**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Artificial Intelligence (BITS F444/ CS F407)**

**I Semester 2019-20**

**Programming Assignment-4**

**Coding Details**

**(November 1, 2019)**

*Instruction: Type the details precisely and neatly*

1. ID **2019H1030023P**

Name **SUBHASHIS DHAR**

1. Mention the names of Submitted files :
   1. **cspGraph.py**
   2. **Node.py**
   3. **cspgraph.jpg**
   4. <filename.ext>
   5. <filename.ext>
   6. <filename.ext>
   7. <filename.ext>
2. Total number of submitted files: **2**
3. Name of the folder : **2019H1030023P\_CSF407\_PA\_4**
4. Have you checked that all the files you are submitting have your name in the top?(yes/no) **Yes**
5. Have you checked that all the files you are submitting are in the folder as specified in 4 (and no subfolder exists)?(yes/no) **Yes**
6. Problem formulation
   1. List of variables (Specify all variables):

**Group : List of list. It contains all the personalities represented as integers for a particular group**

**Nodes : Structure which represents each personality. It contains an int for identification and corresponding hours when that particular personality is free. This is essentially the domain.**

**CSPGraph : This is the constraint graph represented as an upper triangular matrix.**

* 1. Value domains of variables (Also list the variables against each value domain correspondingly)

Group = [  
 [],  
 [3, 5, 8, 9, 12, 18, 19],  
 [8, 9, 12, 19, 2],  
 [3, 5, 4, 16, 8, 9, 19],  
 [8, 9, 12, 15],  
 [15, 16, 17, 18, 19, 20],  
 [3, 5, 7, 11, 14, 20],  
 [3, 5, 12, 2, 18, 19, 20, 1],  
 [3, 5, 8, 9, 10, 18, 19, 20],  
 [3, 13, 8, 9, 7, 19, 20],  
 [1, 8, 9, 13, 20],  
 [18, 19, 20],  
 [3, 11, 8, 18, 19, 20],  
 [3, 8, 10, 12, 4, 20],  
 [3, 5, 11, 9, 10, 17, 19, 20],  
 [2, 8, 12, 18, 19, 20]  
]

Node(1, list((2, 5, 7))))  
Node(2, list(((1, 4, 6, 2)))))  
Node(3, list(((2, 5, 6, 1)))))  
Node(4, list(((2, 4, 6, 8)))))  
Node(5, list(((2, 6, 5)))))  
Node(6, list(((1, 5, 3)))))  
Node(7, list(((2, 4, 6, 1, 8)))))  
Node(8, list(((1, 3, 4)))))  
Node(9, list(((4, 1, 5, 8, 6)))))  
Node(10, list(((8,)))))  
Node(11, list(((2, 3)))))  
Node(12, list(((1, 2, 3, 4, 7)))))  
Node(13, list(((7, 1, 8)))))  
Node(14, list(((5, 3, 6, 1)))))  
Node(15, list(((2, 5)))))  
Node(16, list(((2, 5, 1, 4)))))  
Node(17, list(((1, 4, 5, 6)))))  
Node(18, list(((5, 4)))))  
Node(19, list(((1, 3, 6, 8)))))  
Node(20, list(((6,)))))

* 1. Mention the constraints
     + 1. **No two group meeting the same personality can be assigned the same hour**
       2. **Meeting with personalities can scheduled only on the hours present within their domain**
       3. **Scheduling should be done so that each group is able to meet all the personalities present in its list.**

1. Data structure used
   1. Constraint graph node structure:

**Id is an integer corresponding to personality**

**domain : is a list of free time for that particular personality**

class Node:  
 def \_\_init\_\_(self,id,domain):  
 self.domain = domain  
 self.id = id

* 1. Constraint graph edge structure:

**Constraint edge are denoted by presence of 1 in the adjacency matrix.**

* 1. Constraint graph (Adjacency list/ adjacency matrix/ any other(specify)

Constraint Graph Represented as Adjacency matrix:

==================================================

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

--- --- --- --- --- --- --- --- --- --- --- --- --- --- ---

1| --- 1 1 1 1 1 1 1 1 1 1 1 1 1 1

2| --- 1 1 1 0 1 1 1 1 1 1 1 1 1

3| --- 1 1 1 1 1 1 1 1 1 1 1 1

4| --- 1 0 1 1 1 1 0 1 1 1 1

5| --- 1 1 1 1 1 1 1 1 1 1

6| --- 1 1 1 1 1 1 1 1 1

7| --- 1 1 1 1 1 1 1 1

8| --- 1 1 1 1 1 1 1

9| --- 1 1 1 1 1 1

10| --- 1 1 1 1 1

11| --- 1 1 1 1

12| --- 1 1 1

13| --- 1 1

14| --- 1

15| ---

* 1. How are you maintaining value domains as you go with search process?

**Domain values are part of the node structure**

1. DFS + backtracking technique details
   1. Variable ordering used (List heuristics used):

**Normal ordering was used without heuristics**

* 1. Node structure for DFS:

**Same as constraint graph**

* 1. Method for assignment of a value to a variable and backtracking:

**The first time for a personality is allocated to a group. Then we check if a time can be assigned to the next group without failing the constraints. This is done recursively until the base case. Whenever an assignment is not possible, we backtrack to previous level in the tree and assign the next open time slot.**

* 1. How is edge node of your adjacency list (constraint graph) useful in deciding upon which constraint module( or modules) to use for testing the violation of the constraints while you assign a value to a variable?

**We check if the assignment that we are making is safe and does not violate constraints using the adjacency list of constraint graph. If there is an edge from one node to another in our case, that means that they cannot be assigned the same time slot. They need to be assigned different time slot to prevent collision in the schedule**

* 1. Total number of nodes generated for assignment of values to all variables: **1203**
  2. Write the statistics here as asked

R1 = **1203** R2 = **13.2** R3 = **6.133**

R4 = **0.1433858871459961**  R5= **N/A**

* 1. Code status (implemented fully/ partially/ not done) **Implemented per group**

1. DFS+ Backtracking using constraint propagation:
   1. Explain the method for constraint propagation. How are you updating the value domains? What do you do with the value domains of the variables when you backtrack while performing DFS?
   2. Total number of nodes generated using the above technique
   3. Write the statistics here as asked

R6 = R7 = R8 =

1. Code status (implemented fully/ partially/ not done) **Not done**

1. Comparative analysis

Fill in the following information

|  |  |  |
| --- | --- | --- |
|  | DFS+BT | DFS+BT+Constraint propagation |
| Average number of nodes created | 1203 |  |
| Average time taken | 0.1433858871459961 |  |

1. Compilation Details:
   1. Code Compiles (Yes/ No): **Yes**
   2. Mention the .py files that do not compile: **N/A**
   3. Any specific function that does not compile: **N/A**
   4. Ensured the compatibility of your code with the specified Python version(yes/no) **Yes**
   5. Instructions for compilation of your files mentioning the multi file compilation process used by you (We may use the replica of these for compiling your files while evaluating your code) **python cspGraph.py**
2. Driver Details: Does it take care of the options specified earlier(yes/no): No
3. Execution status (describe in maximum 2 lines)
4. Declaration: I, SUBHASHIS DHAR declare that I have put my genuine efforts in creating the python code for the given programming assignment and have submitted only the code developed by me. I have not copied any piece of code from any source. If the code is found plagiarized in any form or degree, I understand that a disciplinary action as per the institute rules will be taken against me and I will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani.

ID 2019H1030023P Name: Subhashis Dhar

Date: 1st November 2019

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*