Klenovszky Áron H9MRBV

findMaxSCC(n : ℕ , a\0 : ℕ[n][n]) : Queue

```
allSCC : Queue
isSCC\0 : ℕ[n]
i = 0 ... n-1
    isSCC[i] = 0
adjacent : E1*[n]
i = 0 ... n-1
    j = 0 ... n-1
        a[i,j] = 1
            e := new E1 ; e -> key := j
            e -> next := adjacent[i]
            adjacent[i] := e
        else
            Skip
i = 0 ... n-1
    isSCC[i] = 0
        scc : Queue ; scc.add(i)
        j := (i+1) ... (n-1)
            isSCC[j] = 0 ∧ isPath(i, j, adjacent, n) ∧ isPath(j, i, adjacent, n)
                isSCC[j] = 1
                scc.add(j)
            else
                Skip
        allSCC.add(scc)
    else
        Skip
```

folytatás

maxQ := allSCC.first()

allSCC.length() $\neq$ 0

maxQ.length() < allSCC.first().length()

maxQ := allSCC.rem()	Skip

return maxQ

isPath(source : $\mathbb{N}$, destination : $\mathbb{N}$, adjacent : E_1^*[n] n : $\mathbb{N}$) : $\mathbb{L}$

visited\0 : $\mathbb{N}$[n];

i = 0 ... n - 1

visited[i] = 0

return search(source, destination, adjacent, visited);

$search(current : \mathbb{N},\ destination : \mathbb{N},\ adjacent : E1^*[n],\ visited \setminus O : \mathbb{N}[n]) : \mathbb{L}$

- **if** current = destination
  - **then:** return true
  - **else:** skip
- visited[current] := 1
- c := adjacent[current]
- **while** $c \neq \emptyset$
  - **if** visited[c -> key] = 0
    - **then:**
      - **if** search(c -> key, destination, adjacent, visited)
        - **then:** return true
        - **else:** Skip
    - **else:** Skip
- return false