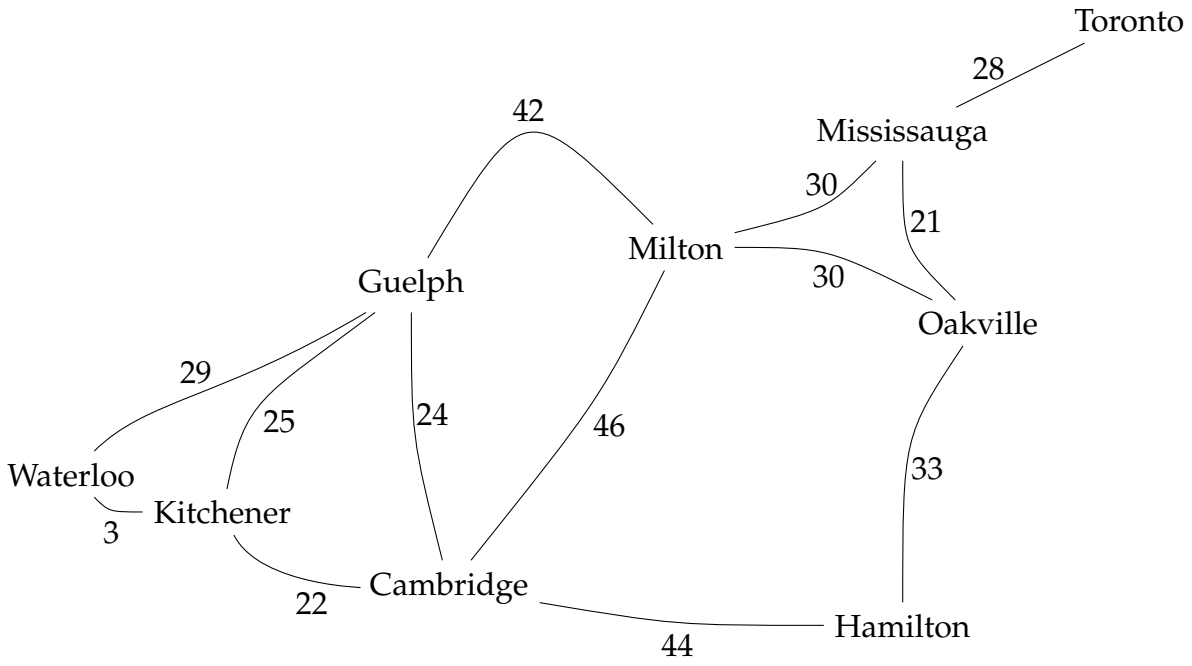
  Do not include any additional files in the submission `.zip` file, including but not limited to `__MACOSX` directories, `.DS_Store` files, and any auto-generated files for Python execution.

  If your submission does not exactly contain two files named `writeup.pdf` and `code.py`, we will deduct 5 points from your final assignment mark.
- Use the latest Python 3.12 to implement the code.
- No late assignment will be accepted.
- This assignment is to be done individually.
- Lead TAs:
  - Ruoxi Ning (r2ning)
  - Arthur Chen (h559chen)

  The TA office hours will be scheduled on Piazza.

1 Shortest Route to Waterloo (20 points)

Suppose that you want to drive to Waterloo from Toronto. You are conscious of your carbon footprint; therefore, you are seeking the shortest path to Waterloo. Below is a graph indicating the driving distances between various cities surrounding Toronto and Waterloo. All distances are given in kilometres.

