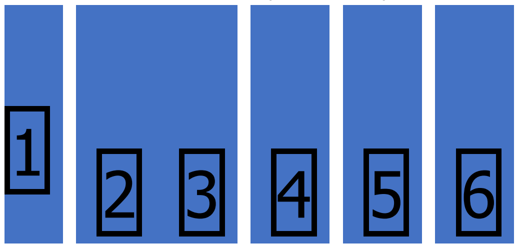
**The University of Texas at Dallas**
**CS 6364**
**Artificial Intelligence**
**Fall 2020**
**Instructor: Dr. Sanda Harabagiu**
**Grader/ Teaching Assistant: Maxwell Weinzierl**

**Homework 1: 100 points (70 points extra-credit)**
**Issued September 2, 2020**
**Due September 21, 2020 before midnight**

**Name: Shankaranarayanan Kallidaikuruchi Ramakrishnan**
**NET-ID: sxk190109**

**PROBLEM 1:** Intelligent Agents (**15 points**)

Enhance the Kitty robot discussed in class to a model-based agent in which the state has 3 components: [position happiness, battery]; where the battery has a value of 100 for FULL initially and 0 when it is discharged. Also add an admissible new action: TURNAROUND. The possible positions of Kitty are:



You are asked to:

1. Describe the entire state space (**2 points**)

**Answer:**
The state space consists of 3 components [position, happiness, battery] of the kitty robot.
Each state contains the following
Position: {1,2,3,4,5,6}
Happiness: {Very happy, Happy, Sad, Curios}
Battery: {0,1,2, 3,…,100}
Total number of possible states = 6 * 4 * 101 = 2424

2. Write a new State-Update function that changes the value of the battery by -1 for every new change of position by walking; by -4 when it turns around; by -5 when it bumps against the wall; by -3 every time Kitty purrs and by -2 every time it meows You should consider that when the battery is discharged, no more actions are possible. (**10 points**)

**Answer:**

```
State ← UPDATE-STATE(previous-state, percept, action)
  IF Battery != 0 # Only update state when battery is not discharged
    Happiness +1 ←UPDATE-KITTY[Happines,Clap]
    Happiness +2 ←UPDATE-KITTY[Happines,Pet]
    Happiness -1 ←UPDATE-KITTY[Happines,Bump]
    Position-1 ←UPDATE-KITTY[Position,Walk]
    Position ← UPDATE-KITTY[Position,Purr|Meow|Blink|Stop|Start]
    Battery-1 ←UPDATE-KITTY[Battery, Walk]
    Battery-4 ←UPDATE-KITTY[Battery, Turn around]
    Battery-5 ←UPDATE-KITTY[Battery, Wall]
    Battery-3 ←UPDATE-KITTY[Battery,Purrs]
    Battery-2 ←UPDATE-KITTY[Battery,Meow]
```

3. Write a utility function that combines the happiness with the state of the battery. (**3 points**)

**Answer:**

We can map happiness value to a number and sum it with battery level which gives maximum utility value with kitty robot is both happy and is at maximum.

```
U ← UTILITY-FUNCTION(State)
        return Happiness + Battery
```

**<u>PROBLEM 2:</u>** Problem Formulation for Search (**55 points**)

---

The Missionaries and Cannibals Problem is stated as follows. Three missionaries and three cannibals are on one side of the river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries outnumbered by cannibals in that place.

a. Formulate the search problem formally. State clearly how you represent states, (including the initial state and the goal state(s)), what actions are available in each state, including the cost of each action. (**15 points**)

**Answer:**

S0: (initial state) (3,3,1)

Si: (State i) (xi,yi,zi)

Where xi – Number of missionaries on the left side xi ∈ [0,3]

Yi – Number of cannibals on the left side yi ∈ [0,3]

Zi – 1 if boat is on left side 0 if boat is on right side zi ∈ {0,1}

Sg: (Goal state) (0,0,0)

Actions:

