Assignment 4 v1

Assignment 4 requires **one project** and **one brief essay**.

Project

An implementation of the **synchronous EIG-based Byzantine agreement** algorithm, working on the complete graph.

The project shall consist of a single source file, containing definitions for **Node** and for **Arcs**, an adapted version of our previous Arcs (not part of the algorithm).

In the submitted version, your own **standard output stream** (Console.Out) must fully conform to the sample output described below. This is very important for our automarker.

The **appendix** contains snapshots with our visual Byzantine demo running a sample scenario.

Essay

Select and critically **review** any interesting paper(s) which discuss(es) one of the following topics (your choice):

- **Byzantine** algorithms and **blockchains**.
- **Authenticated Byzantine** algorithms.
- **Asynchronous Byzantine** algorithms.
- **Practical Byzantine** algorithms (less costly).

Your review should be structured and formatted as a short conference paper, well-argued and supported by additional evidence, e.g. references/citations. It must adhere to a strict page limit, **up to 3 pages** (without references).

Note: Such topics appear under various names, such as: Byzantine / distributed / fault-tolerant generals / algorithm / agreement / consensus / protocol … not all for the same problem, but many are. E.g. just from the Dijkstra prize list: 2001, 2005, 2007, 2010, 2015, 2017 (https://en.wikipedia.org/wiki/Dijkstra_Prize)

If needed, please don't hesitate to check with us about the suitability of your selected topic / papers.

Config

Arcs reads the config file from **stdin**, e.g.:

```
4 2 0                                // N L V0
1 0 1 0 0 1 1 011 011 011 011        // P V F script
2 0 0                                // P V F
4 1 0
3 1 0
```

There are $N+1$ lines in total, where N = # of process nodes, N in 1..9. End -of-line comments are optional and here indicate the meaning of each value: N = # of process nodes; L = # of EIG levels; V_0 (well known); P = process number (1..N); V = initial choice; F = failure flag (0: loyal, 1: faulty).

Each faulty process line P continues with a script indicating the messages that that process P is expected to send. Conceptually, there are $L \cdot N$ blocks of values, in order:

- S=1: N blocks of length 1
- S=2: N blocks of length N-1
- S=3: N blocks of length N-2
- …
- S=L: N blocks of length N-L+1

For each given level S, the N blocks are to be sent to processes 1, 2, …, N, in this order. Each block represents the level S-1 values that need to be sent, in left-to-right order (considering an ordered EIG tree) - more precisely, what values that faulty process claims to have recorded at level S-1.

Faulty-script conventions (similar to our **demo**):

- legal sequence digits are **0**, **1**, **2, 3**
- **0** and **1** indicate the two values exchanged by loyal participants
- **2** indicates **null**, missing/corrupt messages – to be replaced by $\mathbf{v_0}$
- **3** indicates that the faulty participant sends the **correct** value at this position
- **illegal** characters are assumed to be **2**
- **missing** characters are assumed to a **repeat** of the **last** preceding digit (0, 1, 2, 3)
- internal faulty script **spaces** are **optional** and only relevant for readability
- **extra** characters are **discarded**

For example, the following faulty scripts are equivalent:

```
0 0 0 1 011 021 112 222
0001 0110211122
000101102111xxxxxxxxxxxxxxxxxx
```

Workflow

There are 2·*L*+1 steps, numbered *S* = 0, 1, 2, …, 2·*L*. Step *S* = 0 is the initialisation; steps *S* = 1, 2, …, *L* are the top-down messaging rounds; steps *S* = *L*+1, *L*+2, …, 2·*L* are the bottom-up evaluation.

S=0: Arcs sends init messages to each of the *N* processes, in order *P* = 1, 2, …, *N*. Each init message contains *N*, *P*, *V*, *V0*, *L*. For a faulty node, the message also includes the required faulty script. Process *P* sets its EIG root (level 0) value to *V*, and then prints this value in the format: [0 *P V*]. Finally, its responds by sending its level *S*=1 messages to Arcs.

S=1, 2, … *L*-1: Arcs sends all due level *S* messages to each of the *N* processes, in order *P* = 1, 2, …, *N*. Process *P* fills its level *S* top-down values, and then prints these, in left-to-right order, in the format [*S P values*], separating sibling groups by single spaces. Finally, it responds by sending its level *S*+1 messages to Arcs. Note that each sibling group has *N*+1-*S* values.

S=*L*. Same as above, but we are at the leaves level, where there are no more messages to send. Conceptually, top-down values become bottom-up values. Processes respond with “empty” messages.

