# Algorithmics
# Correction Final Exam #3 (P3)

***Solution 1*** (**In the depth of the spanning forest** − *3 points*)

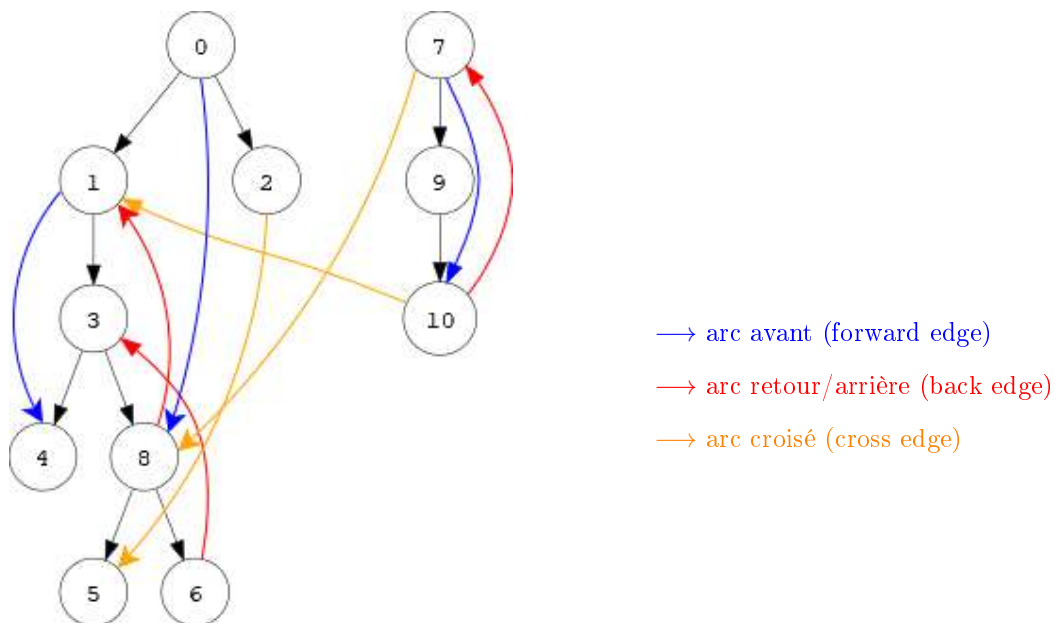1. *Spanning forest and extra-edges for the depth-first search of the graph in figure 1:*



$\longrightarrow$ arc avant (forward edge)

$\longrightarrow$ arc retour/arrière (back edge)

$\longrightarrow$ arc croisé (cross edge)

Figure 1: DFS: Spanning forest

2. *Meeting orders in prefix **pref** and suffix **suff**:*

|          | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
|----------|----|----|----|----|----|----|----|----|----|----|----|
| **pref** | 1  | 2  | 14 | 3  | 4  | 7  | 9  | 17 | 6  | 18 | 19 |
| **suff** | 16 | 13 | 15 | 12 | 5  | 8  | 10 | 22 | 11 | 21 | 20 |

**Solution 2 (Union-Find − 4 points)**

1. Number of vertices of each connected component:

   $C_1$ : 4          $C_2$ : 6          $C_3$ : 4

2. Edges to add: **two** among $5 - 8$  $8 - 12$  $5 - 12$ for example...

3. Among the following chains, those which can not exist in $G$:

   □ $3 \leftrightsquigarrow 7$          ☒ **$11 \leftrightsquigarrow 6$**          ☒ **$0 \leftrightsquigarrow 13$**          □ $4 \leftrightsquigarrow 9$

4. Vector $p$ after adding the edge 7–4:

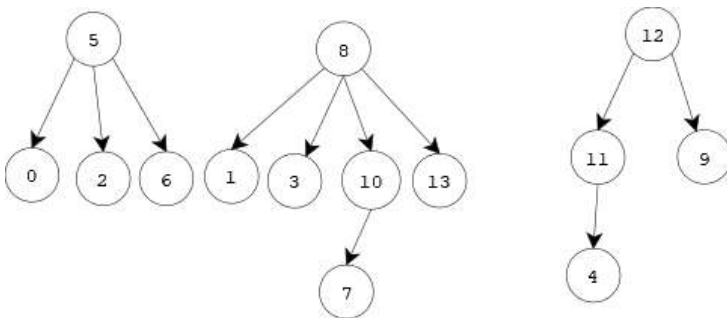| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p$ | 5 | 8 | 5 | 8 | **12** | -4 | 5 | **8** | **-10** | 12 | 8 | 12 | **8** | 8 |



