Matthew Braun
CPSC 474 Project 1 Report

Pseudocode Report

ALGORITHM CALCULATE

```
def calculate(N, Matrix):
    map = [0 * 10]                                  # create process receive map length 10
    result = [[0 * len(Matrix[0])] * len(Matrix)]   #create result matrix same size as Matrix
    row_indexes = [0 * len(Matrix)]                 # list of 0's, size equal to numbr of rows
    while row_indexes != [len(Matrix[0]) * len(matrix)]:   # while there are still unvisited matrix entries
    for r = 0 to len(Matrix):
        for c = row_indexes[r] to len(Matrix[0]):
            if Matrix[r][c] == "":                  # don't do anything if NULL string
                pass
            else if Matrix[r][c][0] == 's':         # if this is a send event
                index = Matrix[r][c][1] - '0'       # convert 2nd char to number
                if c == 0:
                    result[r][c] = 1;
                else:
                    result[r][c] = Matrix[r][c-1] + 1
                map[index] = result[r][c]           # populate map with this events time
            else if Matrix[r][c][0] == 'r':         # if this is a receive event.
                index = Matrix[r][c][1] - '0'
                if c == 0:
                    result[r][c] = map[index] + 1
                else:   # value equals maximum clock time between previous event and the send event
                    result[r][c] = max(map[index], Matrix[r][c-1]) + 1
            else:
                if c == 0:
                    result[r][c] = 1
                else:
                    result[r][c] = result[r][c-1] + 1
            row_indexes[r] += 1                     # increment the column index for this row.
    return result
```

ALGORITHM VERIFY

```
def verify(N, M, Matrix):
    result = [[“” * len(Matrix[0])] * len(Matrix)]          # create result matrix same size as input Matrix
    receive_map = {}            # two dictionaries with integer key values and value is a list of lists of integers
    send_map = {}               # the list of integers will be all the rows and columns pertaining to that key value.
    valid_input = True                                      # bool to track if input is valid
    letter = “a”
    for r =0 to len(Matrix):
        for c = 0 to len(Matrix[0]):
            if Matrix[r][c] == 0:
                continue
            if c == 0:
                if Matrix[r][c] == 1:
                    result[r][c] = letter
                    increment_letter()          # function to increment letter from a -> b -> c, etc.
                    if 1 in send_map:
                        send_map[1].append([r, c])
                    else:
                        send_map[1] = [[r, c]]
                else:
                    if 1 in receive_map:
                        receive_map[1].append([r, c])
                    else:
                        Receive_map[1] = [[r, c]]
            else:
                val = Matrix[r][c]
                # Clock events must be in increasing order. Very import condition to check!
                if val < Matrix[r][c-1]:
                    valid_input = False
                    return
                # if numbers are sequential, this entry is a send candidate.
                if val == Matrix[r][c-1] + 1:
                    result[r][c] = letter
                    increment_letter()
                    if val in send_map:
                        send_map[val].append([r, c])
                    else:
                        send_map[val] = [[r, c]]
                else:   # if numbers are not sequential, this MUST be a receive event.
                    if val in receive_map:
                        receive_map[val].append([r, c])
                    else:
                        receive_map[val] = [[r, c]]
```

```
order_dictionary_by_key()                    # in C++ Map using iterators this is automatic.
receive = “r1”
send = “s1”                                  # send and receive events start at 1
for rec_k, rec_v in receive_map:
        for indexes in rec_v:
                if rec_k - 1 not in send_map:        # if no send candidates with correct clock time,
                        valid_input = False          # then this is invalid input.
                        return
                else:
                        # populate the result matrix with the proper receive event string tag.
                        i = indexes[0]
                        j = indexes[1]
                        result[i][j] = receive

                        # populate the result matrix with the proper send event string tag
                        length = len(send_map[rec_k - 1])
                        i = send_map[rec_k - 1][length - 1][0]
                        j = send_map[rec_k - 1][length - 1][1]
                        result[i][j] = send

                        # If there are more send events with this same key value or if we have reached the last
                        # receive event that needs this key value, it is safe to increment the receive and send
                        # counter. Otherwise, it is not safe because there could be one send event to multiple
                        # receive events for different processes.
                        if length > 1 or indexes == rec_v[-1]:
                                send_map[rec_k - 1].pop_back()
                                receive[1] += 1              #increment receive event counter
                                send[1] += 1                 #increment send event counter

if !valid_input:
        return “INCORRECT”
else:
        return result
```