COMP4600/8460 - Advanced Algorithms **S2 2022**

Assignment 5

[Xueer Zhangsong] ([u7079320]) Date: November 13, 2022

Note: *Plagiarism is strictly prohibited. Even though you may discuss with classmates, you should write your homework assignment by yourself.*

1 Questions

Q1) (10 Marks) Provide a Σ-protocol of zero-knowledge proof to the range of a commitment

- Verifier publicly knows commitment C and k, whereas Prover privately knows (m, r)
- Prover wants to convince Verifier that $g^{\mathsf{m}} \cdot h^{\mathsf{r}} = C$ and $\mathsf{m} = \sum_{i=0}^{k} a_i 2^i$, where $a_i \in \{0, 1\}$, without revealing (m, r) or $(a_i)_{i=0}^{k}$
- Prove completeness, soundness and zero-knowledge of your protocol

Q2) (10 Marks) Implement a program to compute a Nash equilibrium of a congestion game with four nodes and no more than 3 players. Your program should read the linear congestion functions and players' travel requirements from an input text file

1. The input file format for functions and players' travel requirements is given as follows:
```
(#nodeID1, #nodeID2):  a(#nodeID1, #nodeID2), b(#nodeID1, #nodeID2);
(#nodeID3, #nodeID4):  a(#nodeID3, #nodeID4), b(#nodeID3, #nodeID4);
...
#playerID1:  #nodeID1, #nodeID2;
...
```
   For example, the input file for the Braess paradox network is:
```
(A, B): 0, 3;
(A, C): 1, 0;
(B, C): 0, 0.5;
(B, D): 1, 0;
(C, D): 0, 3;
player1:  A, D;
player2:  A, D;
```

2. Output the path and cost of each player at the Nash equilibrium on screen as follows:
```
#playerID1, Cost = [playerID1Cost]:  #nodeID1, #nodeID2, ...;
...
```
   For example, the output for the Braess paradox network is:
```
player1, Cost = 4.5:  A, C, B, D;
player2, Cost = 4.5:  A, C, B, D;
```

Provide a proper readme and examples to show how the input file can be read into your program

2 Answers

2.1 Q1

Firstly, convince Verifier that $g^{\mathsf{m}} \cdot h^{\mathsf{r}} = C$ and $\mathsf{m} = \sum_{i=0}^{k} a_i 2^i$

Prover knows $(a_i)_{i=0}^{k}$ privately, where $\mathsf{m} = \sum_{i=0}^{k} a_i 2^i$.

The Σ-protocol is as follow:

1. Prover randomly generates $\{a'_0, \ldots, a'_k\}$. Use these k numbers to generates m', where $m' = \sum_{i=0}^{k} a'_i 2^i$.
2. Prover randomly generates r'.
3. Prover send $C' = h^{r'} \prod_{i=0}^{k} g^{a'_i 2^i}$ to the Verifier.
4. Verifier send chanllenge β to Prover.
5. Prover generates $z_i = (a'_i + a_i\beta)2^i$ and $z_r = r' + r\beta$. Prover respond with $(\{z_0, \ldots, z_k\}, z_r)$.
6. Verifier check whether: $C'C^{\beta} = h^{z_r} \prod_{i=0}^{k} g^{z_i}$

(1) Completeness

$$\begin{aligned} C'C^{\beta} &= C^{\beta} \cdot h^{r'} \prod_{i=0}^{k} g^{a'_i 2^i} \\ &= h^{r\beta} \prod_{i=0}^{k} g^{a_i 2^i \beta} \cdot h^{r'} \prod_{i=0}^{k} g^{a'_i 2^i} \\ &= h^{r'+r\beta} \prod_{i=0}^{k} g^{a'_i 2^i + a_i 2^i \beta} \\ &= h^{r'+r\beta} \prod_{i=0}^{k} g^{(a'_i + a_i\beta)2^i} \\ &= h^{z_r} \prod_{i=0}^{k} g^{z_i} \end{aligned}$$

(2) Soundness

Assume there is an Extractor, which plays the role of Verifier. Extractor send the challenge β, collect the respond $(\{z_0, \ldots, z_k\}, z_r)$ from Prover. Then Extractor rewind to the former step and sends another challenge β', again collect the respond $(\{z'_0, \ldots, z'_k\}, z'_r)$ from Prover.