Figure 2: before
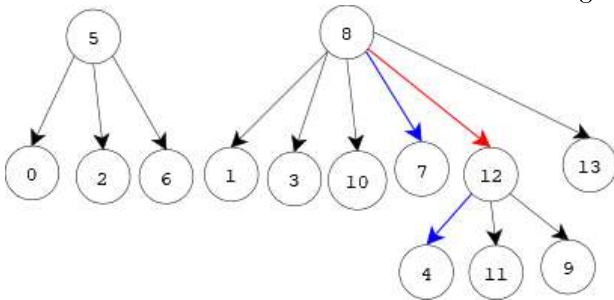


Figure 3: after edge $7 - 4$ added

**Solution 3 (Distance from start − 5 points)**

**Specifications:** dist_range($G$, $src$, $dmin$, $dmax$) returns the list of vertices that are at a distance between $dmin$ and $dmax$ from the vertex $src$ in the graph $G$ (with $0 < dmin \leq dmax$).

```python
def dist_range(G, src, dmin, dmax):
    dist = [None] * G.order
    q = queue.Queue()
    q.enqueue(src)
    dist[src] = 0
    L = []
    while not q.isempty():
        x = q.dequeue()
        if dist[x] >= dmin:
            L.append(x)
        if dist[x] < dmax:
            for y in G.adjlists[x]:
                if dist[y] == None:
                    dist[y] = dist[x] + 1
                    q.enqueue(y)
    return L
```

***Solution 4* (Get cycle – *5 points*)**

**Specifications:**

the function **get_cycle**($G$) returns a cycle of the undirected graph $G$, an empty list if $G$ is acyclic.

**Version 1:** using parent vector

```python
def __get_cycle(G, x, parent):
    """
    DFS on G from x, interrupted at first back edge found
    parent: vertices marked with their parent
    return first back edge found (x, y) or None
    """
    for y in G.adjlists[x]:
        if parent[y] == None:
            parent[y] = x
            get = __get_cycle(G, y, parent)
            if get != None:
                return get
        else:
            if y != parent[x]:
                return (x, y)
    return None

def get_cycle(G):
    parent = [None] * G.order
    s = 0
    get = None
    while s < G.order and get == None:
        if parent[s] == None:
            parent[s] = -1
            get = __get_cycle(G, s, parent)
        s += 1
    L = []
    if get != None:
        (x, y) = get
        L = [x]
        while x != y:
            x = parent[x]
            L.append(x)
        L.append(L[0])
    return L
```

**Version 2:** the cycle is built by the recursive function in going up
Many ways to do it. The difficulty: no longer add vertices when the cycle is complete.

```python
def __get_cycle2(G, x, M, p):
    """
        DFS on G from x
        M: mark vector (boolean)
        p: x's parent
        return (cycle, done):
        - cycle = the vertices of the first cycle found, [] if no cycle
        - done: boolean: is the cycle completed?
    """
    M[x] = True
    for y in G.adjlists[x]:
        if not M[y]:
            (cycle, done) = __get_cycle2(G, y, M, x)
            if cycle:
                if done:
                    return (cycle, True)
                if cycle[0] != y:
                    cycle.append(y)
                return (cycle, cycle[0] == y)
        else:
            if y != p:
                return ([y], False)
    return ([], False)

def get_cycle_2(G):
    M = [False] * G.order
    for s in range(G.order):
        if not M[s]:
            cycle, done = __get_cycle2(G, s, M, -1)
            if cycle:
                return cycle + [cycle[0]]
    return []
```

**Solution 5 (What is this? − 3 points)**

1. The built graph ($NG$):
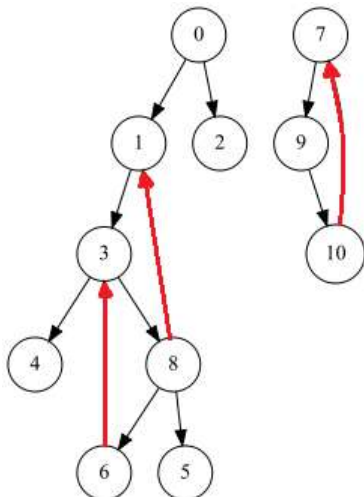


2. For each vertex $s$, during the traversal:

   (a) What does D[$s$] represent?
       D[$s$] is None if the vertex $s$ has not been met. Otherwise it is the depth of $s$ in the spanning forest of the DFS.

   (b) What does P[$s$] represent?
       P[$s$] indicates whether $s$ was encountered in suffix.