M – Move 1 missionary from one side to the other

MM – Move 2 missionaries from one side to the other

C – Move 1 cannibal from one side to the other

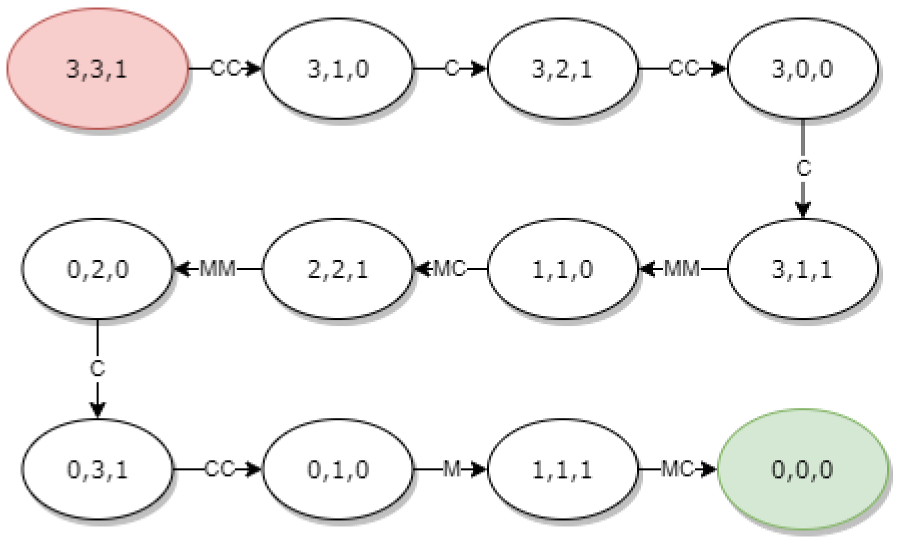
MC – Move 1 cannibal and 1 missionary from one side to the other

CC – Move 2 cannibals from one side to the other

Cost of each action = 1

b. Sketch a solution of the problem. Show the path and its cost. Describe the search strategy that you have used and <u>motivate it.</u> (**15 points** for the manual solution; **5 points** for its motivation). **TOTAL 20 points**

**Solution:**



Breadth first search is used as a search strategy as it yields shortest path to the goal node because BFS will explore the nodes closest to the initial state.

*<u>Programming Assignment for Problem 2:</u>*

1. Use the implementations of the search strategies available in the aima code; e.g. https://github.com/aimacode/aima-java to write the program that will search for the solution to Missionaries and Cannibals Problem. (**10 points**)

2. Provide the path to the solution generated by your code, indicating each state it has visited when using: (a) uniform-cost search; (b) iterative deepening search; (c) greedy best-first search; (d) A* search and (e) recursive best-first search (**1 point/strategy)**. Describe clearly how you have implemented each state in the search space. (**5 points)**
**TOTAL: 10 points**
*Extra-credit:* List the content of the frontier and the list of explored nodes for the first 5 steps of each of the strategies used in 2. (**2 points/step/strategy)**
**TOTAL: 50 points ONLY if it is possible to comprehend (1) the current node; (2) the content of the frontier; (3) the content of the list of explored/expanded nodes; (4) the children of the current node for each step.**