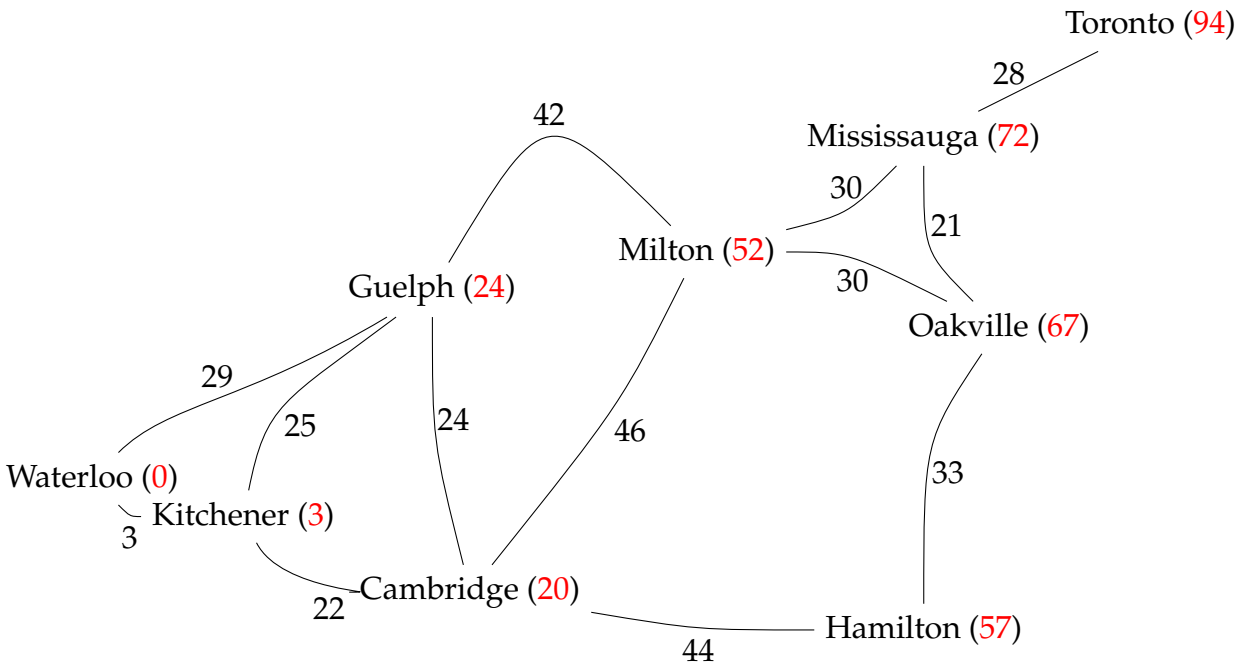


You would like to apply A* search to identify the shortest route to Waterloo from Toronto. Below are the components of the search problem formulation:

- **States:** Each city is the state. We will identify each state by the first 3 letters of its name. For example, the "Guelph" node is denoted as `Gue`.
- **Initial state:** `Tor`
- **Goal state:** `Wat`
- **Successor function:** State B is a successor of state A if and only if there exists an edge on the above graph connecting city B and city A.
- **Cost function:** The cost of an edge connecting two states is the distance between the cities that the states correspond to.

Your decide to use Euclidean distance as a heuristic function. That is, h is the Euclidean distance from a city to Waterloo (in kilometres). That is, if (x_C, y_C) is city C's longitude and

latitude coordinates, $h(C) = \sqrt{(x_C - x_{Wat})^2 + (y_C - y_{Wat})^2}$. On the diagram below, the red text number to each city indicates its Euclidean distance to Waterloo.



Please complete the following tasks.

1. **[5 pts]** Describe why Euclidean distance to the destination is a consistent heuristic function. Use no more than 4 sentences.

2. **[15 pts]** Execute the A* search algorithm on the problem using the Euclidean distance heuristic function as described above. Do not perform any pruning. Add nodes to the frontier in alphabetical order. Remember to stop if you expand the goal state.

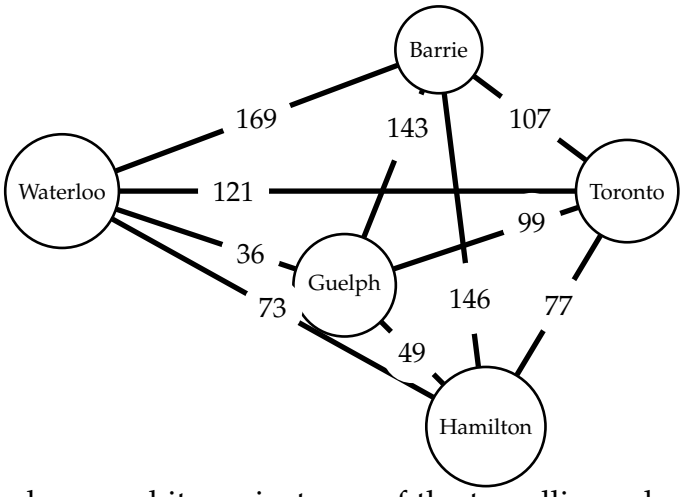
   When drawing nodes, remember to abbreviate cities by writing the first 3 letters. For example, label a "Waterloo" node as `Wat`. Annotate each node in the following format: $C + H = F$ where C is the cost of the path, H is the heuristic value, and F is the sum of the cost and the heuristic values. Clearly indicate which nodes you expanded and in what order. You do not need to write out the frontier, but the tree must show all paths expanded after removing a node from the frontier.

