Assignment 1
CSC 520 – Artificial Intelligence – Fall 2018
Juan Jose Rivera
200262237

1. [20 pts.] Suppose we have an agent who works for the Hunt Library. The agent is assigned a task of transferring a set of boxes one-by-one by lifting them from location A and placing them in location B inside the building. Sensors can tell it whether the agent is near its destination or not. The rooms have stationary obstacles whose locations are not initially known. If the agent bumps into an obstacle, it will drop the box it is carrying. Some boxes contain fragile contents which will break if dropped. The environment contains obstacle-free paths of various lengths. Answer the following questions about the agent.

    1. Define a PEAS specification for the agent.

	Performance	Environment	Actuators	Sensors
Hunt Library Agent	• Percentage of boxes successfully moved to destination. • Condition of boxes and items inside the box. • Time taken to destination. • Distance. • Energy used.	• Library box depot. • Library areas, aisles, hallways, rooms, stairs, elevators. • Library Staff. • Library Customers. • Boxes.	• Agent arms to pick up the boxes. • Legs and wheels to move around the library. • Device to send alert in case of malfunction.	• Camera. • Proximity sensor. • Keyboard entry for configuration (set origin, destination, library map) • Weight sensor (to validate if agent can move the box properly)

    2. Is it sufficient for the agent to be simple reflex? Why or why not? And if not what level is necessary?

    It is not enough for the agent to be simple reflex agent. Given that the main objective of this agent is to achieve a desirable goal (take a box to a defined destination from a defined origin). It is not possible to achieve this with a simple reflex agent that acts based on the current percept only, does not keep track of the environment states, and moreover does not have information of any Goal (desired state).
    The Hunt Library agent in order to achieve a goal should be at least a goal-based agent, and if we want to maximize its utility and improve its performance a goal-based agent is not enough, we need to have an utility-based agent. Both agents (Goal-based and utility-based) could improve performance by adding learning to them.

    3. Would the ability to move randomly help agent performance or not? Identify possible disadvantages to this sort of movement.

    Move randomly does not improve performance of the agent, because in some cases instead of maximizing its performance, given the nature of a random generated move, agent could minimize it. If we take the random movement approach, instead of reducing the distance between current position to goal position, we could end up increasing this distance (Goal located south of current position and random generated movement result is north). This is going to be detrimental for agent's performance increasing time to destination, energy used, traveled distance, etc.

4. Suggest one improvement to the agent design. Since every improvement carries drawbacks, what are the drawbacks to yours?

   Assuming that the designed agent is a Utility-based agent, it is possible to improve this agent by adding learning of environment trends, such as traffic in areas due to concentration of people. Imagine a career fair that takes place every Friday at 3.pm that causes high density of people in the main hall of the library, and this hall happens to be used for the agent as part of the route to get from A to B. Adding a traffic learning component to the agent is going to help to identify this recurrent event and try to avoid routes that has the main hall as part of it during the duration of event. This reduces the impact on performance that this event causes. However, this learning component adds an additional complexity to the agent by adding new components to the existing design like learning element, performance element, critic and problem generator.

2. [20 pts.] Fill in the following table with proper description of the agent's environment.

	Fully vs. Partially observable	Deterministic vs. Stochastic	Episodic vs. Sequential	Static vs. Dynamic	Discrete vs. Continuous	Multiagent vs. Single agent
Vacuum Cleaner Agent	Partially, the agent does not have sensor all over the surface in which the agent works.	Deterministic, next state is determined by current state and action, move to left always going to move to left(except if agents is in boundary of environment).	Episodic, does not need previous states to work.	Dynamic, dirt could get accumulated changing the environment.	Discrete, environment is limited to a room to clean.	Single agent, only one vacuum.
Google Car	Partially, agent only knows a part of the environment.	Stochastic, not possible to predict next state of the environment. (Traffic, accidents)	Sequential, needs to take into account previous states.	Dynamic, environment is changing constantly roads, traffic other cars.	Continuous, there is not a finite set of states.	Multiagent, multiple cars in the environment.
Research and Rescue	Partially, not able to check whole area that is going to be searched.	Stochastic, finding people that need to be rescue is uncertain, not only depends on agent actions.	Sequential, needs to keep track of previous states to avoid redundancy. Does not need to check the same place 2 times.	Dynamic, environment changing constantly, people moving looking for help.	Continuous, there is not a finite set of states.	Single, only one team looking in the same defined environment.
Document Categorizer	Fully, knows all the	Deterministic, document fits	Episodic, does not	Static, environment	Discrete, there is	Single Agent, only one

