Stacey Frasier
Nino Vilagi
CPSC 474

Lamport's Logical Clocks

Conceptual Implementation of Lamport's Logical Clocks using string matrices.

Notes for Usage

1. The program asks the user for the following:
    i. The file path of the input file for testing (ex: ./ex1.txt)
    ii. Name of the output file the results will be written to (ex: out1.txt) *Note: If the file D.N.E it will create one. If the file exists the output will be appended to the file*
2. Events within a process should be separated by a *space*. Each process should be separated by an endline (*enter*).
3. To test each algorithm, please use the following lines of code within the current directory of this project: --> calculate algorithm: `python3 project1_calculate.py` --> verify algorithm `python3 project1_verify.py`
4. If the output is incorrect within the *verify* algorithm, the result will only display in terminal.
5. We have included one *.txt* file per example provided in the *Project 1 Outline*

Project 1: Pseudocode

```
def_algo_calculate(N, M, in[N][[M]):

Class process
       queue individual_process_queue    # events in current process
                                         # → (single row of matrix)
       int private_count = 0              # LC-value count for current process
       queue send_queue                  # keep track of when sends happen
       Queue results queue

       While individual_process_queues!=empty #will have a queue for each process
              If send event
                     individual_process_queue.pop
                     private_count++
                     results.push(private_count)
                     send_queue.push(private_count)

              if receive event

                     individual_process_queue.pop
                     private_count = max(private_count,send_queue.pop)+1
                     results.push(private_count)
                     global_count = private_count


              if NULL event              # NULL/ complete ‘event’
                     individual_process_queue.pop
                     results.push(0)

              else:                              # internal event
                     Individual_process_queue.pop
                     private_count++
                     results.push(private_count)
       print(results)

def_algo_verify(n, m, inp[n][m]):
       # initiate output to all internal events
       out = list[n][m]
       # two dict will store LC-value as key, and row, col as values
       rloc = dict{}
       sloc = dict{}
       # two lists will store LC-value to store in the output
       send = list[]
       receive = list[]
       for i in cols:
              for j in rows:
                     s_value = 0
                     # event is null
                     if inp[i][j] == 0:
```

```
                        out[i][j] = "NULL"
                # receive event
                # first event is not 1
                elif j == 0 && inp[i][j] != 1:
                        receive.append(inp[i][j])
                        s_value = inp[i][j] - 1
                        send.append(s_value)
                        rloc[inp[i][j]] = [i, j]
                # avoid checking outside array size
                elif j+1 < rows && inp[i][j+1] > inp[i][j] + 1:
                        receive.append(inp[i][j+1])
                        s_value = inp[i][j+1]-1
                        send.append(s_value)
                        rloc[inp[i][j+1]] = [i, j+1]
                # an internal event.
                else:
                        continue

                # find the send if it exists
                find = False
                for x in col:
                        if find == True: # end search when found
                                break
                        for y in rows:
                                if inp[x][y] == s_value:
                                        sloc[inp[x][y]] = [x, y]
                                        find = True
                                        break
                if find == False:   # send value d.n.e
                        return "Output is Incorrect"

Sort send And receive
s_count = 1 and r_count = 1
for s in send:
        s_out = sloc.get(s)
        s_row = s_out[0]
        s_col = s_out[1]
        out[s_row][s_col] = "s + s_count”
        s_count += 1
for r in receive:
        r_out = rloc.get(r)
        r_row = r_out[0]
        r_col = r_out[1]
        out[r_row][r_col] = "r + r_count”
        r_count += 1
return out
```