S=*L*+1, *L*+2, …, 2·*L*. Arcs send “empty” messages to each of the *N* processes, in order *P* = 1, 2, …, *N*. Process *P* evaluates its level 2·*L*-bottom-up values, using the same format as above.

Obviously, for *S*=2·*L*, processes print their final decisions, in the format [*S P V*].

Sample printout corresponding to the above config file (*N*=4, etc):

```
0 1 0
0 2 0
0 3 1
0 4 1
1 1 0011
1 2 0011
1 3 1011
1 4 1011
2 1 011 000 111 111
2 2 011 000 111 111
2 3 011 000 111 111
2 4 011 000 111 111
3 1 1011
3 2 1011
3 3 1011
3 4 1011
4 1 1
4 2 1
4 3 1
4 4 1
```

Notes

You are free to select actual message format that suits you best. Important is to design a credible simulation, where all contacts between nodes is realised via messages gathered and forwarded by Arcs.

Please don't hesitate to ask if you need clarifications. Thanks!

Submit electronically:

(1) Your programs, as single source file, to the COMPSCI automarker, and

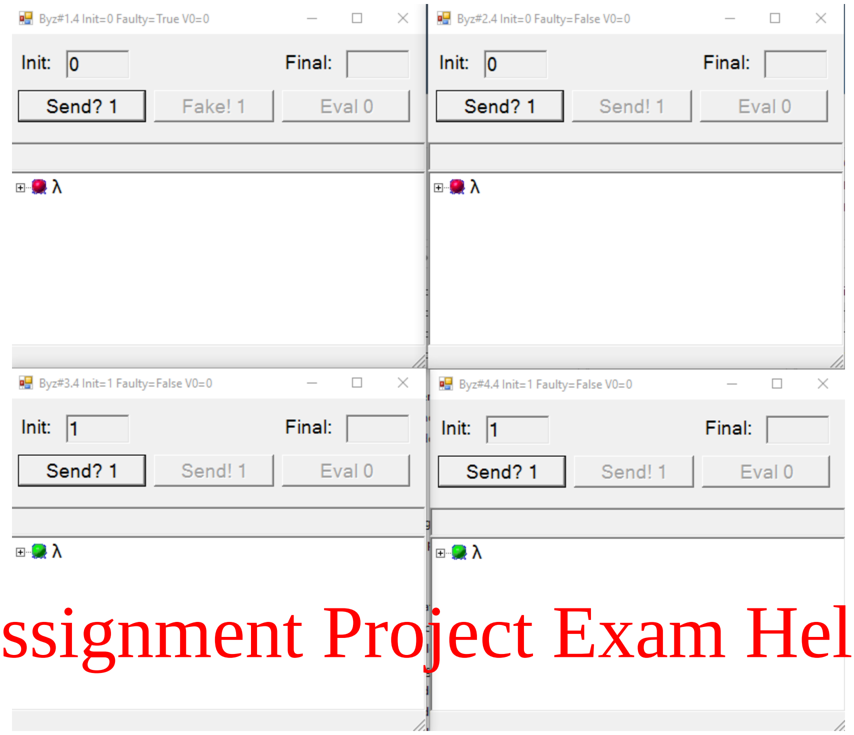
(2) Your essay as PDF, to Canvas.

Deadline: Monday 12 October, 2020, 23:00.

Do not leave it for the last minute, please. Remember that you can resubmit and, by default, we only consider your last submission.

Late submissions will incur penalties, 0.25% off for each hour late, for up to eight days.

APPENDIX



Byz#1.4 Init=0 Faulty=True V0=0

Init: 0 Final:

⊞ 🟥 λ

Byz#2.4 Init=0 Faulty=False V0=0

Init: 0 Final:

⊞ 🟥 λ

Byz#3.4 Init=1 Faulty=False V0=0

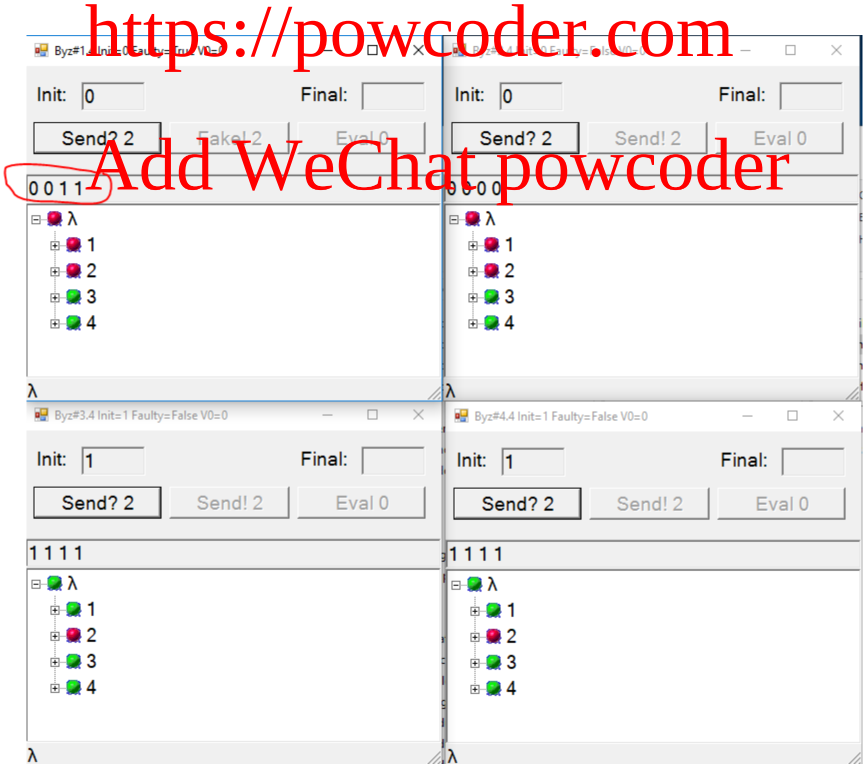
Init: 1 Final:

⊞ 🟩 λ

Byz#4.4 Init=1 Faulty=False V0=0

Init: 1 Final:

⊞ 🟩 λ



Byz#1.4 Init=0 Faulty=True V0=0

Init: 0 Final:

0 0 1 1

```
⊟ 🟥 λ
    ⊞ 🟥 1
    ⊞ 🟥 2
    ⊞ 🟩 3
    ⊞ 🟩 4
```

λ

Byz#2.4 Init=0 Faulty=False V0=0

Init: 0 Final:

0 0 0 0

```
⊟ 🟥 λ
    ⊞ 🟥 1
    ⊞ 🟥 2
    ⊞ 🟩 3
    ⊞ 🟩 4
```

λ

Byz#3.4 Init=1 Faulty=False V0=0

Init: 1 Final:

1 1 1 1

```
⊟ 🟩 λ
    ⊞ 🟩 1
    ⊞ 🟥 2
    ⊞ 🟩 3
    ⊞ 🟩 4
```

λ

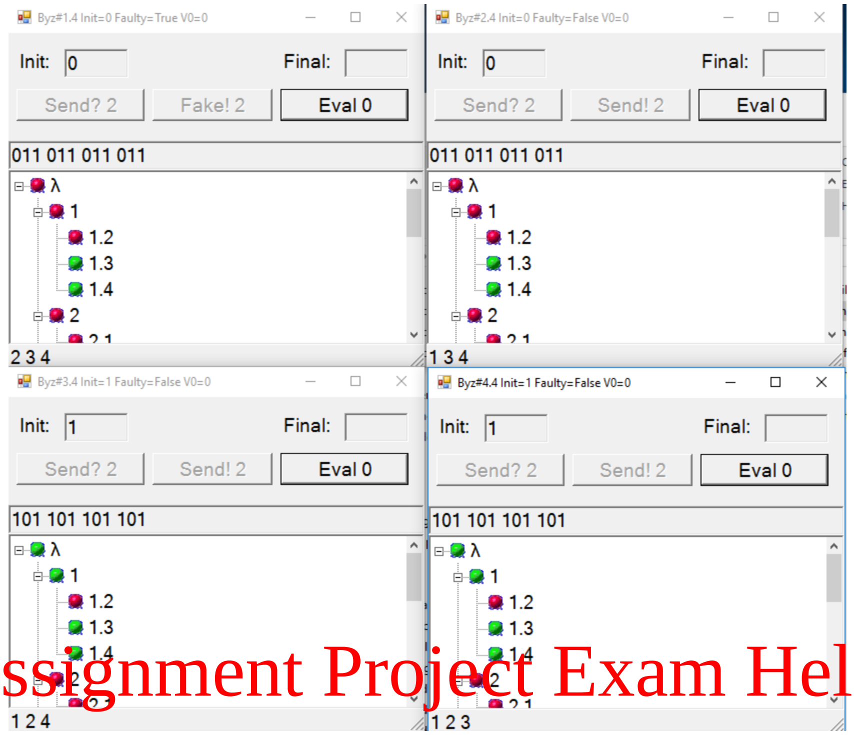
Byz#4.4 Init=1 Faulty=False V0=0

Init: 1 Final:

1 1 1 1

```
⊟ 🟩 λ
    ⊞ 🟩 1
    ⊞ 🟥 2
    ⊞ 🟩 3
    ⊞ 🟩 4
```

λ



Byz#1.4 Init=0 Faulty=True V0=0

Init: 0 Final:

011 011 011 011

```
⊟🔴 λ
  ⊟🔴 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🔴 2
     🔴 2.1
```

2 3 4

Byz#2.4 Init=0 Faulty=False V0=0

Init: 0 Final:

011 011 011 011

```
⊟🔴 λ
  ⊟🔴 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🔴 2
     🔴 2.1
```

1 3 4

Byz#3.4 Init=1 Faulty=False V0=0

Init: 1 Final:

101 101 101 101

```
⊟🟢 λ
  ⊟🟢 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🔴 2
     🔴 2.1
```

1 2 4

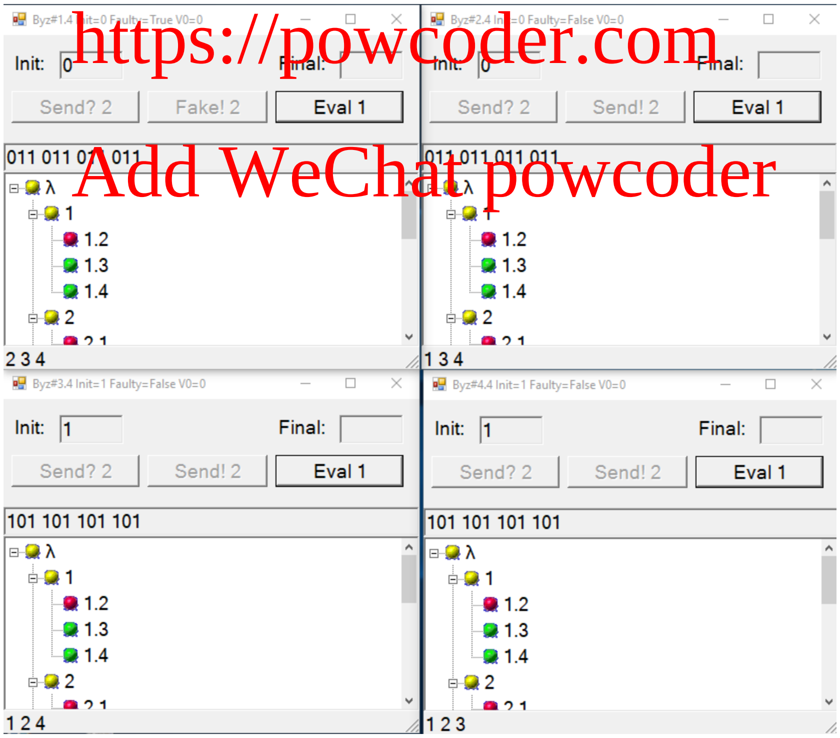
Byz#4.4 Init=1 Faulty=False V0=0

Init: 1 Final:

101 101 101 101

```
⊟🟢 λ
  ⊟🟢 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🔴 2
     🔴 2.1
```

1 2 3



Byz#1.4 Init=0 Faulty=True V0=0

Init: 0 Final:

011 011 011 011

```
⊟🟡 λ
  ⊟🟡 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🟡 2
     🔴 2.1
```

2 3 4

Byz#2.4 Init=0 Faulty=False V0=0

Init: 0 Final:

011 011 011 011

```
⊟🟡 λ
  ⊟🟡 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🟡 2
     🔴 2.1
```

1 3 4

Byz#3.4 Init=1 Faulty=False V0=0

Init: 1 Final:

101 101 101 101

```
⊟🟡 λ
  ⊟🟡 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🟡 2
     🔴 2.1
```