	information that need to be processed.	in to the category.	need previous states to categorize a new document.	stays the same, same documents.	finite set of documents to categorize.	agent working.

3. [10 pts.] Answer the following questions.
    1. Describe a PEAS (R&N Ch. 2) specification for Watson.

	Performance Measure	Environment	Actuators	Sensors
Watson	• Winning the game. • Percentage of answered questions. • Percentage of correct answered question. • Time to get answers. • Time to send signal to buzzer.	• Set of question. • Participants. • Game host.	• Display of answer in natural language. • Voice generator. • Device to press buzzer.	• Input mechanism to send categories, questions/clues, etc. to Watson

    2. Describe Watson's environment (Full/Partially Observable, Deterministic/Stochastic etc.).

	Fully vs. Partially observable	Deterministic vs. Stochastic	Episodic vs. Sequential	Static vs. Dynamic	Discrete vs. Continuous	Multiagent vs. Single agent
Watson	Partially, only the current snapshot of the environment, next responses, questions, and answers are not observable.	Stochastic, next state is determined not only by the action of the agent, but by time and the other participants and questions.	Sequential, actions affect future states of the environment (bad question answered, slow buzzer response)	Dynamic, new questions appear with different level of difficulty and other game participants change environment.	Continuous, actions are based on a set of probabilities, the environment could have a different state depending in which second the agent press the buzzer.	Multiagent, multiple participants on the game.

3. Discuss at least three separate aspects of the Jeopardy problem domain together with the hardware and/or software design choices in Watson that are rational given those problem aspects.

   Questions, challenge in determining what is being asked by the question and which elements on the clue are relevant to solve the question. In order to solve a complex question that has multiple facts about the answer in different places of the question a “Decomposition” component was designed to get subclues out of out of the main question or clue, this subclues are analyzed to come up with a candidate answer and all the candidates answer are synthetized afterwards.

   Answering Speed, the range of time in which a question is answered by other participant is between 1 to 6 seconds having an average of 3 seconds. In order to achieve this, Watson uses UIMA AS framework that allows the uses a parallel computation distributing the workload over 2500 computation nodes, using asynchronous messaging by using open JMS standard. This approach allows Watson to answer a question between 3 to 5 seconds.

   Content Acquisition, in order to answer a question Watson needs to have knowledge repository needed to generate candidate answers. Watson ingest this content from a variety of sources and stores it in Hadoop in a distributed way to get computation power out of the capacity of the computation nodes. After having a reasonable baseline corpus, Watson applies an automatic corpus expansion mechanism increasing the content that Watson is going to use at solving question time (run time).

   Confidence estimation, Watson needs a mechanism to obtain the best answer possible given a set of possible answers. Watson uses an hierarchical machine-learning method to combine all the features from the different components involved in the generation of the answer, this to assign a reliable confidence level to a candidate answer.

4. Describe the DeepQA approach developed for Jeopardy and name the six architectural roles that are designed in this model.

   DeepQA started after a failed attempt to adapt PIQUANT (Practical Intelligent Question and Answering Technology) to Jeopardy Problem. IBM rebuilt almost all the work that they did including technical approach, architecture, metrics, evaluation protocols, engineering practices and even the dynamics of the work team. They realized that system-level advances improved the results dramatically, this fact end up supporting the idea that a single extensible architecture will definitely benefit the QA space, and this allows components to be evaluated in a common context defined by the system.
   DeepQA is a massive probabilistic evidence-based architecture, using more than 100 different techniques to work properly. Here is relevant that DeepQA allows to combine all of these techniques, so that overlapping approaches improve accuracy, confidence and speed.
   DeepQA is bundled with methodology, its architecture is not specific to Jeopardy problem. This architecture was successfully applied to Jeopardy and TREC QA with great results (Problem that PIQUANT tried to solve). DeepQA could also by applied to different domains like medicine, enterprise search and gaming.
   The main principles of DeepQA are, massive parallelism, many experts, pervasive confidence estimation, integration of shallow and deep learning.