Pick corresponding pairs of z_i and z'_i, where $i \in \{0, \ldots, k\}$

$$\begin{aligned} a_i 2^i &= \frac{z_i - z'_i}{\beta - \beta'} = \frac{(a'_i + a_i\beta)2^i - (a'_i + a_i\beta')2^i}{\beta - \beta'} = \frac{(\beta - \beta')a_i 2^i}{\beta - \beta'} \\ m &= \sum_{i=0}^{k} a_i 2^i \\ r &= \frac{z_r - z'_r}{\beta - \beta'} = \frac{r' + r\beta - r' + r\beta'}{\beta - \beta'} = \frac{(\beta - \beta')r}{\beta - \beta'} \end{aligned}$$

However, an Extractor do not exist. These it's Soundness.

(3) Zero-knowledge

Assume there is a Simulator, which plays the role of Prover. The Simulator gets the challenge β, then rewind back to it's former step, and generate C' with $C^{-\beta}$ and random responds $(\{z_0, \ldots, z_k\}, z_r)$.

$$C' = C^{-\beta} \cdot h^{z_r} \prod_{i=0}^{k} g^{z_i}$$

Then the Simulator send C' to the Verifier again, get the same challenge β, and send the same responds $(\{z_0, \ldots, z_k\}, z_r)$ to the Verifier. The Verifier checks $C'C^{\beta} = h^{z_r} \prod_{i=0}^{k} g^{z_i}$.

$$LHS = C^{-\beta} \cdot C^{-\beta} \cdot h^{z_r} \prod_{i=0}^{k} g^{z_i} = h^{z_r} \prod_{i=0}^{k} g^{z_i} = RHS$$

It pass the protocol!

However, a Simulator do not exist. These it's Zero-knowledge.

Secondly, prove $(a_i)_{i=0}^k$ and $a_i \in \{0, 1\}$. Using the membership protocol.

Before start, notice that the private number m, is generated by $\mathsf{m} = \sum_{i=0}^{k} a_i 2^i$. It can be represent as a $k+1$ bit binary number, as:

$$m = (a_0 a_2 \ldots a_k)_2, a_i \in \{0, 1\} \text{ where, } 0 \leq m < 2^{k+1}$$

Thus we can consider m as a member of $X = \{m_0, \ldots, m_n\}$, where m_i orders in sequence $(m_{i-1} < m_i)$ and $n = 2^{k+1}$.

According to above, we can find that m_i's value is actually equal to it's index. That is:

$$m_i = i$$

Now, starts the membership protocol.

The previous steps are consistent with the content on the lecture 10 slides until the last step:

When it's turn for the Verifier to check, it checks:

1. $\beta = \sum_{i=0}^{n}$
2. $n = 2^{k+1}$
3. $g^{z_{m_j}} \cdot h^{z_{r_j}} = g^{m_j'} \cdot h^{r_j'} \cdot (\frac{C}{g^j})^{\beta_j}$, for all $j \in \{1, \ldots, 2^{k+1}\}$.

(1) Completeness According the conclusion I made before the protocol starts, the Completeness can be proved using the same way on the lecture slides.

These also proved that $a_i \in \{0, 1\}$.

(2) Soundness

Assume there is an Extractor, which plays the role of Verifier. Same as before send two challenge β and β' to the Prover. And compare the Prover's responds $(\beta_j)_{j=0}^n$ and $(\beta_j')_{j=0}^n$.

Notice that except β_i, the paired beta for the private m, all the other betas are all the same. This is because they are all set by the Prover and stay unchanged.

