Algorithm 1: MTMO
Input: G: input graph, k: subtree size
Output: Frequencies of all k-subtrees of graph G

```
1: for each u ∈ G do
2:    Visited[u] ← true
3:    Sub_Start_0 ← u //root vertex
4:    Sub_End_0 ← u
5:    Explore(G, u, Sub_Start, Sub_End, 1, k - 1, Qtree.root)
6:    Visited[u] ← false
7: end for

8: procedure Explore(G, u, Sub_Start, Sub_End, i, reminder, curQTNode)
   //Explore extracts all k-subtrees of graph G rooted in u.
   //Sub_Start: selected starting point of edge, Sub_End: selected ending point of edge.
   //i: current depth of the tree, reminder: number of remaining edges to be selected.
   //curQTNode: the current pointer of the labeled rooted tree.
9:    if reminder = 0 then
10:      if curQTNode.pos = -1 then //only in this case it is required to call AHU algorithm
11:         treeCenter(Sub_Start, Sub_End, level, c1, c2) //treeCenter returns one or two centers.
12:         curQTNode.pos = b2d(CANONICALLABELING(c1)) //b2d converts binary to decimal
13:      end if
14:      IncrementCount(curQTNode.pos) //Increases the counter of array element
15:      return
16:   else
17:      ValList ← Validate(G, Sub_End_{i-1}, u)
18:      for k_i ← 1 to reminder do
19:         if |ValList| < k_i then
20:            return
21:         end if
22:         C ← Initial_Comb(ValList, k_i) //Make the first edge combination selection according
23:         repeat
24:            if Not DuplicateVertex(C.second) then //to select an edge that will expand the subtree size by one
25:               (Sub_start_i, Sub_End_i)=(C.first, C.second)
26:               for each v ∈ Sub_End_i do
27:                  Visited[v] ← true
28:               end for
29:               curQTNode' =Search(Sub_Start, Sub_End, i, reminder, curQTNode)
                   //Searching the labeled rooted tree
30:               Explore(G, u, Sub_Start, Sub_End, i + 1, reminder - k_i, curQTNode')
31:               for each v ∈ Sub_End_i do
32:                  Visited[v] ← false
33:               end for
34:            end if
35:         C ← Next_Comb(ValList, k_i) //Make the next vertex combination selection according
36:         until C=NULL
37:      end for
38:   end if

39: procedure Validate(G, Parents, u) // Validate returns valid edges of the current level.
    //Parents: selected ending point of edge of last layer, u: rooted vertex.
40:    ValList ← NILL
41:    for each v ∈ Parents do
42:       for each w ∈ Neighbor[v] do
43:          if label[u] < label[w] AND NOT Visited[w] then
44:             ValList = ValList + (v, w)
45:          end if
46:       end for
47:    end for
```

48: **return** ValList

49: **Procedure** Search(*Sub_Start*, *Sub_End*, *i*, *reminder*, *curQTNode*)
50: ResultNode ← *curQTNode*
51: **for** *j* ← 1 to |Sub_Start_i| **do**
52: **for** *l* ← 1 to |Sub_End_{i-1}| **do**
53: **if** $Sub\_Start_i[j] = Sub\_End_{i-1}[l]$ **then**
54: break;
55: **end if**
56: **end for**
57: **case** subgraphSize – reminder – |Sub_End_{i-l}| + *l* **of** //computing the parent information
58: 1: ResultNode ← child number 1 of ResultNode
59: 2: ResultNode ← child number 2 of ResultNode
60: 3: ResultNode ← child number 3 of ResultNode
61: 4: ResultNode ← child number 4 of ResultNode
62: 5: ResultNode ← child number 5 of ResultNode
63: 6: ResultNode ← child number 6 of ResultNode
64: 7: ResultNode ← child number 7 of ResultNode
65: 8: ResultNode ← child number 8 of ResultNode
66: 9: ResultNode ← child number 9 of ResultNode
67: 10: ResultNode ← child number 10 of ResultNode
68: 11: ResultNode ← child number 11 of ResultNode
69: **default: write**("The subgrapg size is at most 12")
70: **end case**
71: **end for**
72: **return** ResultNode