   The main architectural roles of DeepQA are:

   - Question Analysis.
     Evaluates what the question is asking and how this question will be processed by the rest of the system (Involves question classification and Relation detection).

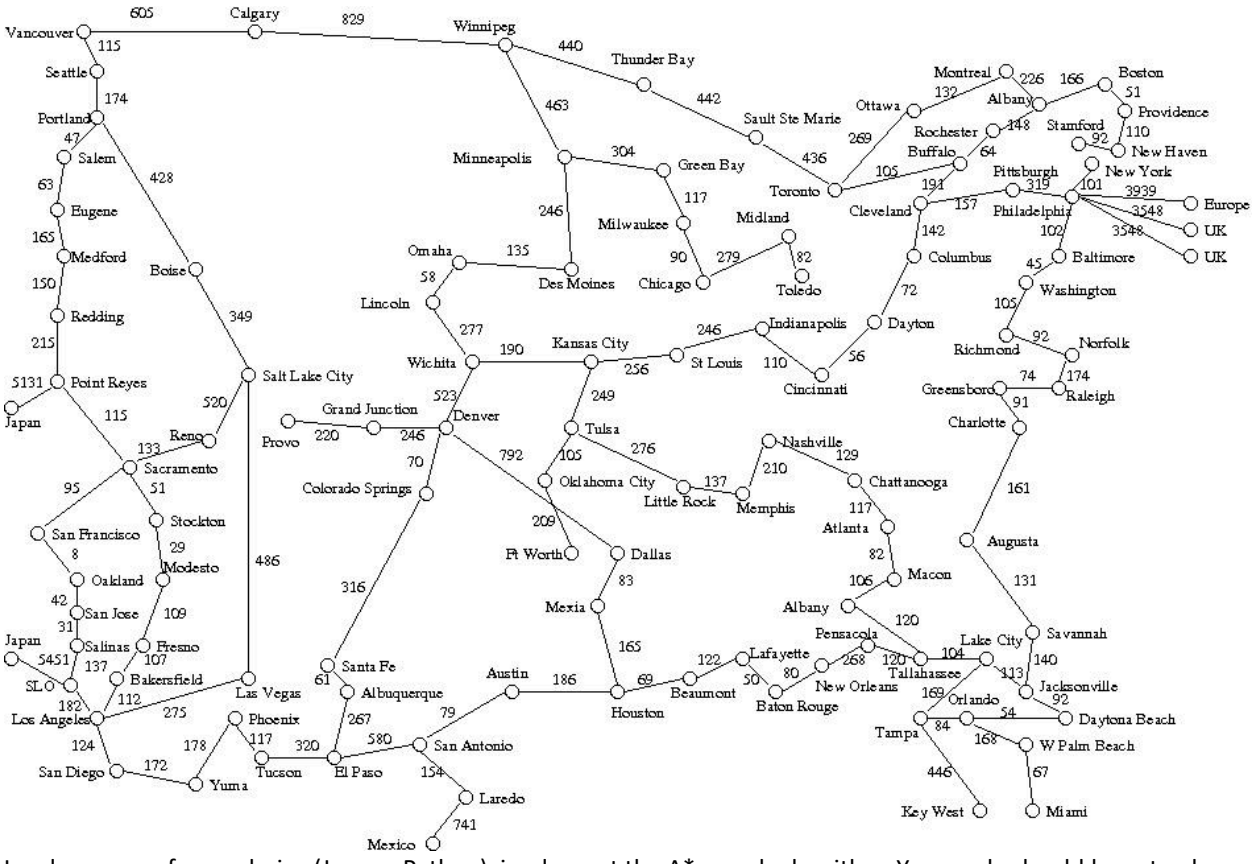
- Query Decomposition.
  Breaks the clue in subclues to better analysis. Deep QA manages parallel decomposable clues, and after that synthetize the final answers.
- Hypothesis Generation.
  Takes the result of Question analysis and generates candidate answers by searching the sources (Primary search and candidate generation).
- Sort Filtering.
  Important step in the process, that filters candidate answers by applying a lightweight scoring algorithms.
- Hypothesis and Evidence Scoring.
  Get evidence to support candidate answers and scores these candidates answers by rigorous process using deep scoring analytics.
- Final Merging and Ranking.
  Evaluate a generated hypothesis based on set of scores to identify the best evidence-supported hypothesis and estimate its confidence (answer merging and Ranking and confidence estimation).