**Code output:**

```
==== Uniform cost search
Step 1 Current Node = MACState(3, 3, 1)
Step 1 Frontier = [MACState(2, 3, 0), MACState(1, 3, 0),
MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 1 Explored = [MACState(3, 3, 1)]
Step 1 Children of current node = [MACState(2, 3, 0), MACState(1,
3, 0), MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 2 Current Node = MACState(2, 3, 0)
Step 2 Frontier = [MACState(2, 2, 0), MACState(1, 3, 0),
MACState(3, 1, 0), MACState(3, 2, 0)]
Step 2 Explored = [MACState(3, 3, 1), MACState(2, 3, 0)]
Step 2 Children of current node = []
Step 3 Current Node = MACState(2, 2, 0)
Step 3 Frontier = [MACState(3, 2, 0), MACState(1, 3, 0),
MACState(3, 1, 0), MACState(3, 2, 1), MACState(2, 3, 1),
MACState(3, 3, 1)]
Step 3 Explored = [MACState(3, 3, 1), MACState(2, 3, 0),
MACState(2, 2, 0)]
Step 3 Children of current node = [MACState(3, 2, 1), MACState(2,
3, 1), MACState(3, 3, 1)]
Step 4 Current Node = MACState(3, 2, 0)
Step 4 Frontier = [MACState(1, 3, 0), MACState(3, 3, 1),
MACState(3, 1, 0), MACState(3, 2, 1), MACState(2, 3, 1),
MACState(3, 3, 1)]
Step 4 Explored = [MACState(3, 3, 1), MACState(2, 3, 0),
MACState(2, 2, 0), MACState(3, 2, 0)]
Step 4 Children of current node = [MACState(3, 3, 1)]
Step 5 Current Node = MACState(1, 3, 0)
Step 5 Frontier = [MACState(3, 1, 0), MACState(3, 3, 1),
MACState(3, 3, 1), MACState(3, 2, 1), MACState(2, 3, 1)]
```

```
Step 5 Explored = [MACState(3, 3, 1), MACState(2, 3, 0),
MACState(2, 2, 0), MACState(3, 2, 0), MACState(1, 3, 0)]
Step 5 Children of current node = []
---------------------------------------------
Solution = MACState(3, 3, 1) - Action[name==CC] -> MACState(3, 1,
0) - Action[name==C] -> MACState(3, 2, 1) - Action[name==CC] ->
MACState(3, 0, 0) - Action[name==C] -> MACState(3, 1, 1) -
Action[name==MM] -> MACState(1, 1, 0) - Action[name==MC] ->
MACState(2, 2, 1) - Action[name==MM] -> MACState(0, 2, 0) -
Action[name==C] -> MACState(0, 3, 1) - Action[name==CC] ->
MACState(0, 1, 0) - Action[name==M] -> MACState(1, 1, 1) -
Action[name==MC] -> MACState(0, 0, 0)
Path cost = 11.0

==== A* search
Step 1 Current Node = MACState(3, 3, 1)
Step 1 Frontier = [MACState(1, 3, 0), MACState(2, 2, 0),
MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 3, 0)]
Step 1 Explored = [MACState(3, 3, 1)]
Step 1 Children of current node = [MACState(2, 3, 0), MACState(1,
3, 0), MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 2 Current Node = MACState(1, 3, 0)
Step 2 Frontier = [MACState(2, 2, 0), MACState(2, 3, 0),
MACState(3, 1, 0), MACState(3, 2, 0)]
Step 2 Explored = [MACState(3, 3, 1), MACState(1, 3, 0)]
Step 2 Children of current node = []
Step 3 Current Node = MACState(2, 2, 0)
Step 3 Frontier = [MACState(3, 1, 0), MACState(2, 3, 0),
MACState(3, 2, 0), MACState(3, 2, 1), MACState(2, 3, 1),
MACState(3, 3, 1)]
Step 3 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 2, 0)]
Step 3 Children of current node = [MACState(3, 2, 1), MACState(2,
3, 1), MACState(3, 3, 1)]
Step 4 Current Node = MACState(3, 1, 0)
Step 4 Frontier = [MACState(2, 3, 0), MACState(3, 2, 1),
MACState(3, 2, 0), MACState(3, 3, 1), MACState(2, 3, 1),
MACState(3, 3, 1), MACState(3, 2, 1)]
Step 4 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 2, 0), MACState(3, 1, 0)]
Step 4 Children of current node = [MACState(3, 3, 1), MACState(3,
2, 1)]
```

```
Step 5 Current Node = MACState(2, 3, 0)
Step 5 Frontier = [MACState(3, 2, 0), MACState(3, 2, 1),
MACState(3, 2, 1), MACState(3, 3, 1), MACState(2, 3, 1),
MACState(3, 3, 1)]
Step 5 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 2, 0), MACState(3, 1, 0), MACState(2, 3, 0)]
Step 5 Children of current node = []
---------------------------------------------
Solution = MACState(3, 3, 1) - Action[name==MC] -> MACState(2, 2,
0) - Action[name==M] -> MACState(3, 2, 1) - Action[name==CC] ->
MACState(3, 0, 0) - Action[name==C] -> MACState(3, 1, 1) -
Action[name==MM] -> MACState(1, 1, 0) - Action[name==MC] ->
MACState(2, 2, 1) - Action[name==MM] -> MACState(0, 2, 0) -
Action[name==C] -> MACState(0, 3, 1) - Action[name==CC] ->
MACState(0, 1, 0) - Action[name==M] -> MACState(1, 1, 1) -
Action[name==MC] -> MACState(0, 0, 0)
Path cost = 11.0

==== Greedy Best first Search
Step 1 Current Node = MACState(3, 3, 1)
Step 1 Frontier = [MACState(1, 3, 0), MACState(2, 2, 0),
MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 3, 0)]
Step 1 Explored = [MACState(3, 3, 1)]
Step 1 Children of current node = [MACState(2, 3, 0), MACState(1,
3, 0), MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 2 Current Node = MACState(1, 3, 0)
Step 2 Frontier = [MACState(2, 2, 0), MACState(2, 3, 0),
MACState(3, 1, 0), MACState(3, 2, 0)]
Step 2 Explored = [MACState(3, 3, 1), MACState(1, 3, 0)]
Step 2 Children of current node = []
Step 3 Current Node = MACState(2, 2, 0)
Step 3 Frontier = [MACState(3, 1, 0), MACState(2, 3, 0),
MACState(3, 2, 0), MACState(3, 2, 1), MACState(2, 3, 1),
MACState(3, 3, 1)]
Step 3 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 2, 0)]
Step 3 Children of current node = [MACState(3, 2, 1), MACState(2,
3, 1), MACState(3, 3, 1)]
