ASSIGNMENT 1

1. Intelligent Office Building

a. Define a PEAS specification for the agent.

1. Performance Measure (P):

- Minimizing the time to turn on the lights as soon as people enter a room
- Time taken to turn the lights off as people leave the room (this time should be within 2 minutes)
- Minimizing the time to get the room within appropriate temperature range of four-degree as soon as a person enters in the room (agent should not take more than 20 minutes for this)
- Maintaining the appropriate temperature range, once reached, while the people are in the room

  All the rewards that the agent is able to achieve after maximizing the performance measure would help in improving the agent's performance.

2. Environment (E):

- People present in the company building and their locations
- Different sizes of rooms (small, medium, huge, etc.)

3. Actuators (A):

- Control over the temperature system of the building (increase/decrease the temperature)
- Control over the lights in the room (turn on/turn off the lights)

4. Sensors (S):

- Location sensors (which tells the locations of people in the building)
- Temperature sensors which tell the agent whether the current room temperature is too cold or too hot (i.e. whether it lies in the appropriate temperature four-degree range)

b. Is it sufficient for the agent to be simple reflex? Why or why not?

- If this agent is a simple reflex agent, then it takes the actions based on the condition matching. Consider a case of a person in the room who leaves the room very frequently and comes back within a few minutes. The agent will stop maintaining the room temperature to a particular level as there are no people in the room. Then as soon as the person comes in, the room temperature must be in the appropriate range. This will increase the cost as the agent just stopped maintaining the temperature, and is needed to start maintaining again.
- Being the simple reflex agent, it won't store any previous perceptions from the sensors. If it would've not been the simple reflex, then it could've stored the locations of that person and could've easily tracked the pattern of his recent locations. It would be easier to find out that the person is consistently moving in and out, so the agent should be watching the room temperature.
- Thus, it won't be a good idea to have this agent of simple reflex kind. It's performance measure will be good if there isn't any unusual action like the one mentioned above.

c. Would it be beneficial for the agent's performance if it randomly heated or cooled rooms where there are no people currently? Identify possible disadvantages to this sort of random action.

- Consider a scenario where the agent randomly heats the rooms with no people inside. Just a minute after that, a person comes to a room. If the room is randomly heated to a higher temperature, it would not be possible for the agent to get it down within the appropriate temperature range within 20 minutes. It will affect the person in the room and he will leave the room, which is not considered to be a good performance for that agent. The same goes with random cooling of the rooms with no people.
- Another similar scenario where a person wants to leave the room for a short time frequently. If person leaves and there is no one else in the room, agent will randomly heat/cool the room. Once the person comes back, he will experience the change in temperature. This will go on and on, as the person has to leave the room frequently for very short period of time. This affects the agent's performance.
- Other disadvantage maybe in the case of rooms where people go occasionally like the huge conference room, auditorium, etc. The agent will just randomly heat or cool the room unless there are people in the room. This increases the cost for the performance. Though the agent performs well, it is not worth to continue the work needlessly.

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

- One improvement that could be done is to consider the office timings. In the sense, generally office hours are from morning to evening. Seldom people work late night. So, if the agent could check whether all people have left the office building (or all the rooms are empty), then it would do nothing. Instead of checking continuously whether a person has come into any of the rooms or not, it can check after specific time intervals. If the room is not empty, it should start working again. It will save the cost of working for the time when nobody is in the building.
- Disadvantage for this is, if there is an emergency and people want to be in the room for important work. As being no people in the room previously, the agent has stopped working. It will take enough time to achieve the appropriate temperature if needed. It won't even turn on the lights for some time. This would affect his performance.

2. Watson

a. Describe a PEAS (R&N Ch. 2) specification for Watson.

- **Performance:** The performance measure for Watson includes minimizing the amount of time that Watson takes for answer, high precision and confidence in the answer which Watson comes up with. Here, not only the accuracy is important, but also the efficiency i.e. giving the accurate answer within a second or two. It also needs to check whether it has enough precision for the answer. If not, then it should not answer the question.
- **Environment:** Watson's environment is made up with the opponents who compete with it in the game of Jeopardy, host of the game, questions being asked to the competitors, categories of the question, and the current amount of money the it has with him. Amount which Watson has must be sufficient enough to proceed for the next bet. If it doesn't have that information about the money it has, it can't decide whether it is eligible for the bet or not. (It might have zero balance)
- **Actuators:** Speakers which tell the answer generated by Watson, and also tell the next category from which to choose the question.
- **Sensors:** It includes sensors which perceives the question and its category (like reader), the balance available with Watson for next bet. One might include this scenario, where a contestant buzzes in and fails to answer the question correctly. Then, it is Watson's time to make sure that the answer which was given by the previous contestant is not the one, which Watson is also going to give.

b. 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.