In addition, Content Acquisition is of the utmost importance to establish a baseline corpus and this could be used by DeepQA to perform automatic corpus expansion process.

4. [50 points]

This question concerns route-finding, with comparison of several search algorithms. This time, we're in the U.S. Here's jpg of the map below.

The solution consists of the series of cities a network packet must pass through, each city connected to one or more others by network links of the indicated length. There are no other network links.



In a language of your choice (Java or Python), implement the A* search algorithm. Your code should keep track of nodes expanded and should be able to compute the number of such nodes. Then run your algorithm on the telecom link map.

- Name the source code file with the main function SearchUSA, with the file extension appropriate to the language.
- The inputs should will be given through the command line, similar to Assignment 1.
  - In java:
    % java SearchUSA searchtype srccityname destcityname
  - In C++:
    % mv ./a.out SearchUSA
    % SearchUSA searchtype srccityname destcityname
- The searchtype should be either astar, greedy, or dynamic.
- To save a bit of typing, the network is implemented as Prolog procedures in usroads.pl. This is a Prolog source file, and this assignment does not use Prolog. The file is provided solely as a convenience; you will have to modify it for use with your code. This time we will need the distances, and the longitude/latitude of the cities. (The percent sign (%) is a Prolog comment char, from there to end-of-line.)
- The spelling of srccityname and destcityname must be the as given in usroads.pl. Do NOT change the names from lower case to upper case.
- A node is said to be expanded when it is taken off a data structure and its successors generated.

- Use as a heuristic the straight line distance between cities. The straight-line distance between cities is computed using decimal degrees of latitude and longitude, which also given in the file. There is a complication in computing straight line distance from longitude and latitude that arises because the earth is roughly a sphere, not a cylinder. As a guide, heuristic.pl is another Prolog file with a header comment indicating how the heuristic should work.
- As in Assignment 1, test your code on remote linux machines. Your code must compile without errors (warnings are okay) and must not print statements other than the ones stated above.

Now perform some experiments.

The experiments

1. [10 points] Experiment with executing your implementation of A* to find various paths, until you understand the meaning of the output. Are there any pairs of cities (A,B) for which the algorithm finds a different path from B to A than from A to B? Are there any pairs of cities (A,B) for which the algorithm expands a different total number of nodes from B to A than from A to B?
   Output in such cases should consist of the following:
   - A comma separated list of expanded nodes (the closed list);
   - The number of nodes expanded;
   - A comma-separated list of nodes in the solution path;
   - The number of nodes in the path;
   - The total distance from A to B in the solution path.

   ∇ Are there any pairs of cities (A,B) for which the algorithm finds a different path from B to A than from A to B?

   Yes, there are 2 scenarios, **[japan ↔ desMoines]** and **[indianapolist ↔ keyWest]**.

