CS711: Introduction to Game Theory and Mechanism Design -- Assignment 1

General Instructions

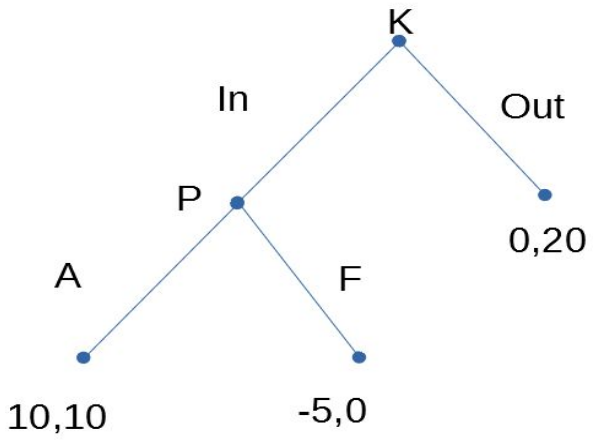
- You are required to use the input format of GAMBIT (http://www.gambit-project.org/gambit15/formats.html) for your assignment.
  Example:
  NFG 1 R "Selten (IJGT, 75), Figure 2, normal form"
  { "Player 1" "Player 2" } { 3 2 }
  0 1 1 2 0 3 2 2 0 1 0 0
  The above file represents a 2 player game, where Player 1 has 3 strategies and Player 2 has 2 strategies:

Player 1 \ Player 2	**1**	**2**
1	0, 1	2, 2
2	1, 2	0, 1
3	0, 3	0, 0

  There is also EFG format which will be used in some question, following is a example for that
  Example:

```
EFG 2 R "GAME NAME" { "K" "P" }
""
p "K" 1 1 "" { "In" "Out" } 0
p "P" 2 1 "" { "A" "F" } 0
t "" 1 "" { 10, 10 }
t "" 2 "" { -5, 0 }
t "" 3 "" { 0, 20 }
```

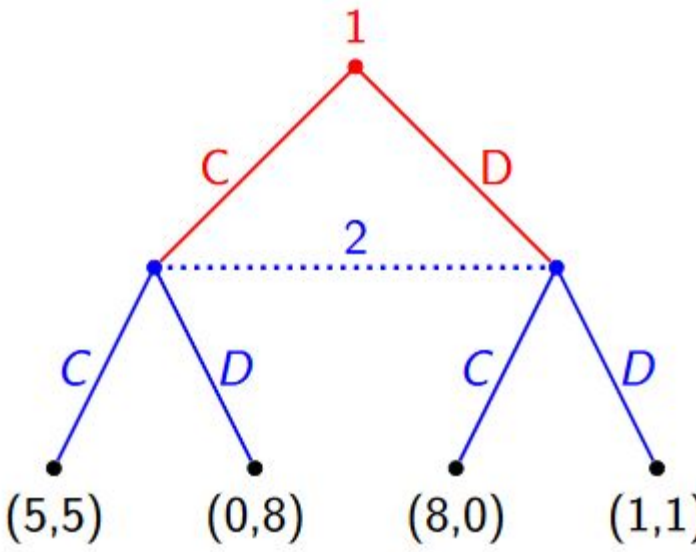
  This represents the following game:



For the imperfect EFG following is the representation:
EFG 2 R "Prisoner's Dilemma" { "Player 1" "Player 2" }
""

p "1" 1 1 "" { "C" "D" } 0
p "2" 2 1 "" { "C" "D" } 0
t "" 1 "" { 5, 5 }
t "" 2 "" { 0, 8 }
p "2" 2 1 "" { "C" "D" } 0
t "" 3 "" { 8, 0 }
t "" 4 "" { 1, 1 }

And this represents the following game:



- The code should be scalable to any number of players and strategies, even though the examples shown are rather small in every question.
- The code must be written in Python 3 (or any similar language that supports text file reading). No game theory related packages should be used in your code (irrespective of which language you use), and the package dependency should be as minimal as possible. E.g., only numpy may be used for array-related operations.
- For every question, you are required to implement a function (details are provided with each question) which returns a vector or vector of vectors (lists in case of Python). The element of each vector represents the strategy of each player.
  Example: For the game defined above, Pure Strategy Nash Equilibrium will be [1, 0, 0, 0, 1] and [0, 1, 0, 1, 0], where each list corresponds to [Player 1: 1, Player 1: 2, Player 1: 3, Player 2: 1, Player 2: 2].
  Therefore, the function should return [[1, 0, 0, 0, 1], [0, 1, 0, 1, 0]]. The inner list is one PSNE, where the equilibrium strategies for player 1 are listed first followed by that of player 2. E.g., in [1, 0, 0, 0, 1] the first three entries correspond to the three strategies of player 1, and the first is picked, the last two entries correspond to the two strategies of player 2, and the second is picked.
  Note: Please refer to the respective question for a precise output format.
- The final submission checklist is as follows:
  a. One .zip archive for each group (of at most 5 members), that contains
  b. One .py file containing the functions for each question in the assignment,
  c. No file should use any game theory related packages.
- **All utility values are integral.**

