Graphs

- `G.n`
- `G.adj(x)`
- `distance(G, x, v)` - lancia bfs a partire da `x` e salva i risultati in `v`
- `NODE n`
- `G.insertNode(u)`
- `G.insertEdge(u, v)`

Stack

- `STACK s = new STACK()`
- `s.push(x)`
- `s.pop()`
- `s.top()`
- `s.isEmpty()`

Tree

- `TREE T = new TREE()`
- `T.left()`
- `T.right()`
- `count(TREE T)` ritorna il numero di nodi dell'albero
- `T.deleteLeft()`
- `T.deleteRight()`

Queue

- `QUEUE q = new QUEUE()`
- `q.enqueue(x)`
- `q.dequeue()`
- `q.top()`

Set

- `SET s = new SET()`
- `s.size()`
- `s.contains(ITEM i)`
- `s.insert(ITEM i)`
- `s.remove(ITEM i)`
- `union(SET a, SET b)`
- `intersection(SET a, SET b)`
- `difference(SET a, SET b)`

Dictionary

- `DICTIONARY d = new DICTIONARY()`
- `d.lookup(ITEM i)`
- `d.insert(ITEM key, value)`
- `d.remove(ITEM key)`

```
local function lowerBound(v, k, i, j)
    if i == j then return i end

    local m = math.floor((i+j)/2)

    if v[m] >= k then
      return lowerBound(v, k, i, m)
    else
      return lowerBound(v, k, m+1, j)
    end
end
```

Ritorna il primo indice in cui si trova l'elemento k

Se `k` non è presente nel vettore, allora viene ritornato l'indice in cui andrebbe inserito per mantenere l'ordinamento. NON considera casi in cui l'elemento va inserito all'inizio o alla fine

---

```
local function upperBound(v, k, i, j)
  if i == j then return i end

  local m = math.ceil((i+j)/2)

  if v[m] > k then
    return upperBound(v, k, i, m-1)
  else
    return upperBound(v, k, m, j)
  end
end
```

Ritorna l'ultimo indice in cui si trova l'elemento k

```
local function bfs(graph, start)
  local queue = {}
  local distances = {}

  for i = 1, graph.size do
    distances[i] = -1
  end

  distances[start] = 0
  table.insert(queue, start)

  while #queue ~= 0 do
    local currVertex = queue[1]
    table.remove(queue, 1)
    for _, vTo in ipairs(graph[currVertex]) do
      if distances[vTo] == -1 then
        distances[vTo] = distances[currVertex] + 1
        table.insert(queue, vTo)
      end
    end
  end
  return distances
end
```

```
local function locateRec(t, index, level, pos)

  if t == nil then return nil end

  if t.index==index then return level, pos end

  local levelL, posL = locateRec(t.left, index, level+1, 2*pos)
  if levelL then return levelL, posL end

  local levelR, posR = locateRec(t.right, index, level+1, 2*pos
    +1)
  if levelR then return levelR, posR end

  return nil

end
```

Teorema

Sia $a \geq 1$, $b > 1$, $f(n)$ asintoticamente positiva, e sia

$$T(n) = \begin{cases} aT(n/b) + f(n) & n > 1 \\ d & n \leq 1 \end{cases}$$

Sono dati tre casi:

(1)	$\exists \epsilon > 0 : f(n) = O(n^{\log_b a - \epsilon})$	$\Rightarrow$	$T(n) = \Theta(n^{\log_b a})$
(2)	$f(n) = \Theta(n^{\log_b a})$	$\Rightarrow$	$T(n) = \Theta(f(n) \log n)$
(3)	$\exists \epsilon > 0 : f(n) = \Omega(n^{\log_b a + \epsilon}) \wedge$ $\exists c : 0 < c < 1, \exists m \geq 0 :$ $af(n/b) \leq cf(n), \forall n \geq m$	$\Rightarrow$	$T(n) = \Theta(f(n))$