   AStar, [japan → desMoines].
   - #path = 13
   - Total distance = 8203.0

```
JuanJo:Code jjrivera$ java SearchUSA astar japan desMoines
Expanded nodes      ->[reno, tucson, sanAntonio, bakersfield, salem, japan, yuma, modesto, coloradoSprings,
vancouver, kansasCity, denver, winnipeg, medford, austin, boise, portland, albuquerque, santaFe, seattle,
minneapolis, saltLakeCity, calgary, pointReyes, redding, sanDiego, salinas, elPaso, lasVegas, wichita, losAngeles,
sacramento, phoenix, sanJose, stockton, oakland, sanFrancisco, sanLuisObispo, fresno, eugene]
#expanded           ->40
Solution Path       ->[japan, pointReyes, redding, medford, eugene, salem, portland, seattle, vancouver,
calgary, winnipeg, minneapolis, desMoines]
#path               ->13
Total distance      ->8203.0
```

   AStar, [desMoines → japan].
   - #path = 16
   - Total distance = 8251.0

```
JuanJo:Code jjrivera$ java SearchUSA astar desMoines japan
Expanded nodes      ->[tulsa, tucson, thunderBay, salem, yuma, coloradoSprings, ftWorth, vancouver,
kansasCity, denver, midland, winnipeg, desMoines, medford, portland, albuquerque, santaFe, stLouis, seattle,
minneapolis, chicago, calgary, redding, sanDiego, elPaso, wichita, losAngeles, greenBay, milwaukee, phoenix,
omaha, grandJunction, lincoln, provo, oklahomaCity, sanLuisObispo, eugene]
#expanded           ->37
Solution Path       ->[desMoines, omaha, lincoln, wichita, denver, coloradoSprings, santaFe, albuquerque,
elPaso, tucson, phoenix, yuma, sanDiego, losAngeles, sanLuisObispo, japan]
#path               ->16
Total distance      ->8251.0
```

AStar, [indianapolist → keyWest].

- #path = 21
- Total distance = 2647.0

```
JuanJo:Code jjrivera$ java SearchUSA astar indianapolis keyWest
Expanded nodes      ->[tulsa, tampa, dayton, baltimore, montreal, newHaven, raleigh, miami, littleRock,
norfolk, orlando, lakeCity, richmond, wichita, savannah, charlotte, omaha, providence, albanyGA, albanyNY,
stamford, washington, atlanta, columbus, boston, macon, nashville, toronto, jacksonville, indianapolis, ftWorth,
westPalmBeach, saultSteMarie, kansasCity, philadelphia, daytonaBeach, desMoines, cincinnati, stLouis,
chattanooga, cleveland, tallahassee, ottawa, pittsburgh, memphis, buffalo, newYork, rochester, greensboro,
lincoln, oklahomaCity, augusta, pensacola]
#expanded           ->53
Solution Path       ->[indianapolis, cincinnati, dayton, columbus, cleveland, pittsburgh, philadelphia,
baltimore, washington, richmond, norfolk, raleigh, greensboro, charlotte, augusta, savannah, jacksonville,
daytonaBeach, orlando, tampa, keyWest]
#path               ->21
Total distance      ->2647.0
```

AStar, [keyWest → indianapolis].

- #path = 15
- Total distance = 2647.0

```
JuanJo:Code jjrivera$ java SearchUSA astar keyWest indianapolis
Expanded nodes      ->[tulsa, tampa, dallas, macon, dayton, nashville, baltimore, keyWest, jacksonville,
westPalmBeach, raleigh, miami, kansasCity, littleRock, norfolk, daytonaBeach, philadelphia, houston, austin,
mexia, cincinnati, orlando, stLouis, lakeCity, chattanooga, tallahassee, cleveland, pittsburgh, memphis, lafayette,
richmond, savannah, beaumont, charlotte, newYork, greensboro, batonRouge, albanyGA, newOrleans, augusta,
washington, atlanta, pensacola, columbus]
#expanded           ->44
Solution Path       ->[keyWest, tampa, lakeCity, tallahassee, albanyGA, macon, atlanta, chattanooga,
nashville, memphis, littleRock, tulsa, kansasCity, stLouis, indianapolis]
#path               ->15
Total distance      ->2647.0
```

AStar uses as cost function: f(n) = g(n) + h(n) where g(n) is the path cost from the starting node to the current node and h(n) is the estimate from current node to goal.
AStar is optimal when heuristic h(n) is admissible and consistent, h(n) is admissible when it never overestimates the cost to reach the goal and h(n) is constant when h(n)<= h(successor n) + g(n to successor n).
In these scenarios, h(n) is not getting a value lesser than the real one (provided by map (g(n)), therefore AStar is not optimal in those scenarios.

