Algorithm 1, Find an independent subgraph containing a given vertex.

An independent subgraph is a subgraph that satisfies

with n being the # of vertices, being the # of independent bonds and being the # of free pebbles summed over the vertices of the subgraph.

Recursive version:

1. Start the search at the given vertex. Denote it as vertex i.
2. If vertex I has **2 free pebbles**, it is an independent subgraph by itself. Return.
3. If vertex i has been **visited**, we find an independent subgraph. Return.
4. If vertex i has no edge, or all edge has been previously searched:
   1. The search **fails**. No independent subgraph can be found.
   2. Return.
5. Set result\_i = 0, indsubgraph\_i = [].
6. Mark vertex i as **visited**.
7. If result\_i == # of fixed pebble:
   1. return the indsubgraph\_i.
8. Else if all edges connecting vertex i has been searched:
   1. **Clear** the visited flag of vertex i.
9. Choose an edge connecting vertex i. Denote the other end as vertex j. Mark the edge as **searched**.
10. **Recursion**: search an independent subgraph containing vertex j.
11. If the search finds indsubgraph\_j:
    1. Add edge (i, j) to indsubgraph\_i.
    2. Add indsubgraph\_j to indsubgraph\_i.
    3. increase result\_i by 1.
12. Repeat 7-12

Iterative version:

1. Set satisfy\_vertex=i.
2. Start the search at the given vertex. Denote it as vertex i.
3. If vertex i has 2 free pebbles, it is an independent subgraph by itself. Return.
4. For (vertex k, path, parent q) in depth\_first\_search\_iterator(start=i):
   1. If vertex k has a child vertex c other than q that has **2** free pebbles, or is in **path**, or is in the **indsubgraph**:
      1. Set satisfy\_vertex = k.
      2. Increase result\_k by 1.
      3. Add edge (k, c) to indsubgraph\_k.
   2. If result\_k == # fixed pebbles of vertex k:
      1. Set satisfy\_vertex = q.
      2. Increase result\_q by 1.
      3. Add indsubgraph\_k to indsubgraph\_q.
      4. Repeat step c for vertex q.
   3. Skip the iterations until we search the next edge of satisfy\_vertex.
5. If result\_i = # fixed pebbles of vertex i:
   1. Return indsubgraph.
6. Else:
   1. Search fails.
   2. Return.

Iterative Version

Let’s say we start from vertex a.

if free\_pebble(a) == 2:

vertex a is an independent subgraph itself. Just return.

found[:] = 0

for (v, path, p) in dfs(start=a):

for e in path:

if found(nn)

for nn in v.neighbors():

if (found(nn) == fix\_pebble(nn) or nn in path or nn in indvertex(top\_vertex)):

found(v) ++

indedge(v).append((v, nn))

i = v

for j in path:

if found(i) == fix\_pebble(i):

found(j) ++

i = j

else:

break

The pebble game version:

1. find a free pebble:

for (v, path, parent) in dfs(start=i):

if free\_pebble(v):

return path + [v]

return None

2. transfer a pebble

def transfer\_pebble(path):

start = path[0]

start.free\_pebble ++

for node in path[1:]:

bond = get\_bond(start, node)

bond.reverse()

start = node

start.free\_pebble --

3. find a rigid subgraph

(a, b) = bond

for i in range(2):

path = find\_free\_pebble(a)

transfer\_pebble(path)

for i in range(2):

path = find\_free\_pebble(b)

transfer\_pebble(path)

subgraph = set()

for (v, path, parent) in dfs(start=a, mode="out"): # reserve search on the directed graph

v.visited = true

for (v, path, parent) in dfs(start=b, mode="out"): # reserve search on the directed graph

if v.visited:

subgraph.add(v)

while parent.visited:

continue

4. find a stressed subgraph

(a, b) = bond

for i in range(2):

path = find\_free\_pebble(a)

transfer\_pebble(path)

path = find\_free\_pebble(b)

transfer\_pebble(path)

subgraph=set()

for (v, path, parent) in dfs(start=b):

subgraph.add(v)