# CS130A HW5

Matthew Ho

November 2021

## 1 Question 1.

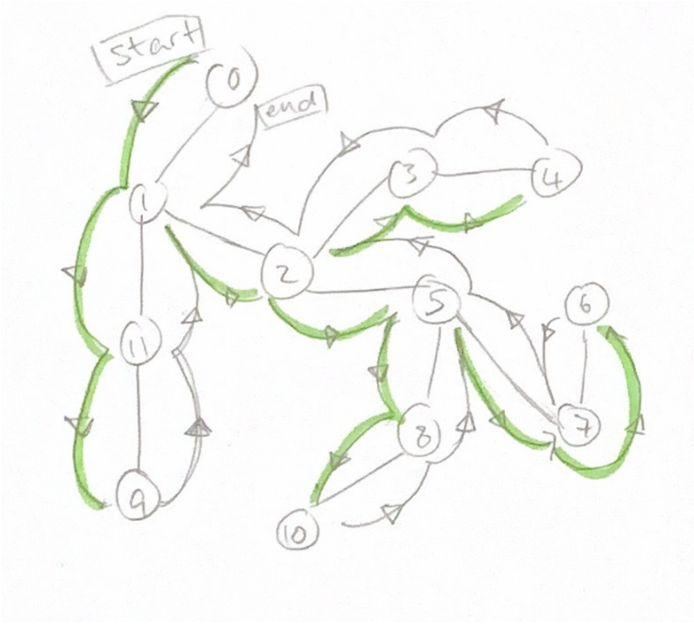
Show how depth-first search works on the graph below. Assume that your source vertex is 0 and that your adjacency list is ordered in decreasing numerical values.

**Steps:**

1. start with 0
2. visit 1
3. visit 11 (first in 1's adj list)
4. visit 9
5. return to 11
6. return to 1
7. visit 2 (next in 1's adj list)
8. visit 5 (first in 2's adj list)
9. visit 8 (first in 5's adj list)
10. visit 10
11. return to 8
12. return to 5
13. visit 7 (next in 5's adj list)
14. visit 6
15. return to 7
16. return to 5
17. return to 2
18. visit 3 (next in 2's adj list)
19. visit 4
20. return to 3
21. return to 2
22. return to 1
23. return to 0

**Drawing**

- green highlighted arrows are visiting new unvisited nodes
- non-highlighted arrows are returning from recursive subcall / removing edge from stack



## 2 Question 2.

There are two types of professional wrestlers: "good guys" and "bad guys". Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers, and we have a list of r pairs of wrestlers for which there are rivalries. Give an O(n + r)-time algorithm that determines whether it is possible to designate some of the wrestlers as "good guys" and the remainder as "bad guys" such that each rivalry is between a good guy and a bad guy. If it is possible to perform such a designation, your algorithm should produce it.

Wrestlers are vertices and rivalries are edges. Designating a wrestler one of two labels (good or bad guy) such that each edge (rivalry) is between nodes of different labels is equivalent to testing for bipartiteness. As shown in lecture, this is equivalent to finding an odd cycle, and can be done with a tree traversal where vertices are marked with opposite labels as their "parent" vertex in the traversal tree. If we mark a vertex and find a neighbor with that same label, then the graph is not bipartite.

```
1:  procedure LabelWrestlerAndTheirRivals(start: vertex to begin with, label: starting label)
2:      start.label ← label
3:      for neighbor in neighbors(start) do
4:          if label == neighbor.label then                    ▷ odd cycle detected, abort
5:              return false
6:          end if
7:          successful ← LabelWrestlerAndTheirRivals(neighbor, opposite(label))
8:          if not successful then
9:              return false
10:         end if
11:     end for
12:     return true
13: end procedure
14:
15: procedure LabelGoodOrBad(wrestlers: array of nodes)
16:     for wrestler in wrestlers do                           ▷ initialize all labels as None
17:         wrestler.label ← None
18:     end for
19:     for wrestler in wrestlers do
20:         if wrestler.label is None then
21:             successful ← LabelWrestlerAndTheirRivals(wrestler, good)
22:             if not successful then
23:                 return false
24:             end if
25:         end if
26:     end for
27:     return true
28: end procedure
```

# 3 Question 3.

Explain how both Prim's and Kruskal's Minimum Spanning Tree algorithm works on the graph below and print out the edges and its weight as you add it to your MST set. Assume that your source vertex is 0 and that your adjacency list is ordered in decreasing numerical values.