Case **[japan → desMoines]**:

Distances between japan and its neighbors are considerably greater than all the other distances between cities in the map (5000/japan vs {10…800}/other cities), this could cause and strange behavior if heuristic(h(n)) does not have enough quality.

In this case what happens in the graph search for desMoines → japan is that frontier includes pointReyes and slo (japan neighbors) with the following cost values:

f(pointReyes) = g(pointReyes) + h(pointReyes) where
g(pointReyes) = 3071, h(pointReyes) = 5419 (map value 5131 (real value)/ h(n) does not provide high quality estimate for this scenario)
f(pointReyes) = 8490

f(slo) = g(slo) + h(slo) where

g(slo) = 2800, h(slo) = 5635 (map value 5451 (real value)/h(n) does not provide high quality estimate for this scenario)
f(slo) = 8435

f(slo) has higher priority than f(pointReyes), so "slo" will be expanded and reach goal japan, when distance is calculated g(slo) + map distance to japan(goal) = 2800 + 5451 = 8251.

If h(pointReyes) has better quality (better estimation) we could take this node from the frontier and expanded, it having as result: distance = g(pointReyes) + distance to japan(goal) = 3071 +5131 = 8202. This distance is better than the result of the AStar search for desMoines → japan.

Case **[indianapolis → keyWest]**:

This scenario is different than the one mentioned above. Here, the heuristic is working properly for the nodes that the algorithm is exploring, therefore AStar is having the optimal expected behavior by minimizing f(n) cost. At one point during the execution of AStar, the algorithm has tampa in the frontier that was expanded from orlando(having 20 nodes on the path) with f(n) = 2441. After that moment in the time, lakeCity tries to expand tampa as well (having 14 nodes in the path, less than Orlando) but with the same f(n) = 2441.
Given that AStar has as main function f(n) =g(n)+h(n) and it is not taking as part of f(n) the number of nodes in the path, AStar is going to keep tampa in the frontier (from Orlando/20 nodes), this happens given the nature of AStar, that when finds the same node already explored, only updates parents when the node to be expanded has cost f(n) that is lesser than the cost of the node that is already in frontier). Next iteration of the algorithm will poll keyWest from frontier and reach goal, with an optimal distance (but not with minimum number of nodes in the path).

∇ Are there any pairs of cities (A,B) for which the algorithm expands a different total number of nodes from B to A than from A to B?

Yes, [toledo ↔ calgary].

Astar, [toledo → calgary].

- #expanded = 7

```
JuanJo:Code jjrivera$ java SearchUSA astar toledo calgary
Expanded nodes      ->[midland, winnipeg, greenBay, chicago, minneapolis, milwaukee, toledo]
#expanded           ->7
Solution Path       ->[toledo, midland, chicago, milwaukee, greenBay, minneapolis, winnipeg, calgary]
#path               ->8
Total distance      ->2164.0
```

Astar, [calgary → toledo].

- #expanded = 10

```
JuanJo:Code jjrivera$ java SearchUSA astar  calgary toledo
Expanded nodes      ->[thunderBay, midland, winnipeg, desMoines, greenBay, minneapolis, chicago,
saultSteMarie, calgary, milwaukee]
#expanded           ->10
Solution Path       ->[calgary, winnipeg, minneapolis, greenBay, milwaukee, chicago, midland, toledo]
#path               ->8
Total distance      ->2164.0
```

Neighbors for toledo → calgary appear in a different way than neighbors for calgary → toledo, this causes a different node expansion behavior. We also observe that AStar is optimal in both cases given that is returning the same distance, that is the priority on the frontier.

2. [10 points] Change your code so as to implement greedy search, as discussed in the web notes.