- For Jeopardy game, it is very much important to come up with the most accurate answer as quickly as possible. Because, if it takes too much time to find the answer, then other contestants will surely buzz in. The problem here contains both the speed and accuracy test. Coming up with any random answer, just for the sake of speed, doesn't help as there are negative points system in the game of Jeopardy. For this problem, Watson is made up of very high end, powerful computer systems connected in parallel, which make it very efficient to find the answer within 1-2 seconds.
- Watson is not connected to the internet while the gameplay. So, it can't just find the answer within 2-3 seconds as it is fed a terabytes of information. Watson learns from the data and whenever it comes across any question which it has never dealt with, it tries to look for the pattern and maximize the confidence of the final answer with respect to remaining ones.
- Question category is one of problems which Watson faces during the game. If the questions are straightforward then Watson quickly comes up with the correct answer. But, the questions type like decomposition, nested clues, puzzles, etc. need more time to solve. So, Watson can't expect any particular type of question to be asked every time. It needs to be trained over unbiased dataset i.e. which contains the questions from all the possible categories.

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

- **DeepQA** architecture is massively parallel architecture which is used in Watson for getting answers with high precision and confidence for the game of Jeopardy. It is basically an amalgamation of different computer science technologies like machine learning, knowledge representation, automated learning, etc. DeepQA approach for Jeopardy uses different algorithms which are independent of each other; but at the end those algorithms sync with each other so as to achieve the highest precision answer.
- It has several roles that help Watson for the Jeopardy challenge. They include:

i. **Content Acquisition:** As Watson is not connected to internet during the game, it needs to have a tremendous amount of general knowledge. It is impossible to store every single information available on the internet into the local database for Watson. Here, it needs to learn from the content it has. So, Wikipedia, encyclopedia, reference books, etc. are fed to it and it extracts the important information from the corpus and uses it during the game.

ii. **Question Analysis:** Natural language is difficult for a computer to understand. If the question asked is straightforward then Watson will have no problem in finding the correct answer for that question. But, the questions which contains the words with multiple meaning, hidden meaning, sarcasm, make Watson's life difficult. The remedy for this is to perform question analysis.

iii. **Hypothesis Generation:** The candidate answers are generated here based on the output of question analysis. Primary search is used here for searching the corpus of data and it extracts the relevant answers as candidates. One by one, answers are put in the question to make it standalone. If in this stage, the system can't find any relevant candidate answer, then it ultimately has to give up. So, it mainly prefers recall over precision here.

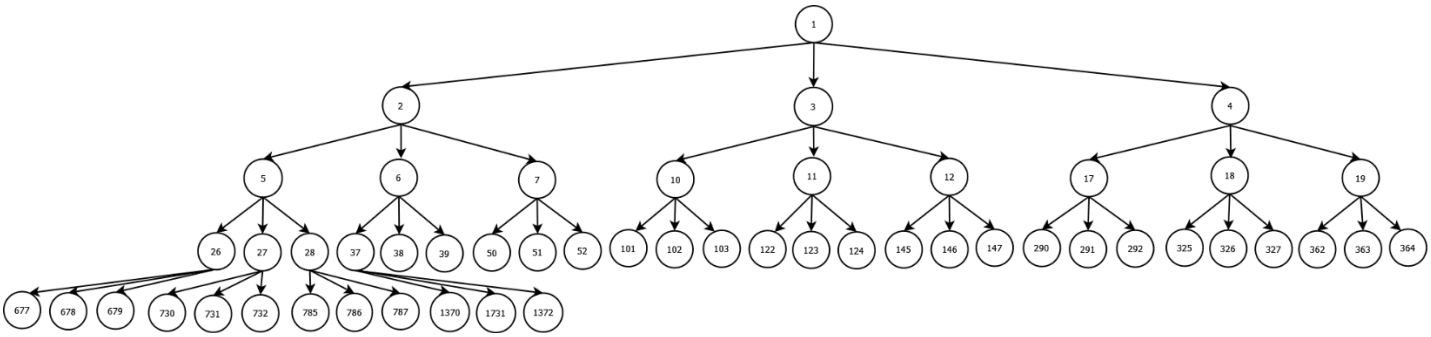
iv. **Sort Filtering:** In this stage, sort filtering score threshold is applied. The candidates, generated in the previous stage, above the threshold value are eligible for the next stage of scoring.