Step 4 Current Node = MACState(3, 1, 0)
Step 4 Frontier = [MACState(2, 3, 0), MACState(3, 2, 1),
MACState(3, 2, 0), MACState(3, 3, 1), MACState(2, 3, 1),
MACState(3, 3, 1), MACState(3, 2, 1)]
```

```
Step 4 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 2, 0), MACState(3, 1, 0)]
Step 4 Children of current node = [MACState(3, 3, 1), MACState(3,
2, 1)]
Step 5 Current Node = MACState(2, 3, 0)
Step 5 Frontier = [MACState(3, 2, 1), MACState(3, 2, 1),
MACState(3, 2, 0), MACState(3, 3, 1), MACState(2, 3, 1),
MACState(3, 3, 1)]
Step 5 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 2, 0), MACState(3, 1, 0), MACState(2, 3, 0)]
Step 5 Children of current node = []
---------------------------------------------
Solution = MACState(3, 3, 1) - Action[name==CC] -> MACState(3, 1,
0) - Action[name==C] -> MACState(3, 2, 1) - Action[name==CC] ->
MACState(3, 0, 0) - Action[name==C] -> MACState(3, 1, 1) -
Action[name==C] -> MACState(3, 0, 0) - Action[name==C] ->
MACState(3, 1, 1) - Action[name==MM] -> MACState(1, 1, 0) -
Action[name==MC] -> MACState(2, 2, 1) - Action[name==MM] ->
MACState(0, 2, 0) - Action[name==C] -> MACState(0, 3, 1) -
Action[name==CC] -> MACState(0, 1, 0) - Action[name==M] ->
MACState(1, 1, 1) - Action[name==MC] -> MACState(0, 0, 0)
Path cost = 13.0

==== RBFS
Step 1 Current Node = MACState(3, 3, 1)
Step 1 Children of current node = [MACState(2, 3, 0), MACState(1,
3, 0), MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 1 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(1, 3, 0), MACState(2, 3, 0), MACState(3, 1, 0)]
Step 1 Explored = [MACState(3, 3, 1)]
Step 2 Current Node = MACState(1, 3, 0)
Step 2 Children of current node = []
Step 2 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(2, 3, 0), MACState(3, 1, 0)]
Step 2 Explored = [MACState(1, 3, 0), MACState(3, 3, 1)]
Step 3 Current Node = MACState(3, 1, 0)
Step 3 Children of current node = [MACState(3, 3, 1), MACState(3,
2, 1)]
Step 3 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(3, 2, 1), MACState(3, 3, 1), MACState(2, 3, 0)]
Step 3 Explored = [MACState(3, 1, 0), MACState(3, 3, 1)]
Step 4 Current Node = MACState(2, 2, 0)
Step 4 Children of current node = [MACState(3, 2, 1), MACState(2,
3, 1), MACState(3, 3, 1)]
Step 4 Frontier = [MACState(3, 2, 0), MACState(3, 2, 1),
MACState(3, 3, 1), MACState(2, 3, 1), MACState(2, 3, 0)]
Step 4 Explored = [MACState(3, 3, 1), MACState(2, 2, 0)]
Step 5 Current Node = MACState(2, 3, 0)
Step 5 Children of current node = []
Step 5 Frontier = [MACState(3, 2, 0), MACState(3, 2, 1),
MACState(3, 3, 1), MACState(2, 3, 1)]
Step 5 Explored = [MACState(3, 3, 1), MACState(2, 3, 0)]
---------------------------------------------
Solution = MACState(3, 3, 1) - Action[name==MC] -> MACState(2, 2,
0) - Action[name==M] -> MACState(3, 2, 1) - Action[name==CC] ->
MACState(3, 0, 0) - Action[name==C] -> MACState(3, 1, 1) -
Action[name==MM] -> MACState(1, 1, 0) - Action[name==MC] ->
MACState(2, 2, 1) - Action[name==MM] -> MACState(0, 2, 0) -
Action[name==C] -> MACState(0, 3, 1) - Action[name==CC] ->
MACState(0, 1, 0) - Action[name==M] -> MACState(1, 1, 1) -
Action[name==MC] -> MACState(0, 0, 0)
Path cost = 11.0

==== Iterative Deepening Search
Step 1 Current Node = MACState(3, 3, 1)
Step 1 Children of current node = [MACState(2, 3, 0), MACState(1,
3, 0), MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 1 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(1, 3, 0), MACState(2, 3, 0), MACState(3, 1, 0)]
Step 1 Explored = [MACState(3, 3, 1)]
Step 2 Current Node = MACState(3, 3, 1)
Step 2 Children of current node = [MACState(2, 3, 0), MACState(1,
3, 0), MACState(3, 1, 0), MACState(3, 2, 0), MACState(2, 2, 0)]
Step 2 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(1, 3, 0), MACState(2, 3, 0), MACState(3, 1, 0)]
Step 2 Explored = [MACState(3, 3, 1)]
Step 3 Current Node = MACState(2, 3, 0)
Step 3 Children of current node = []
Step 3 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(1, 3, 0), MACState(3, 1, 0)]
Step 3 Explored = [MACState(3, 3, 1), MACState(2, 3, 0)]
Step 4 Current Node = MACState(1, 3, 0)
Step 4 Children of current node = []
Step 4 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(3, 1, 0)]
Step 4 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 3, 0)]
Step 5 Current Node = MACState(3, 1, 0)
Step 5 Children of current node = [MACState(3, 3, 1), MACState(3,
2, 1)]
Step 5 Frontier = [MACState(3, 2, 0), MACState(2, 2, 0),
MACState(3, 2, 1), MACState(3, 3, 1)]
Step 5 Explored = [MACState(3, 3, 1), MACState(1, 3, 0),
MACState(2, 3, 0), MACState(3, 1, 0)]
---------------------------------------------
Solution = MACState(3, 3, 1) - Action[name==CC] -> MACState(3, 1,
0) - Action[name==C] -> MACState(3, 2, 1) - Action[name==CC] ->
MACState(3, 0, 0) - Action[name==C] -> MACState(3, 1, 1) -
Action[name==MM] -> MACState(1, 1, 0) - Action[name==MC] ->
MACState(2, 2, 1) - Action[name==MM] -> MACState(0, 2, 0) -
Action[name==C] -> MACState(0, 3, 1) - Action[name==CC] ->
MACState(0, 1, 0) - Action[name==M] -> MACState(1, 1, 1) -
Action[name==MC] -> MACState(0, 0, 0)
Path cost = 11.0
```