Questions

1. SDSE

Given a normal form game, you have to write a program to find the strongly dominant strategy of each player. Define a function named **computeSDS(file_name)** which takes input and produces output as defined below.

Input Format:
For input, you will be given a filename as **file_name** corresponding to a .nfg file which will contain the representation of the game.
Output Format:
You have to return a single list of strategies that will show the strongly dominant strategy of each player. In case no such strategy exists for a given game, each element of the list would be zero [0, 0,..., 0].

Example:
Input:
Game.nfg
The contents of this file will look like:-
NFG 1 R "Game Title"
{ "Player 1" "Player 2" } { 3 3 }
2 12 0 12 0 12 1 10 0 10 0 10 1 11 0 11 0 9
Output :
[1, 0, 0, 1, 0, 0]

Player 1\Player 2	**1**	**2**	**3**
1	2, 12	1, 10	1, 11
2	0, 12	0, 10	0, 11
3	0, 12	0, 10	0, 9

Explanation:
For the given game we can see that for player 1 the strictly dominant strategy would be 1 and similarly for player 2 it would be strategy 1 as well, hence the output will be: [1, 0, 0, 1, 0, 0] which corresponds to [Player 1:1, Player 1: 2, Player 1: 3, Player 2: 1, Player 2: 2, Player 2: 3].

2. WDSE

Given a normal form game, write a python program for finding all the weakly dominant strategies of each player. Define a function named **computeWDS(file_name)** which takes input and output as defined below.

Input Format:
You will be given a .nfg file (Gambit format) which will contain the game title, player names, and their respective utilities.

Output Format:
Return the list which will consist of 0 and 1(0 represent not a weakly dominant strategy and 1 that it is) corresponding to the strategy of a player. For more exposure look into the example below

Example:

P 1\P 2	1	2	3
1	2 , 2	3 , 2	3 , 2
2	2 , 3	4 , 4	6 , 3
3	2 , 3	3 , 6	5 , 5

In the above game, strategy 2 is a weakly dominant strategy for both player one and two so the output will be:
[0,1,0,0,1,0] which represents [P 1: S 1,P1: S2,P1 :S3,P2:S1,P2:S2,P2:S3]
P is for player and S for strategy.

3. PSNE

Given a n player normal form game find all the possible pure nash equilibrium. Define a function named **computePSNE(file_name)** which will take input and return output as defined below.

Input:
You will be given the name of the game file, **file_name**, as input corresponding to a .nfg file which will contain the representation of the game.

Output:
Output all the possible pure nash equilibrium for the given game. The output will be in the format of a vector of vectors (list in case of python) where each vector/list will represent a pure nash equilibrium. The vectors can be returned in any order, they need not be sorted in lexicographic order.
For a 2 player game, each nash equilibrium corresponds to [Player 1: 1, Player 1: 2, ... , Player 1: n1, Player 2: 1, ..., Player 2: n2] where player 1 has n1 strategies and player 2 has n2 strategies.
In case there is no PSNE for a given game, return an empty list [].

Example:
Input:
NFG 1 R "Battle of Sexes"
{ "Player 1" "Player 2" } { 2 2 }
3 2 0 0 0 0 2 3

Output :
[[1, 0, 1, 0], [0, 1, 0, 1]]
Explanation:
Given a 2 player finite game:

Player 1\Player 2	1	2
1	3, 2	0, 0
2	0, 0	2, 3

As it is clear from the above matrix, there are two nash equilibrium for the given game [1, 0, 1, 0] and [0, 1, 0, 1] where each nash equilibrium corresponds to [Player 1:1, Player 1: 2, Player 2: 1, Player 2: 2].
Therefore, the function will return [[1, 0, 1, 0], [0, 1, 0, 1]].
Note: The function will return a vector of vectors as there can be more than 1 pure nash equilibrium.

