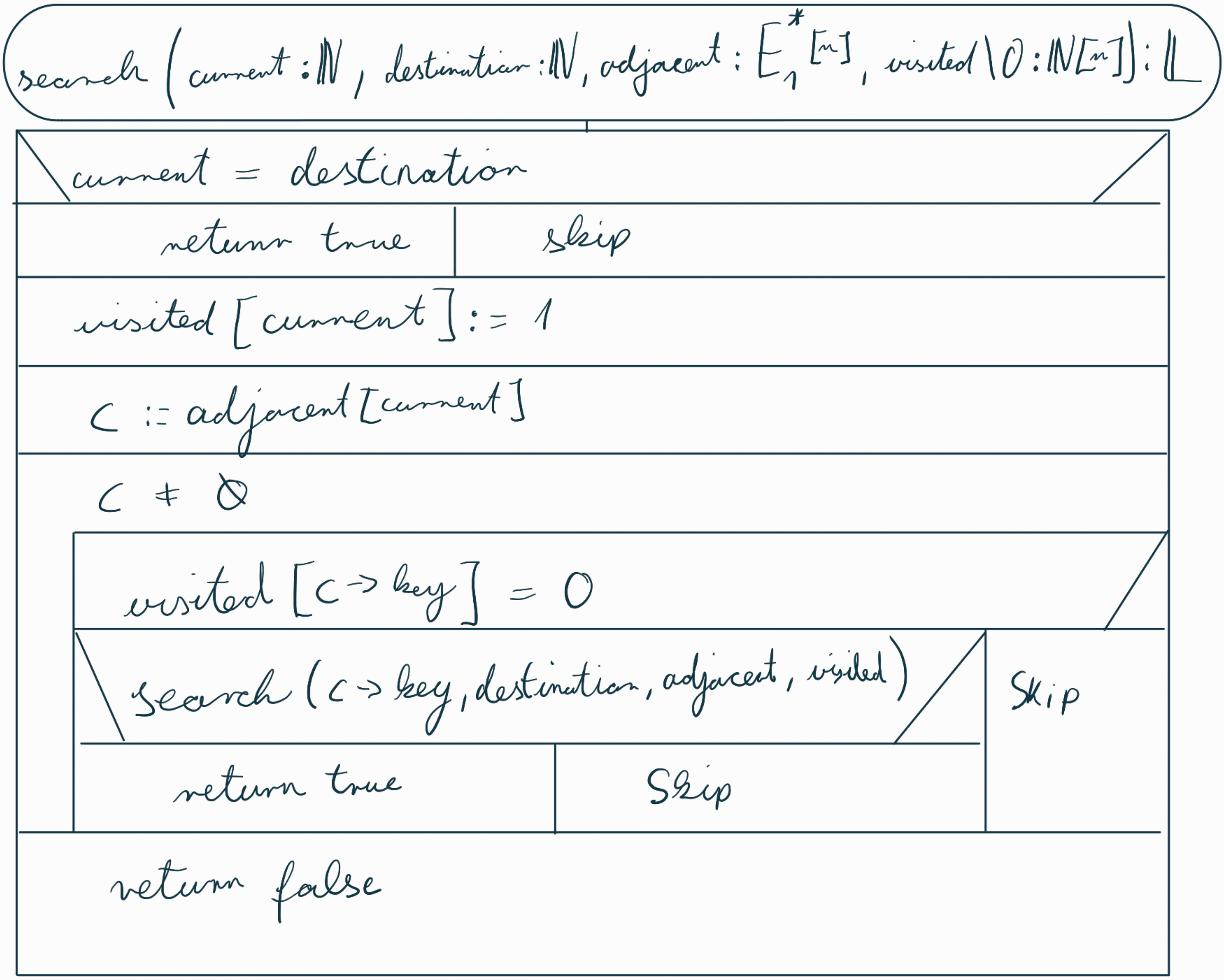
```
findMaxSCC(n : ℕ, a\0 : ℕ[n][n]) : Queue
    allSCC : Queue
    isSCC\0 : ℕ[n]
    i = 0 ... n - 1
        isSCC[i] = 0
    adjacent : $E_1^*$[n]
    i = 0 ... n - 1
        j = 0 ... n - 1
            a[i,j] = 1
                e := new $E_1$ ;   e -> key := j
                e -> next := adjacent[i]
                adjacent[i] : e
            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)
                Skip
            allSCC.add(scc)
        SKIP
```

folytatás

```
maxQ := allSCC.first()
allSCC.length() $\neq 0$
    maxQ.length() < allSCC.first().length()
        then: maxQ := allSCC.rem()
        else: 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 : ℕ, destination : ℕ, adjacent : $E_1^*[n]$, visited \ 0 : ℕ[n]) : 𝕃
    current = destination
        return true
        skip
    visited[current] := 1
    c := adjacent[current]
    c ≠ ∅
        visited[c -> key] = 0
            search(c -> key, destination, adjacent, visited)
                return true
                Skip
            Skip
    return false
```