**PROBLEM 3:** Searching for Road Trips in the U.S.A. (**30 points**) Considering the following table:

*Table of direct/flight distances to Dallas (in miles).*

| | |
|---|---|
| Austin | 182 |
| Charlotte | 929 |
| San Francisco | 1230 |
| Los Angeles | 1100 |
| New York | 1368 |
| Chicago | 800 |
| Seattle | 1670 |
| Santa Fe | 560 |
| Bakersville | 1282 |
| Boston | 1551 |

1/ You contemplate to search for a trip back to Dallas from Seattle, having access to a road map which should be implemented as a graph. Use the aima code to find the solution and return a simulation of the RBFS strategy by generating automatically the following five values: (1) *f_limit*; (2) *best*; (3) *alternative*; (4) *current-city*; and (5) *next-city* for each node visited. (**10 points)**

**Code output:**

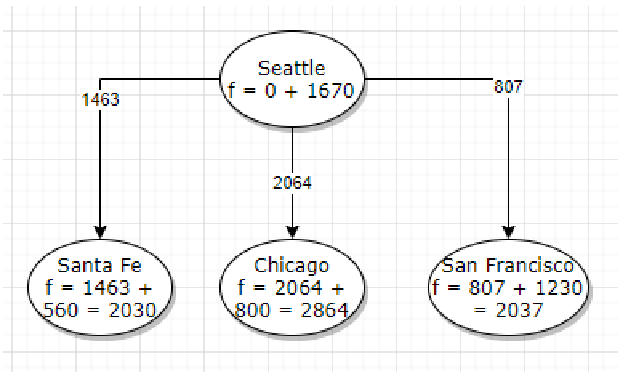
```
==== RBFS
fLimit = INFINITY
Current City = Seattle
Best = 2023.0
Alternative = 2037.0
Next-city = Santa Fe
---------------------------------------------
fLimit = 2037.0
Current City = Santa Fe
Best = 2037.0
Alternative = 2103.0
Next-city = San Francisco
---------------------------------------------
fLimit = 2103.0
Current City = San Francisco
Best = 2103.0
Alternative = 2290.0
Next-city = Santa Fe
---------------------------------------------
fLimit = 2290.0
Current City = Santa Fe
Best = 2103.0
Alternative = 3535.0
Next-city = Dallas
---------------------------------------------
Solution = Seattle - Action[name==moveTo, location==Santa Fe] ->
Santa Fe - Action[name==moveTo, location==Dallas] -> Dallas
Path cost = 2103.0
```