4. EFG TO NFG

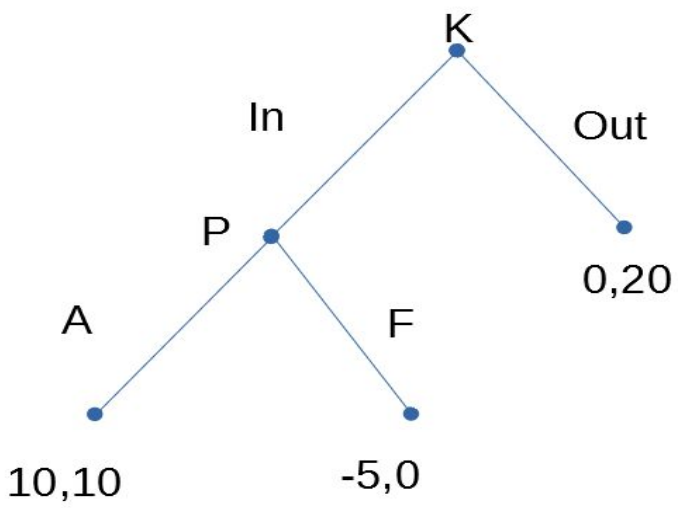
Given a finite extensive form game (can be either perfect or imperfect information -- see the representation in gambit in the general instructions), you have to write a program to convert it to a Normal form game. Define a function named **efg_NFG(file_name)** which takes input and provides output as defined below.

Input Format:
You will be given a .efg file(Gambit format) which will contain the game title, player names and their respective utilities. Format of file will be as follows,which is representing the game given in example

```
EFG 2 R "GAME NAME" { "K" "P" }
""
p "K" 1 1 "" { "In" "Out" } 0
p "P" 2 1 "" { "A" "F" } 0
t "" 1 "" { 10, 10 }
t "" 2 "" { -5, 0 }
t "" 3 "" { 0, 20 }
```

Output Format:
The output will be a .nfg file, but your function should return a list of strings where each string will be a line of .nfg file.
Example:
Following is a game with a left image in EFG format and you have to convert it into an NFG format like the right image.

N={K,P}
S_K={Out,In}, S_P={F,A}



K\P	A	F
In	10,10	-5,0
Out	0,20	0,20

Output:
Following is the output file
NFG 1 R "GAME NAME"
{ "K" "P" }{ { "In" "Out" }{ "A" "F" }}
10 10 0 20 -5 0 0 20
Your function will return list of string as follows
['NFG 1 R "GAME NAME"', '{ "K" "P" }{ { "In" "Out" }{ "A" "F" }}' , '10 10 0 20 -5 0 0 20']
Note:: Game will be for N player and N strategy.

5. SPNE using Backward Induction

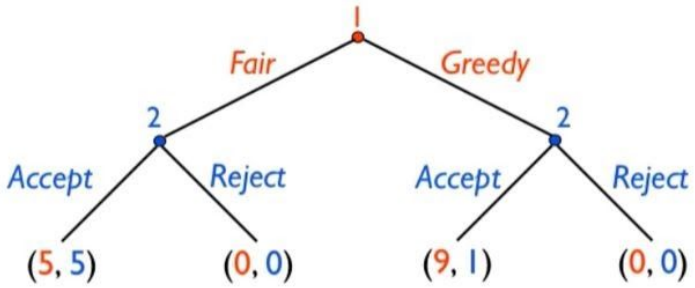
Given a n player finite perfect information extensive form game, find the subgame perfect Nash equilibrium using backward induction. Define a function named **computeSPNE(file_name)** which will take input and return output as defined below.

Input Format:
You will be given a .efg file (Gambit format) which will contain the game title and player names in the header while the body of the file lists the nodes which comprise the game tree.
Output Format:

You have to return a single list of strategies that will show the SPNE for each player.

Example:
Input:
EFG 2 R "Untitled Extensive Game" { "Player 1" "Player 2" }
""
p "" 1 1 "" { "Greedy" "Fair" } 0
p "" 2 1 "" { "Reject" "Accept" } 0
t "" 1 "" { 5, 5 }
t "" 2 "" { 0, 0 }
p "" 2 2 "" { "Reject" "Accept" } 0
t "" 3 "" { 9, 1 }
t "" 4 "" { 0, 0 }
Output :
[[[Greedy], [Accept, Accept]]]

Explanation:



For the given example if we do backward induction the SPNE we get will be (9, 1) which can be represented by the list [[[Greedy], [Accept, Accept]]] where [Accept, Accept] refers to the player 2's strategies and [Greedy] is player 1's strategy.