Only need to find the different beta we are able to get m's index, which is also equal to the value of m. Change m into a binary number, we can get the values of $(a_i)_{i=0}^k$.

However, an Extractor do not exist. These it's Soundness.

(3) Zero-knowledge

Assume there is a Simulator, which plays the role of Prover. The Simulator gets the challenge β, then rewind back to it's former step, and generate C' with $C^{-\beta}$, g^{m_j} and randoms z_{m_j}s and z_{r_j}s

$$C' = C^{-\beta} \cdot g^{m_j} \cdot g^{z_{m_j}} h^{z_{r_j}}$$

Then the Simulator send $(C', z_{m_j})_{j=0}^n$ to the Verifier again, get the same challenge β, and send the same responds.

$$RHS = C^{-\beta} \cdot g^{m_j} \cdot g^{z_{m_j}} h^{z_{r_j}} \cdot (\frac{C}{g^j})^{\beta_j} = g^{z_{m_j}} h^{z_{r_j}} = LHS$$

Notice that $m_j = j$

It pass the protocol!

However, a Simulator do not exist. These it's Zero-knowledge.

2.2 Q2

(1) Readme

The program needs two inputs to run. The first is to choose the edge mode, enter ”one” or ”two”, represent the one-way edge and two-way edge respectively. And the second is input a file name, which contains a valid format of linear functions and travel requirements.

In the ”one” mode, the edge have directions. And the direction is shown as in the input file.

For example, if the input file contains:

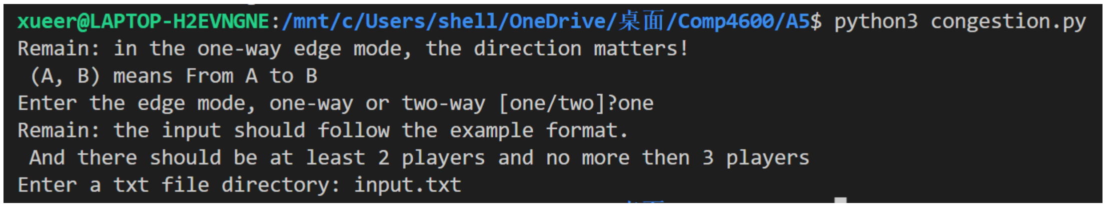
```
(A, B): 0, 3;
(A, C): 1, 0;
(B, C): 0, 0.5;
(B, D): 1, 0;
(C, D): 0, 3;
player1:  A, D;
player2:  A, D;
```

Then in the one-way mode, ”(A, B): 0, 3;” only represent the cost of the edge ”From A to B”, ”(B, C): 0, 0.5;” only represent the cost of the edge ”From B to C”. The topology doesn't contains cost for edge ”From B to A” nor edge ”From C to B”. Thus, it's invalid to go from B to A, and these path will not consider by the player.

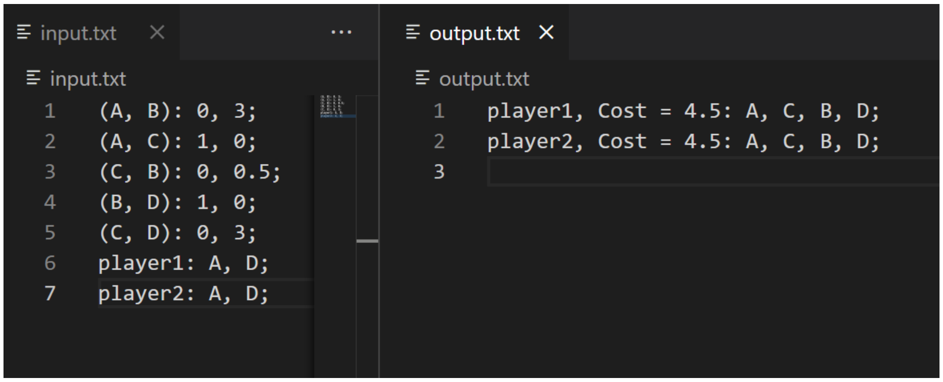
However, in the two-way mode, ”(A, B): 0, 3;” represent the cost of both edges ”From A to B” and ”From B to A”. The same for all the other edges. Compare to the one-way mode, player have more strategies to meets the requirement.

