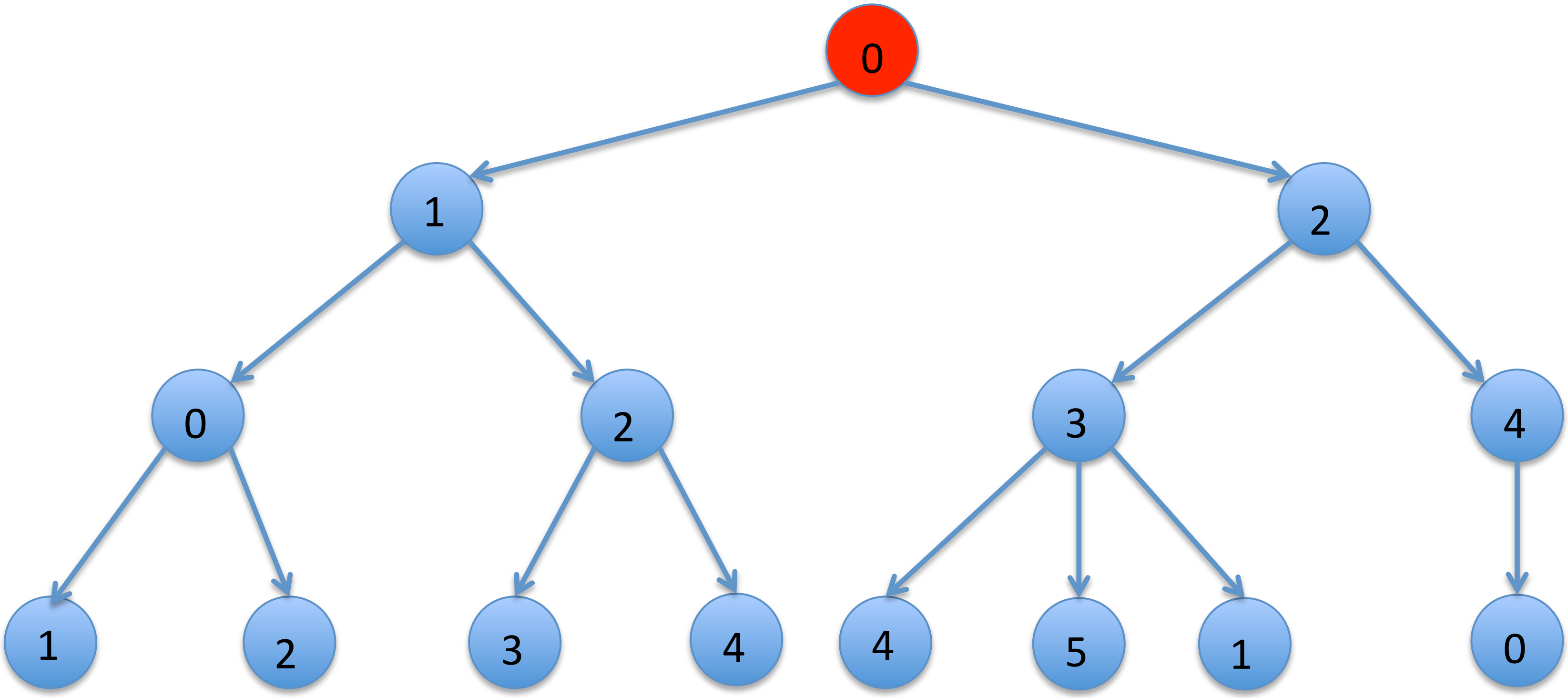
Depth first search

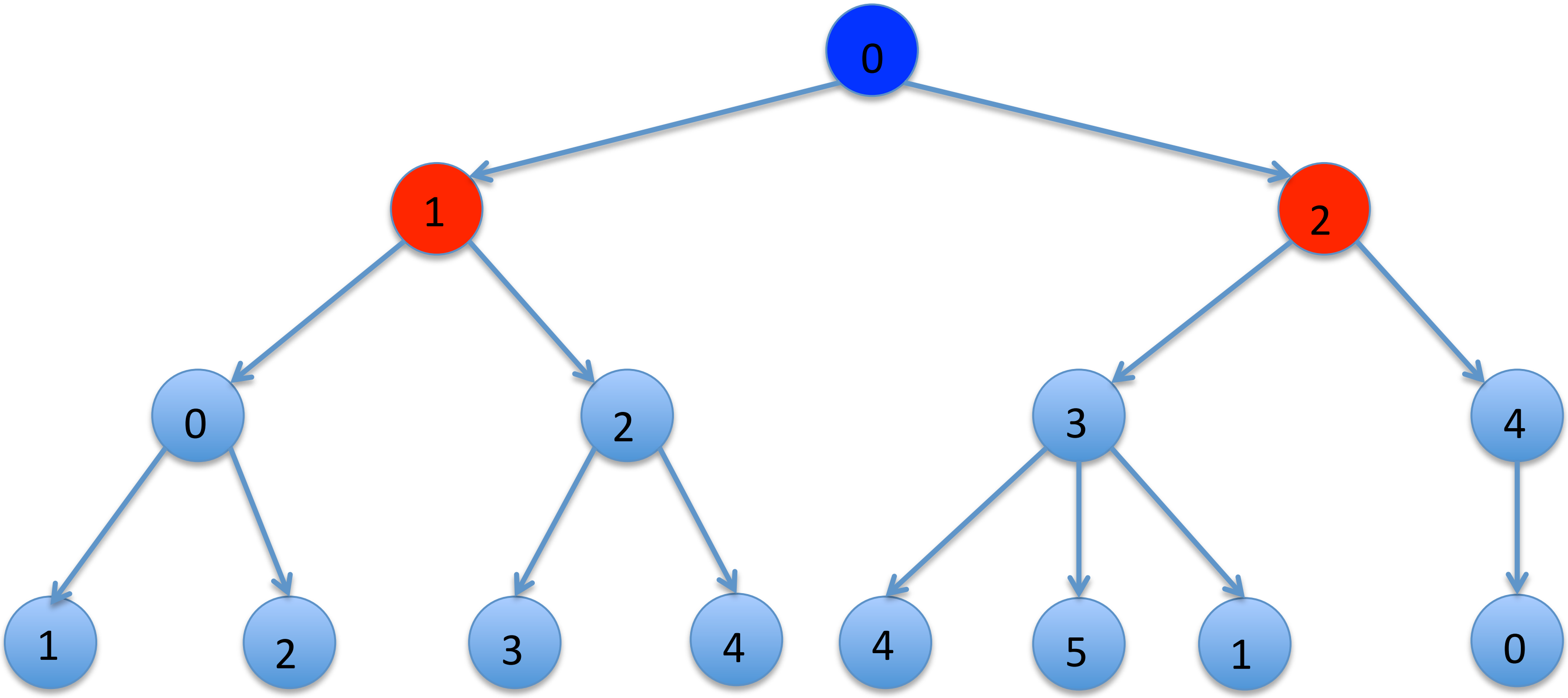
- Start at “root” node
  - Set of possible paths is just root node
- If not at “goal” node, then
  - Extend current path by adding each “child” of current node to path, unless child already in path
  - Add these new paths to potential set of paths, at front of set
  - Select next path and recursively repeat
  - If current node has no “children”, then just go to next option
- Stop when reach “goal” node, or when no more paths to explore