v. **Scoring:** Here, for all those candidate answers, who have been above the threshold filtering value, the supporting evidence is evaluated. Score is given based on how confidently the evidence support the candidate answer.

vi. **Merging:** Possibility of multiple candidate answers representing the same thing but in different form can't be ruled out. Such candidate answers are merged here before scoring them. Finally, different hypotheses are ranked based on the scores they receive.

3. **Consider a state space where the start state is labeled 1 to each state k has three successors: labeled (k^2) + 1, (k^2) + 2, and (k^2) + 3 respectively.**

- **Draw the portion of the state space for states 1 to 1370.**
  The state space will start from k = 1. Thus, its successors will be:
  k^2+1 = 1+1 = 2; k^2+2 = 1+2 = 3; k^2+3 = 1+3 = 4.
  If we continue this process till we come across 1370, we get the following search space:



- **Suppose the goal state is 101. List the order in which states will be visited for the breath-first search.**
  BFS expands the nodes in level by level fashion. Here, it will start with the root node 1. Then, it will expand all the nodes in the next level viz. 2,3,4. It will continue the process level by level until it finds 101.
  1→2→3→4→5→6→7→10→11→12→17→18→19→26→27→28→37→38→39→50→51→52→**101**

- **Suppose the goal state is 101. List the order in which states will be visited for the depth-limited search with limit 3.**
  Depth Limited search follows DFS. But, the depth is limited to 3. So, it will start from the root node 1 and expand the leftmost child of 1 viz. 2. It will recursively follow this procedure until it hits the leaf node, where nothing can be expanded further. It backtracks from there to the previous position and carries out the same procedure again. The condition is that; it can't go beyond level 3. Here I will not find 101 because it is on level 4.
  1→2→5→6→7→3→10→11→12→4→17→18→19

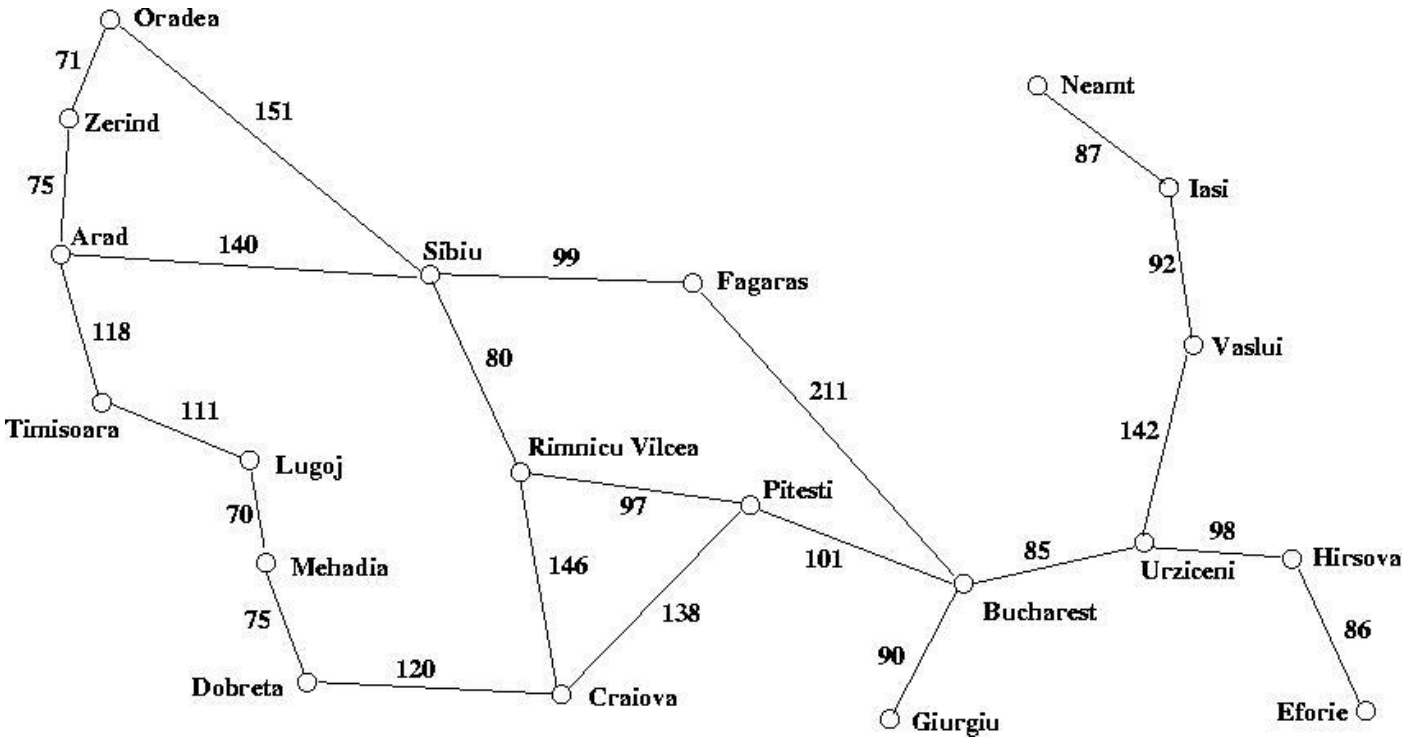
- **Suppose the goal state is 101. List the order in which states will be visited for the iterative deepening search with initial cutoff 1 and cutoff increment 1.**
  DFID also uses DFS but with cutoff. It first considers only for cutoff 1. So, it expands the root node only. Here, the goal state is 101. If it doesn't find 101 for the current cutoff, it will increase the cutoff by cutoff_increment factor. Here, it is 1. So, the next cutoff is 1+1 = 2. It will carry out Depth First Search for cutoff 2. If it finds 101, it stops. This process continues till it comes across 101.