1 2 4

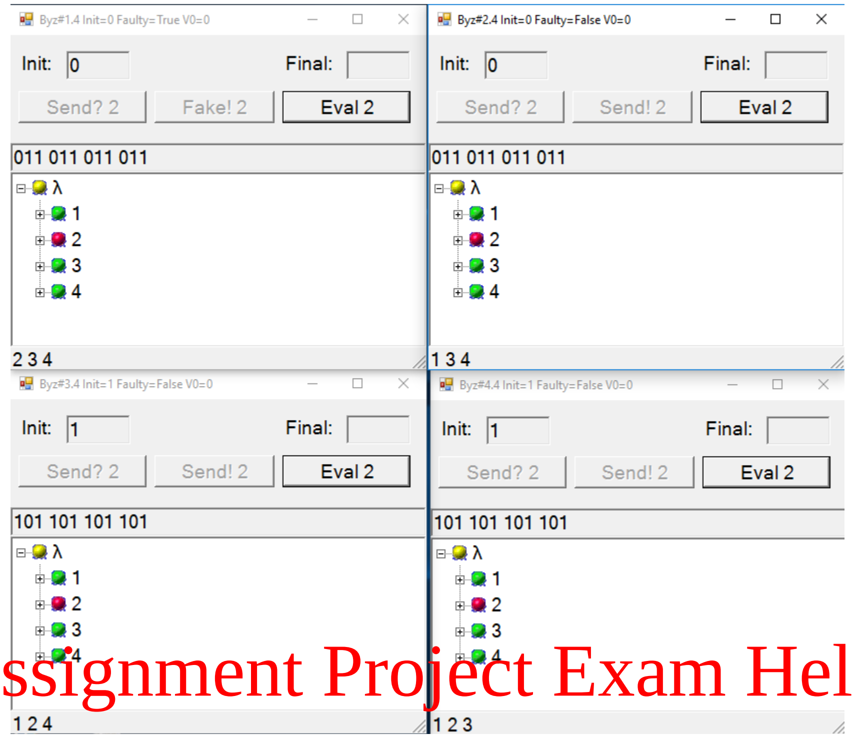
Byz#4.4 Init=1 Faulty=False V0=0

Init: 1 Final:

101 101 101 101

```
⊟🟡 λ
  ⊟🟡 1
     🔴 1.2
     🟢 1.3
     🟢 1.4
  ⊟🟡 2
     🔴 2.1
```

1 2 3



Byz#1.4 Init=0 Faulty=True V0=0

Init: 0 Final:

011 011 011 011

2 3 4

Byz#2.4 Init=0 Faulty=False V0=0

Init: 0 Final:

011 011 011 011

1 3 4

Byz#3.4 Init=1 Faulty=False V0=0

Init: 1 Final:

101 101 101 101

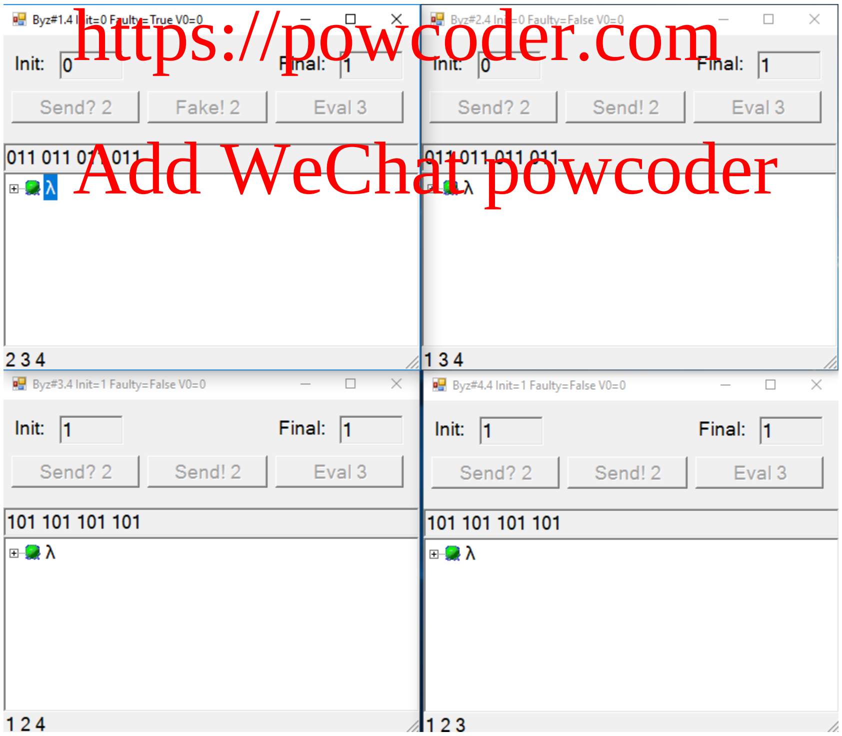
1 2 4

Byz#4.4 Init=1 Faulty=False V0=0

Init: 1 Final:

101 101 101 101

1 2 3



Byz#1.4 Init=0 Faulty=True V0=0

Init: 0 Final: 1

011 011 011 011

2 3 4

Byz#2.4 Init=0 Faulty=False V0=0

Init: 0 Final: 1

011 011 011 011

1 3 4

Byz#3.4 Init=1 Faulty=False V0=0

Init: 1 Final: 1

101 101 101 101

1 2 4

Byz#4.4 Init=1 Faulty=False V0=0

Init: 1 Final: 1

101 101 101 101

1 2 3