The contain of the input file should strictly follow the question examples.

Here is an implement example:



```
xueer@LAPTOP-H2EVNGNE:/mnt/c/Users/shell/OneDrive/桌面/Comp4600/A5$ python3 congestion.py
Remain: in the one-way edge mode, the direction matters!
 (A, B) means From A to B
Enter the edge mode, one-way or two-way [one/two]?one
Remain: the input should follow the example format.
 And there should be at least 2 players and no more then 3 players
Enter a txt file directory: input.txt
```



input.txt

```
(A, B): 0, 3;
(A, C): 1, 0;
(C, B): 0, 0.5;
(B, D): 1, 0;
(C, D): 0, 3;
player1: A, D;
player2: A, D;
```

output.txt

```
player1, Cost = 4.5: A, C, B, D;
player2, Cost = 4.5: A, C, B, D;
```

(2) Example

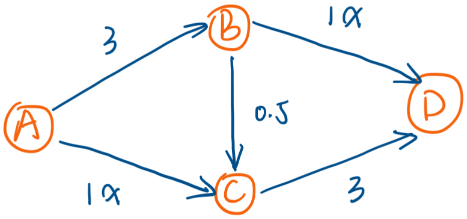
1. Using ”one” mode and 2 players. The contain of the input file is:

```
(A, B): 0, 3;
(A, C): 1, 0;
(B, C): 0, 0.5;
(B, D): 1, 0;
(C, D): 0, 3;
player1:  A, D;
player2:  A, D;
```

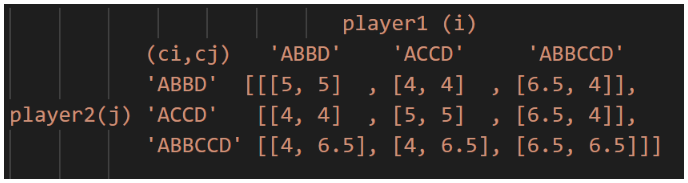
   The output is:

```
player1, Cost = 4:  A, C, D;
player2, Cost = 4:  A, B, D;
player1, Cost = 4:  A, B, D;
player2, Cost = 4:  A, C, D;
```

   This indicate that there are 2 Nash equilibrium.
   The topology is as follow:



   And the cost table is as follow:



```
                      player1 (i)
           (ci,cj)   'ABBD'    'ACCD'     'ABBCCD'
           'ABBD'  [[[5, 5]   , [4, 4]   , [6.5, 4]],
player2(j) 'ACCD'    [[4, 4]   , [5, 5]   , [6.5, 4]],
           'ABBCCD' [[4, 6.5], [4, 6.5], [6.5, 6.5]]]
```

2. Using ”one” mode and 2 players.The contain of the input file is:

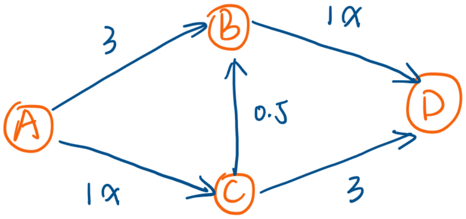
```
(A, B): 0, 3;
(A, C): 1, 0;
(C, B): 0, 0.5;
(B, D): 1, 0;
(C, D): 0, 3;
player1:  A, D;
player2:  A, D;
```

The output is:

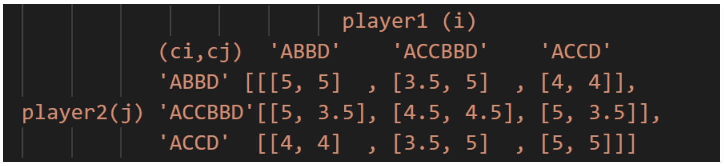
```
player1, Cost = 4.5:  A, C, B, D;
player2, Cost = 4.5:  A, C, B, D;
```

This indicate that there are 1 Nash equilibrium.