2/ Find the same solution manually and specify how you have computed the following five values: (1*) f_limit*; (2) *best*; (3) *alternative*; (4) *current-city*; and (5) *next-city* for each node visited. Specify at each step the current node and the next node. (**5 points)**

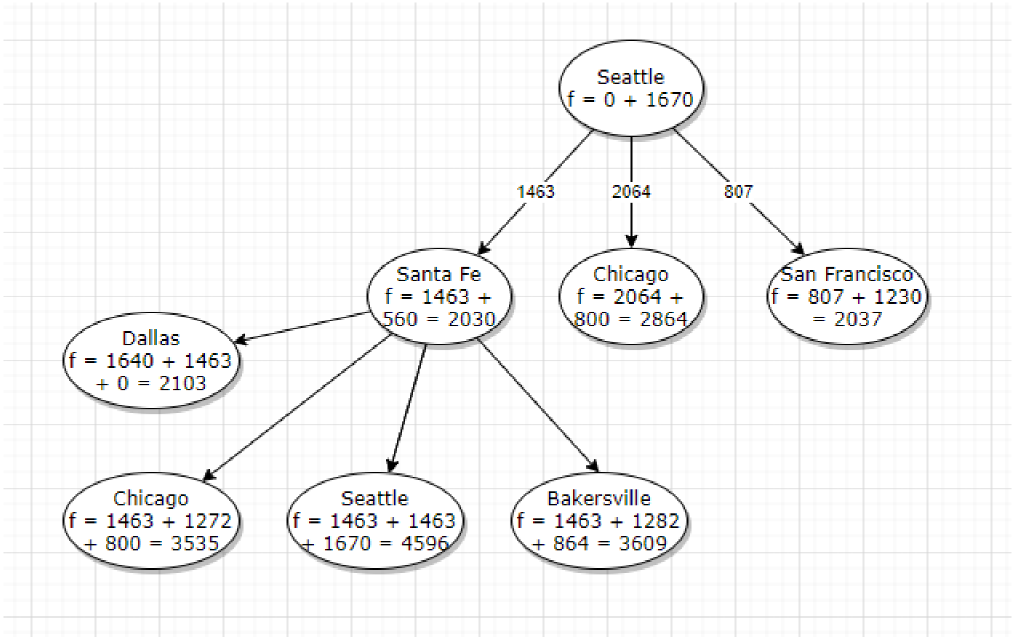
```
The road graph is (in miles):
_____________________________________
Los Angeles --- San Francisco ::: 383
Los Angeles --- Austin        ::: 1377
Los Angeles --- Bakersville   ::: 153
San Francisco --- Bakersville  ::: 283
San Francisco --- Seattle  ::: 807
Seattle  --- Santa Fe  ::: 1463
Seattle --- Chicago  :::  2064
Bakersville -- Santa Fe ::: 864
Austin --- Dallas ::: 195
Santa Fe --- Dallas ::: 640
```

```
Boston --- Austin  ::: 1963
Dallas --- New York  ::: 1548
Austin --- Charlotte  ::: 1200
Charlotte -- New York ::: 634
New York --- Boston ::: 225
Boston --- Chicago ::: 983
Chicago -- Santa Fe ::: 1272 Boston
--- San Francisco ::: 3095
```