  For cutoff 1: 1
  For cutoff 2: 1→2→3→4
  For cutoff 3: 1→2→5→6→7→3→10→11→12→4→17→18→19
  For cutoff 4: 1→2→5→26→27→28→6→37→38→39→7→50→51→52→3→10→**101**

4. **Fill in the following table with proper description of the agents' environment:**

	Fully vs. Partially Observable	**Deterministic vs. Stochastic**	**Episodic vs. Sequential**	**Static vs. Dynamic**	**Discrete vs. Continuous**	**Multiagent vs. Single Agent**
Vacuum Cleaner Agent	Fully (Global dirt sensor)	Deterministic	Episodic	Static	Discrete	Single Agent
Google Car	Partially	Stochastic	Sequential	Dynamic	Continuous	Multiagent
Search and Rescue Robots	Partially	Stochastic	Sequential	Dynamic	Continuous	Multiagent
Document Categorizer	Fully	Deterministic	Episodic	Static	Discrete	Single Agent
Watson	Fully	Stochastic	Sequential	Dynamic	Discrete	Multiagent

5. Romania Map



a. Consider the path from Lugoj to Bucharest and the path from Bucharest to Lugoj. Run your algorithms and show the paths returned by DFS and BFS results for each case. How do the solution paths compare for the two algorithms? Give an explanation for what you observe.

Source city: Lugoj | Destination city: Bucharest

- **BFS: Lugoj** → Mehadia → Timisoara → Dobreta → Arad → Craiova → Sibiu → Zerind → Pitesti → Rimnicu Vilcea → Fagaras → Oradea → **Bucharest**
  Total number of nodes visited: 13 (including Lugoj and Bucharest)
- **DFS: Lugoj** → Mehadia → Dobreta → Craiova → Pitesti → **Bucharest**
  Total number of nodes visited: 6 (including Lugoj and Bucharest)

Source city: Bucharest | Destination city: Lugoj

- **BFS: Bucharest** → Fagaras → Giurgiu → Pitesti → Urziceni → Sibiu → Craiova → Rimnicu Vilcea → Hirsova → Vaslui → Arad → Oradea → Dobreta → Eforie → Iasi → Timisoara → Zerind → Mehadia → Neamt → **Lugoj**
  Total number of nodes visited: 20 (including Bucharest and Lugoj)
- **DFS: Bucharest** → Fagaras → Sibiu → Arad → Timisoara → **Lugoj**
  Total number of nodes visited: 6 (including Bucharest and Lugoj

Comparison of BFS and DFS based on the output for the code:

For the path from Lugoj to Bucharest, **Breadth First Search (BFS)** expands too many nodes as it goes on expanding all the nodes on the same level from the source (root) node at a time. Bucharest being

far deep from Lugoj, BFS visits each and every node from all the levels above Bucharest and expands its children. The worst case (here, visiting all the nodes in the graph) is found at the time of going from Bucharest to Lugoj, where Lugoj is the last node which is expanded as it is the deepest from the source (root) node.
For the path from Lugoj to Bucharest, **Depth First Search (DFS)** seems to be better performing. For the particular path, it goes on expanding from leftmost child and recursively calling itself. Luckily being the deepest of all, Bucharest is visited prior to approaching all the other nodes in the graph. The same goes with the reverse path i.e. from Bucharest to Lugoj.

b. Is there a case where Depth-First performs worse than Breadth-First (in terms of number of cities visited in the path, not the distance)? If yes, what is the case? If not, explain why.

Yes. Consider a path from Urziceni to Hirsova.
DFS: Urziceni → Bucharest → Fagaras → Sibiu → Arad → Timisoara → Lugoj → Mehadia → Dobreta → Craiova → Zerind → Oradea → Rimnicu Vilcea → Giurgiu → Pitesti → **Hirsova**
BFS: Urziceni → Bucharest → **Hirsova**

Nodes expanded by DFS: 16
Nodes expanded by BFS: 3
Here, DFS goes to the leftmost child of Urziceni and the goes to the opposite direction. But, the destination is on the other side. In general, whenever the destination is the rightmost child of the root (even though it is the shallowest among all the sub children), it will go in the leftmost direction expanding all the nodes over there and will come back to the destination. In this case, BFS is better as it expands the nodes on the same level at the same time.

c. Is there a case where Breadth-First performs worse than Depth-First (in terms of number of cities visited in the path, not the distance)? If yes, what is the case? If not, explain why.

Yes. Consider a path from Timisoara to Giurgiu.
BFS: Timisoara → Arad → Lugoj → Sibiu → Zerind → Mehadia → Fagaras → Oradea → Rimnicu Vilcea → Dobreta → Bucharest → Craiova → Pitesti → **Giurgiu**
DFS: Timisoara → Arad → Sibiu → Fagaras → Bucharest → **Giurgiu**

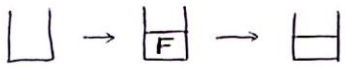
Nodes expanded by BFS: 14
Nodes expanded by DFS: 6
Here, BFS expands level by level, starting from Timisoara. Comparatively, Giurgiu is one of the deepest. That's why it takes too much time to expand all the nodes from levels above it. DFS works fine because it goes deeper and deeper for the same sub child by recursion. In general, whenever the destination is at the deepest level and to the extreme right, BFS will almost expand all the nodes on its way to the destination, resulting in the worst case. This is the time when DFS helps.

d. For the same graph, perform a hand-execution of Depth-First Iterative Deepening (DFID) with increment and cutoff initialized to 1, starting at Fagaras. List the nodes in the order expanded for the first five iterations of DFID, and the state of the data structure (stack) after each iteration. Expand the nodes alphabetically and insert them in non-decreasing alphabetical order. How does this list compare with the list of expansions in Breadth-First Search?

DFID : cutoff increment = 1

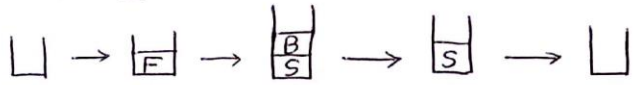
The stack used here is not a function call stack. Thus, we aren't doing recursive calling in DFS. Thus, there is no need to maintain the parent in the stack, once it is expanded.

1) cutoff = 1

Stack : 

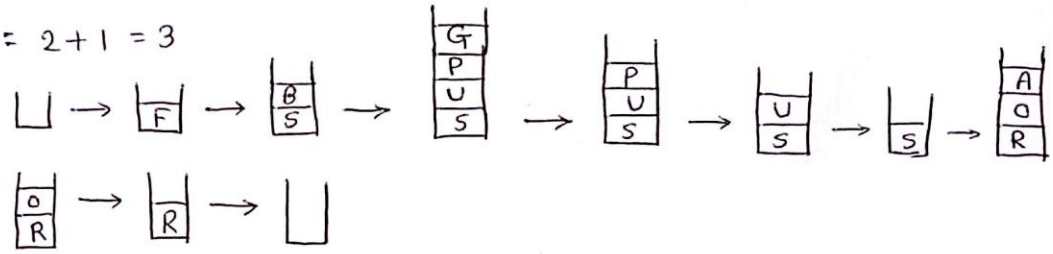
Nodes : Fagaras

2) Cutoff = 1+1 = 2

Stack : 