   Greedy, [santFe ↔ dallas].

```
JuanJo:Code jjrivera$ java SearchUSA greedy santaFe dallas
Expanded nodes      ->[sanAntonio, elPaso, austin, houston, mexia, albuquerque, santaFe]
#expanded           ->7
Solution Path       ->[santaFe, albuquerque, elPaso, sanAntonio, austin, houston, mexia, dallas]
#path               ->8
Total distance      ->1421.0
```

```
JuanJo:Code jjrivera$ java SearchUSA greedy  dallas santaFe
Expanded nodes      ->[dallas, denver, coloradoSprings]
#expanded           ->3
Solution Path       ->[dallas, denver, coloradoSprings, santaFe]
#path               ->4
Total distance      ->1178.0
```

3. [10 points] Do enough exploration to find at least one path that is longer using greedy search than that found using A*, or to satisfy yourself that there are no such paths. Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.
   Output in such cases should consist of the following:
   - A comma separated list of expanded nodes (the closed list);
   - The number of nodes expanded;
   - A comma-separated list of nodes in the solution path;
   - The number of nodes in the path;
   - The total distance from A to B in the solution path.

   ∇ Do enough exploration to find at least one path that is longer using greedy search than that found using A*, or to satisfy yourself that there are no such paths

   Greedy, [tulsa → baltimore].
   - Total distance = 2407.0

```
JuanJo:Code jjrivera$ java SearchUSA greedy tulsa baltimore
Expanded nodes      ->[tulsa, memphis, macon, nashville, richmond, jacksonville, savannah, raleigh, charlotte,
littleRock, greensboro, norfolk, albanyGA, chattanooga, lakeCity, augusta, washington, atlanta, tallahassee]
#expanded           ->19
Solution Path       ->[tulsa, littleRock, memphis, nashville, chattanooga, atlanta, macon, albanyGA,
tallahassee, lakeCity, jacksonville, savannah, augusta, charlotte, greensboro, raleigh, norfolk, richmond,
washington, baltimore]
#path               ->20
Total distance      ->2407.0
```

AStar, [tulsa → baltimore].

- Total distance = 1709.0

```
JuanJo:Code jjrivera$ java SearchUSA astar tulsa baltimore
Expanded nodes      ->[tulsa, pittsburgh, memphis, dayton, macon, nashville, buffalo, wichita, indianapolis,
ftWorth, kansasCity, littleRock, rochester, philadelphia, cincinnati, stLouis, oklahomaCity, chattanooga, atlanta,
cleveland, columbus]
#expanded           ->21
Solution Path       ->[tulsa, kansasCity, stLouis, indianapolis, cincinnati, dayton, columbus, cleveland,
pittsburgh, philadelphia, baltimore]
#path               ->11
Total distance      ->1709.0
```

Greedy takes as priority only the distance to goal. In contrast, AStar uses a combination of distance to goal(estimate/heuristic) + path cost. This could be also understood using the "rubber band" analogy. In the case of Greedy, the rubber band ties current node and goal node, meaning that Greedy is going to expand only the nodes that are closer to goal according to the estimated cost. On the other hand, AStar has two has 2 rubber bands, one ties current node and goal (Greedy approach) and the other one ties current node and start node (Dynamic programing approach).
Greedy is not optimal but often is efficient (lesser expanded nodes).

∇ Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.

Path	[atlanta, macon, albanyGA, tallahassee, pensacola, newOrleans, batonRouge, lafayette, beaumont, houston]
Distance	1017.0

Greedy, [atlanta → houston].

- #expanded = 19

```
JuanJo:Code jjrivera$ java SearchUSA greedy atlanta houston
Expanded nodes      ->[tulsa, memphis, lafayette, macon, nashville, wichita, ftWorth, beaumont, kansasCity,
littleRock, batonRouge, albanyGA, stLouis, newOrleans, chattanooga, oklahomaCity, atlanta, tallahassee,
pensacola]
#expanded           ->19
Solution Path       ->[atlanta, macon, albanyGA, tallahassee, pensacola, newOrleans, batonRouge, lafayette,
beaumont, houston]
#path               ->10
Total distance      ->1017.0
```

AStar, [atlanta → houston].