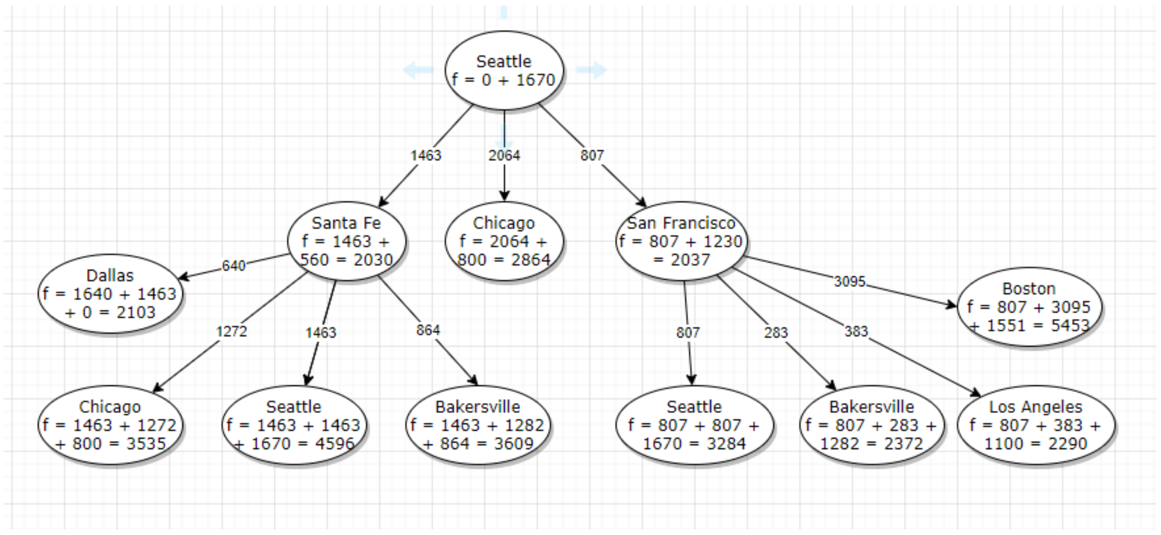
---



| | |
|---|---|
| F_limit | ∞ |
| Best | 2030 |
| Alternative | 2037 |
| Current | Seattle |
| Next | Santa Fe |



|  |  |  |
|---|---|---|
| F_limit | ∞ | 2037 |
| Best | 2030 | 2103 |
| Alternative | 2037 | 3535 |
| Current | Seattle | Santa Fe |
| Next | Santa Fe | Fail |



Seattle f = 0 + 1670
1463
2064
807
Santa Fe f = 1463 + 560 = 2030
Chicago f = 2064 + 800 = 2864
San Francisco f = 807 + 1230 = 2037
640
Dallas f = 1640 + 1463 + 0 = 2103
1272
1463
864
Chicago f = 1463 + 1272 + 800 = 3535
Seattle f = 1463 + 1463 + 1670 = 4596
Bakersville f = 1463 + 1282 + 864 = 3609
3095
Boston f = 807 + 3095 + 1551 = 5453
807
283
383
Seattle f = 807 + 807 + 1670 = 3284
Bakersville f = 807 + 283 + 1282 = 2372
Los Angeles f = 807 + 383 + 1100 = 2290

|  |  |  |  |
|---|---|---|---|
| F_limit | ∞ | 2037 | 2103 |
| Best | 2030 | 2103 | 2290 |
| Alternative | 2037 | -- | -- |
| Current | Seattle | Santa Fe | San Francisco |
| Next | Santa Fe | Fail | Fail |

Back track to Santa Fe

| | | | | |
|---|---|---|---|---|
| F_limit | ∞ | 2037 | 2103 | 2290 |
| Best | 2030 | 2103 | 2290 | 2103 |
| Alternative | 2037 | -- | -- | 3535 |
| Current | Seattle | Santa Fe | San Francisco | Santa Fe |
| Next | Santa Fe | Fail | Fail | Dallas |

```
Goal Test = True
```
Solution Path = Seattle -> Santa Fe -> Dallas; Cost = 2103

3/ Perform the same search for your optimal road trip from Seattle to Dallas when using A*. List the contents of the frontier and explored list for each node that you visit during search, in the format:
[Current node: X; Evaluation function (current node)=???; Explored Cities: [ ….]; Frontier: [(City_X, f(City_X)), ….]; ] (**10 points)**

**Code output:**
```
==== A* search
[{Seattle, f=1670.0}, Explored=[Seattle],
Frontier=[{Santa Fe, f=2023.0}, {Chicago,
f=2864.0}, {San Francisco, f=2037.0}]]
[{Santa Fe, f=2023.0}, Explored=[Seattle, Santa
Fe], Frontier=[{San Francisco, f=2037.0}, {Dallas,
f=2103.0}, {Bakersville, f=3609.0}, {Chicago,
f=3535.0}, {Chicago, f=2864.0}]]
[{San Francisco, f=2037.0}, Explored=[Seattle,
Santa Fe, San Francisco], Frontier=[{Dallas,
f=2103.0}, {Bakersville, f=2372.0}, {Los Angeles,
f=2290.0}, {Chicago, f=3535.0}, {Chicago,
f=2864.0}, {Boston, f=5453.0}, {Bakersville,
f=3609.0}]]
Solution = Seattle - Action[name==moveTo,
location==Santa Fe] -> Santa Fe -
Action[name==moveTo, location==Dallas] -> Dallas
Path cost = 2103.0
```

4/ Find the same solution manually, specifying:

(a) the current node; If it is not a goal node then also:

(b) the children of the current node and the value of their evaluated function (detailing how you have computed it)
(c) The current path from Seattle to the current city + the cost
(d) The Contents of the Explored Cities list
(e) The Contents of the Frontier
(f) Next node.