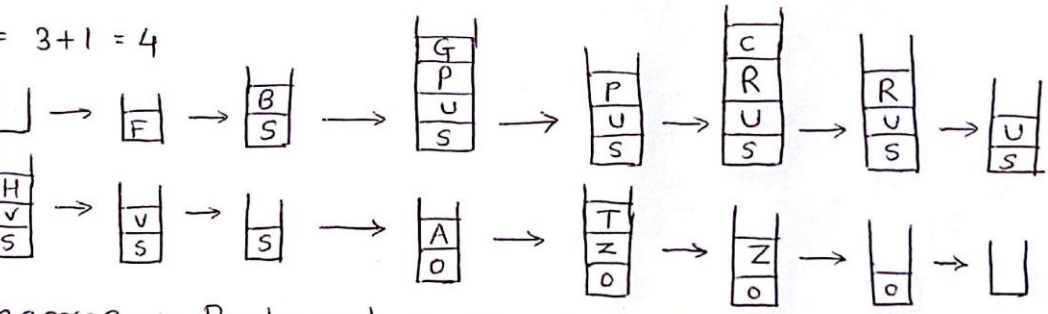
Nodes : Fagaras → Bucharest → Sibiu

3) Cutoff = 2+1 = 3

Stack : 

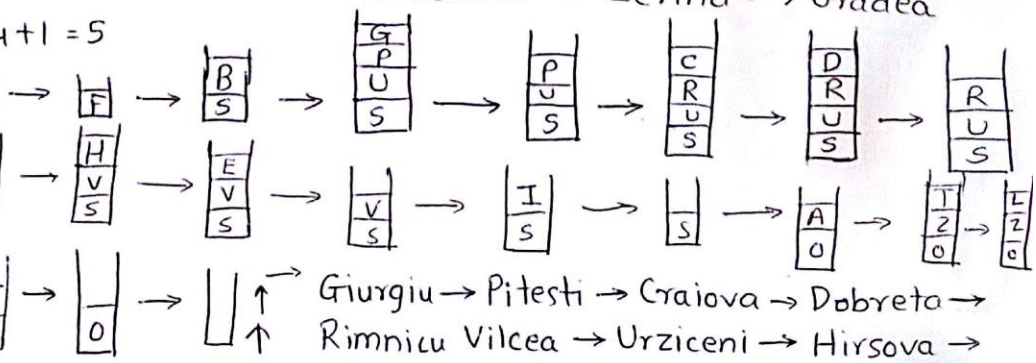
Nodes : Fagaras → Bucharest → Giurgiu → Pitesti → Urziceni → Sibiu → Arad → Oradea → Rimnicu Vilcea

4) Cutoff = 3+1 = 4

Stack : 

Nodes : Fagaras → Bucharest → Giurgiu → Pitesti → Craiova → Rimnicu Vilcea → Urziceni → Hirsova → Vaslui → sibiu → Arad → Timisoara → Zerind → Oradea

5) Cutoff = 4+1 = 5

stack : 

Nodes : Fagaras → Bucharest → Giurgiu → Pitesti → Craiova → Dobreta → Rimnicu Vilcea → Urziceni → Hirsova → Eforie → Vaslui → Iasi → Sibiu → Arad → Timisoara → Lugoj → Zerind → Oradea

For the first two iterations of DFID (cutoff = 1 and cutoff = 2), it expands the nodes as same as that BFS does. Because, DFID has nothing to expand beyond level 2, thus it just works same as BFS. The next level onwards, BFS goes on expanding the nodes on the same level and DFID continues using DFS. Thus, they start deviating after some iterations, and in the end come with different sequence of nodes expanded.

BFS from Fagaras to Neamt:
Fagaras → Bucharest → Sibiu → Giurgiu → Pitesti → Urziceni → Arad → Oradea → Rimnicu Vilcea → Craiova → Hirsova → Vaslui → Timisoara → Zerind → Dobreta → Eforie → Iasi → Lugoj → Mehadia → Neamt