The topology is as follow:



And the cost table is as follow:



```
                    player1 (i)
           (ci,cj)  'ABBD'     'ACCBBD'     'ACCD'
           'ABBD'  [[[5, 5]   , [3.5, 5]   , [4, 4]],
player2(j) 'ACCBBD'[[5, 3.5], [4.5, 4.5], [5, 3.5]],
           'ACCD'   [[4, 4]   , [3.5, 5]   , [5, 5]]]
```

3. Using ”two” mode and 2 players. The contain of the input file is:

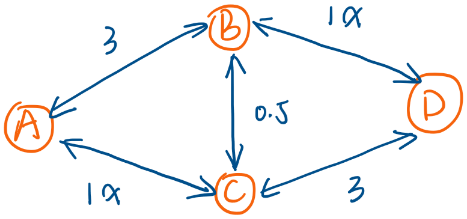
```
(A, B): 0, 3;
(A, C): 1, 0;
(B, C): 0, 0.5;
(B, D): 1, 0;
(C, D): 0, 3;
player1:  A, D;
player2:  A, D;
```

The output is:

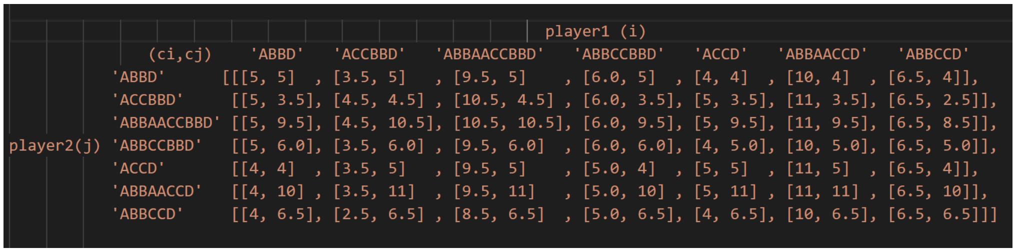
```
player1, Cost = 4.5:  A, C, B, D;
player2, Cost = 4.5:  A, C, B, D;
```

```
player1, Cost = 5:  A, C, D;
player2, Cost = 3.5:  A, C, B, D;
player1, Cost = 6.5:  A, B, C, D;
player2, Cost = 2.5:  A, C, B, D;
player1, Cost = 4:  A, C, D;
player2, Cost = 5.0:  A, B, C, B, D;
player1, Cost = 3.5:  A, C, B, D;
player2, Cost = 5:  A, C, D;
player1, Cost = 5.0:  A, B, C, B, D;
player2, Cost = 4:  A, C, D;
player1, Cost = 5:  A, C, D;
player2, Cost = 5:  A, C, D;
player1, Cost = 6.5:  A, B, C, D;
player2, Cost = 4:  A, C, D;
player1, Cost = 2.5:  A, C, B, D;
player2, Cost = 6.5:  A, B, C, D;
player1, Cost = 4:  A, C, D;
player2, Cost = 6.5:  A, B, C, D;
player1, Cost = 6.5:  A, B, C, D;
player2, Cost = 6.5:  A, B, C, D;
```

This indicate that there are 11 Nash equilibrium.
The topology is as follow:



And the cost table is as follow:



```
                                                                  player1 (i)
                (ci,cj)     'ABBD'    'ACCBBD'    'ABBAACCBBD'   'ABBCCBBD'    'ACCD'    'ABBAACCD'    'ABBCCD'
             'ABBD'      [[[5, 5]   , [3.5, 5]    , [9.5, 5]     , [6.0, 5]   , [4, 4]   , [10, 4]   , [6.5, 4]],
             'ACCBBD'     [[5, 3.5], [4.5, 4.5] , [10.5, 4.5] , [6.0, 3.5], [5, 3.5], [11, 3.5], [6.5, 2.5]],
             'ABBAACCBBD' [[5, 9.5], [4.5, 10.5], [10.5, 10.5], [6.0, 9.5], [5, 9.5], [11, 9.5], [6.5, 8.5]],
player2(j)   'ABBCCBBD'   [[5, 6.0], [3.5, 6.0] , [9.5, 6.0]  , [6.0, 6.0], [4, 5.0], [10, 5.0], [6.5, 5.0]],
             'ACCD'       [[4, 4]  , [3.5, 5]   , [9.5, 5]    , [5.0, 4]  , [5, 5]  , [11, 5]  , [6.5, 4]],
             'ABBAACCD'   [[4, 10] , [3.5, 11]  , [9.5, 11]   , [5.0, 10] , [5, 11] , [11, 11] , [6.5, 10]],
             'ABBCCD'     [[4, 6.5], [2.5, 6.5] , [8.5, 6.5]  , [5.0, 6.5], [4, 6.5], [10, 6.5], [6.5, 6.5]]]
```

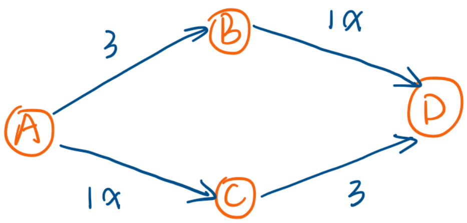
4. Using "one" mode and 3 players. The contain of the input file is:

```
(A, B): 0, 3;
(A, C): 1, 0;
(B, D): 1, 0;
(C, D): 0, 3;
player1:  A, D;
player2:  A, D;
player3:  A, D;
```

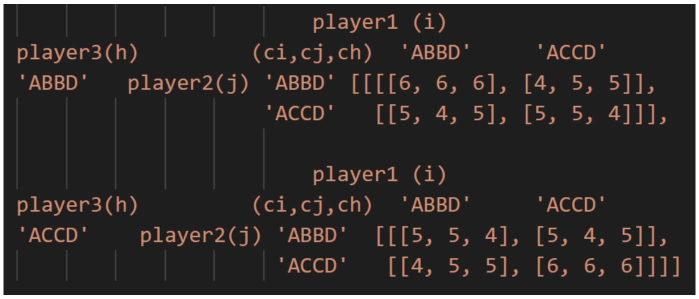
The output is:

```
player1, Cost = 4:  A, C, D;
player2, Cost = 5:  A, B, D;
player3, Cost = 5:  A, B, D;
player1, Cost = 5:  A, B, D;
player2, Cost = 4:  A, C, D;
player3, Cost = 5:  A, B, D;
player1, Cost = 5:  A, C, D;
player2, Cost = 5:  A, C, D;
player3, Cost = 4:  A, B, D;
player1, Cost = 5:  A, B, D;
player2, Cost = 5:  A, B, D;
player3, Cost = 4:  A, C, D;
player1, Cost = 5:  A, C, D;
player2, Cost = 4:  A, B, D;
player3, Cost = 5:  A, C, D;
player1, Cost = 4:  A, B, D;
player2, Cost = 5:  A, C, D;
player3, Cost = 5:  A, C, D;
```

This indicate that there are 6 Nash equilibrium.
The topology is as follow:



And the cost table is as follow:



```
                           player1 (i)
player3(h)          (ci,cj,ch)  'ABBD'      'ACCD'
'ABBD'   player2(j) 'ABBD' [[[[6, 6, 6], [4, 5, 5]],
                    'ACCD'   [[5, 4, 5], [5, 5, 4]]],

                           player1 (i)
player3(h)          (ci,cj,ch)  'ABBD'      'ACCD'
'ACCD'    player2(j) 'ABBD'  [[[5, 5, 4], [5, 4, 5]],
                     'ACCD'   [[4, 5, 5], [6, 6, 6]]]]
```

. . .

References

[BE] Borodin & El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press.