**(5 points)**

**Step 1**
Current Node = Seattle
Goal Test = False
Children = [{Chicago, f=2864}, {Santa Fe, f=2023}, {San Francisco, f=2037}]
f(Chicago) = g(Chicago) + h(Chicago) = 2064 + 800 = 2864
f(Santa Fe) = g(Santa Fe) + h(Santa Fe) = 1463 + 560 = 2023
f(San Francisco) = g(San Francisco) + h(San Francisco) = 807 + 1230 = 2037
Explored = { Seattle }
Frontier=[{Santa Fe, f=2023.0}, {Chicago, f=2864.0}, {San Francisco, f=2037.0}]
Next Node = Santa Fe

**Step 2**
Current Node = Santa Fe
Goal Test = False
Children = [{Bakersville, f=3609}, {Chicago, f=3535}, {Dallas, f=2103}, {Seattle, f=4596}]
Path = Seattle -> Santa Fe; Cost = 1463
f(Seattle) = (1463 + 1463) + 1670 = 4596
f(Chicago) = (1463 + 1272)  + 800 = 3535
f(Bakersville) = (1463 + 864) + 1282 = 3609
f(Dallas) = (1463 + 640) + 0 = 2103
Frontier=[{San Francisco, f=2037.0}, {Dallas, f=2103.0}, {Bakersville, f=3609.0}, {Chicago, f=3535.0}, {Chicago, f=2864.0}]
Explored = {Seattle, Santa Fe}
Next Node = San Francisco

**Step 3**
Current Node = San Francisco
Goal Test = False
Children = [{Bakersville, f=2372.0}, {Boston, f=5453.0}, {Los Angeles, f=2290.0}, {Seattle, f=3284.0}]
Path = Seattle -> San Francisco; Cost = 807
f(Bakersville) = (807 + 283) /- 1282 = 2372
f(Boston) = (807 + 3095) + 1551 = 5453

f(Los Angeles) = (807 + 383) + 1100 = 2290
f(Seattle) = (807 + 807) + 1670 = 3284
Frontier=[{Dallas, f=2103.0}, {Bakersville, f=2372.0}, {Los Angeles, f=2290.0}, {Chicago, f=3535.0}, {Chicago, f=2864.0}, {Boston, f=5453.0}, {Bakersville, f=3609.0}]
Explored = {Seattle, Santa Fe, San Francisco}
Next Node = Dallas

**Step 4**
Current Node = Dallas
Goal Test = True
Path = Seattle -> Santa Fe -> Dallas; Cost = 2103

*Extra-credit* Write a program that will check if the heuristic provided in this problem is consistent, given the road graph **TOTAL: 20 points**

**Code output**

```
Checking for Consistent Heuristic
Checking if h(Seattle) <= h(Chicago) + c(Seattle,Chicago):  true
Checking  if  h(Seattle)  <=  h(San  Francisco)  +  c(Seattle,San Francisco):  true
Checking if h(Seattle) <= h(Santa Fe) + c(Seattle,Santa Fe):  true
Checking if h(Chicago) <= h(Boston) + c(Chicago,Boston):  true
Checking  if  h(San  Francisco)  <=  h(Bakersville)  +  c(San Francisco,Bakersville):  true
Checking  if  h(San  Francisco)  <=  h(Los  Angeles)  +  c(San Francisco,Los Angeles):  true
Checking if h(Santa Fe) <= h(Dallas) + c(Santa Fe,Dallas):  true
Checking if h(Boston) <= h(Austin) + c(Boston,Austin):  true
Checking if h(Boston) <= h(New York) + c(Boston,New York):  true
Checking if h(Austin) <= h(Charlotte) + c(Austin,Charlotte):  true

Heuristic is Consistent
```

---

**Software Engineering (includes documentation for your programming assignments)**

**Your README file must include the following:**

- Your name and email address.
- *Homework number* for this class (AI CS6364), and the *number of the problem* it solves.
- A description of every file for your solution, the programming language used, supporting files from the aima code used, etc.

- How your code operates, in detail.
- A description of special features (or limitations) of your code.

## Within Code Documentation:

- Methods/functions/procedures should be documented in a meaningful way. This can mean expressive function/variable names as well as explicit documentation.
- Informative method/procedure/function/variable names.
- Efficient implementation
- Don't hardcode variable values, etc