- #expanded = 13

```
JuanJo:Code jjrivera$ java SearchUSA astar atlanta houston
Expanded nodes      ->[memphis, lafayette, macon, nashville, beaumont, littleRock, batonRouge, albanyGA,
newOrleans, chattanooga, atlanta, tallahassee, pensacola]
#expanded           ->13
Solution Path       ->[atlanta, macon, albanyGA, tallahassee, pensacola, newOrleans, batonRouge, lafayette,
beaumont, houston]
#path               ->10
Total distance      ->1017.0
```

This is could be explained with the rubber band analogy from before, the attraction that the goal produces on the current node on Greedy causes this behavior for a set of scenarios in the map(graph), Greedy expands more nodes than AStar because only cares about distance to goal

4. [10 points] Change your code so as to implement dynamic programming search, as discussed in the web notes.

   Dynamic, [santFe ↔ dallas].

```
JuanJo:Code jjrivera$ java SearchUSA dynamic santaFe dallas
Expanded nodes      ->[sanDiego, tucson, sanAntonio, laredo, elPaso, yuma, wichita, coloradoSprings, kansasCity,
denver, phoenix, grandJunction, austin, houston, albuquerque, provo, santaFe]
#expanded           ->17
Solution Path       ->[santaFe, coloradoSprings, denver, dallas]
#path               ->4
Total distance      ->1178.0
```

```
JuanJo:Code jjrivera$ java SearchUSA dynamic dallas santaFe
Expanded nodes      ->[dallas, lafayette, sanAntonio, laredo, elPaso, jacksonville, coloradoSprings, beaumont,
denver, grandJunction, houston, austin, batonRouge, mexia, albanyGA, newOrleans, lakeCity, tallahassee, pensacola]
#expanded           ->19
Solution Path       ->[dallas, denver, coloradoSprings, santaFe]
#path               ->4
Total distance      ->1178.0
```

5. [10 points] Do enough exploration to find at least one path that is longer using dynamic programming than that found using A*, or to satisfy yourself that there are no such paths. Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.
   Output in such cases should consist of the following:
   - A comma separated list of expanded nodes (the closed list);
   - The number of nodes expanded;
   - A comma-separated list of nodes in the solution path;
   - The number of nodes in the path;
   - The total distance from A to B in the solution path.

   ∇ Do enough exploration to find at least one path that is longer using dynamic programming than that found using A*, or to satisfy yourself that there are no such paths.

   There is no path that satisfies this condition.

   As dynamic search performs an exhaustive exploration, it is going to get always the shortest path (distance), meaning that other search algorithms are going to find the same path at best.

∇ Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.

Path	[santaFe, coloradoSprings, denver, dallas]
Distance	1178.0

Dynamic, [santFe → dallas].

- #expanded = 17

```
JuanJo:Code jjrivera$ java SearchUSA dynamic santaFe dallas
Expanded nodes      ->[sanDiego, tucson, sanAntonio, laredo, elPaso, yuma, wichita, coloradoSprings,
kansasCity, denver, phoenix, grandJunction, austin, houston, albuquerque, provo, santaFe]
#expanded           ->17
Solution Path       ->[santaFe, coloradoSprings, denver, dallas]
#path               ->4
Total distance      ->1178.0
```

AStar, [santaFe → dallas].

- #expanded = 7

```
JuanJo:Code jjrivera$ java SearchUSA astar santaFe dallas
Expanded nodes      ->[denver, sanAntonio, elPaso, austin, albuquerque, santaFe, coloradoSprings]
#expanded           ->7
Solution Path       ->[santaFe, coloradoSprings, denver, dallas]
#path               ->4
Total distance      ->1178.0
```

Dynamic programing is based its priority on the path distance, this strategy is considered an uninformed strategy given that it doesn't have information about the goal. AStar uses a combination of path distances and distance to the goal that helps to improve its performance by reducing the expanded nodes in comparison to dynamic. If AStar has a high-quality heuristic, AStar is optimal.

6. As part of your answer, compare the solution paths and explain what happened, especially any weird behavior you might detect.