   Break any F-value ties using alphabetical order. For example, "Milton" precedes "Mississauga" and should be expanded first if the F values for both nodes are the same.

   We recommend using `https://app.diagrams.net/` to draw the search tree.

2 Travelling Salesperson (30 points)

The Travelling Salesperson Problem (TSP) is a famous problem with many applications. As a small example, a salesperson must visit each of the 5 cities shown in the map on the right. Starting at Waterloo, the problem is to find a route of minimal distance that visits each of the cities only once and returns to Waterloo. You can assume all cities are connected to all other cities (the graph is complete).



Suppose we wish to use algorithm A* to solve an arbitrary instance of the travelling salesperson problem (NOT just the one above, but any TSP).

(a) **[5 pts]** Give a representation of the general TSP problem stated above suitable for A* (that is, describe states and their representation, the initial state, the goal state or goal condition, and the neighbour rules - how to generate neighbours of a node - and their cost). Use the small example above to illustrate your representation, but your representation must be applicable to any (complete) TSP. You can use the letters W,B,G,H,T to refer to the cities by their first letter when referring to the small example above.

(b) **[5 pts]** Specify the *cost* function. Be precise. Give an example using the small problem above so it is clear what you are proposing.

(c) **[10 pts]** Propose an admissible $h(n)$ (heuristic) function for node n. Be precise. Give an example of computing $h(n)$ for one node using the small problem above. your admissible heuristic cannot be trivial (e.g. 0 everywhere) - it has to be an actual heuristic, that is, it has to give you information about where to go next.

(e) **[10 pts]** Using your heuristic and cost function, show part of the search tree produced by algorithm A* when applied to the small example shown above. Specifically, continue until you have expanded (generated the successors of) **eight** nodes. Break ties by expanding the node with the smallest value of the cost to get to that node. Label each node with its cost value (total path cost to get to that node) and its heuristic value. Also, number the nodes to indicate the order in which they are expanded.

3 Sentence with the Highest Probability (50 points)

An autoregressive language model parameterized by Θ defines the probability of a sentence $S = \langle w_1, \ldots, w_n \rangle$ as the product of the probabilities of each word w_i in the sentence given the words that came before it; that is,

$$P_\Theta(S) = P_\Theta(w_1)P_\Theta(w_2|w_1)\ldots P_\Theta(w_n|w_1, w_2, \ldots, w_{n-1})P(\langle \text{EOS} \rangle \mid w_1, w_2, \ldots, w_n).$$

Here, $\langle \text{EOS} \rangle$ is a special token that indicates the end of a sentence, and should be added to the end of every sentence. The language model is associated with a vocabulary V of words, and can only handle sentences that only contain words in V.

You are now given an API of this language model `prob`(w, c), which takes two arguments, word of interest w and prefix $c = \langle c_1, c_2, \ldots, c_m \rangle$ and returns $P_\Theta(w \mid c_1, c_2, \ldots, c_m)$ in constant time.

We are interested in finding the highest probability sentence of length n.

1. **[5pts]** An intuitive way is to greedily select the word with the highest probability at each step, and it will return one sentence. Does this approach always find the highest probability sentence? If yes, provide a proof. If no, provide a counterexample with $|V| \leq 3$ and $n \leq 3$, where your counterexample should specify the value of $P(w)$ and $P(w \mid c)$.

2. **[5pts]** What is the time complexity of finding the highest probability sentence of length n using this autoregressive language model?

3. **[15pts]** Another way to find the highest probability sentence is to use lowest-cost-first search. Please describe how to formulate this problem as a search problem. You should define the states, initial state(s), goal state(s), successor function, and cost function.

4. **[5pts]** What is the memory complexity of finding the highest probability sentence of length n using lowest-cost-first search?

5. **[20pts]** Now we restrict the size of the frontier to k in the lowest-cost-first search; that is, when we add a path to the frontier (implemented with a priority queue), we only keep the k paths with the lowest cost. Please implement this algorithm in Python by completing the function `search` in `code.py` provided in the assignment material.

   For sanity check, if you implement the function correctly, executing the code with `python code.py` should print out "`success!`" to the console. In addition to the language model provided in the assignment material, we will grade your implementation using ten random language models.