Better depth first search



0

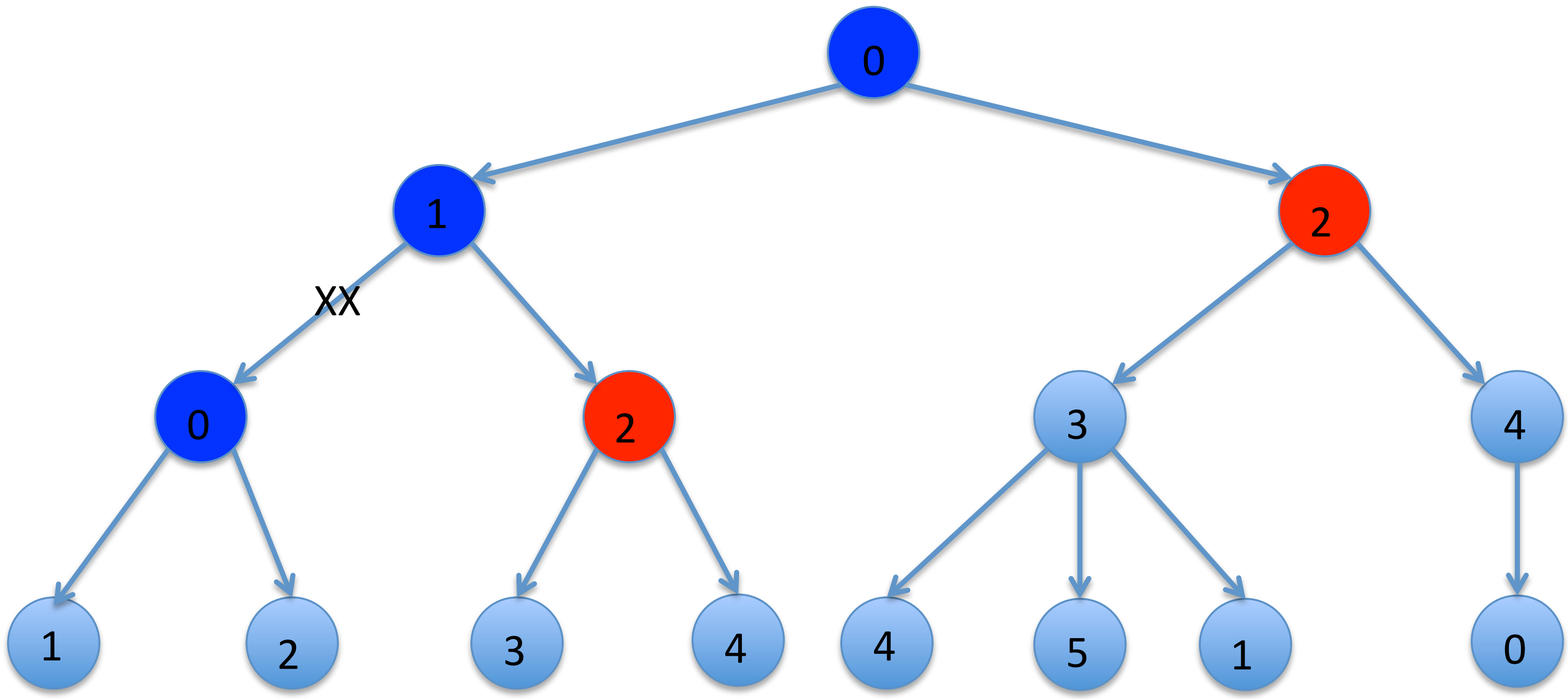
Better depth first search



0

01 02

Simple depth first search

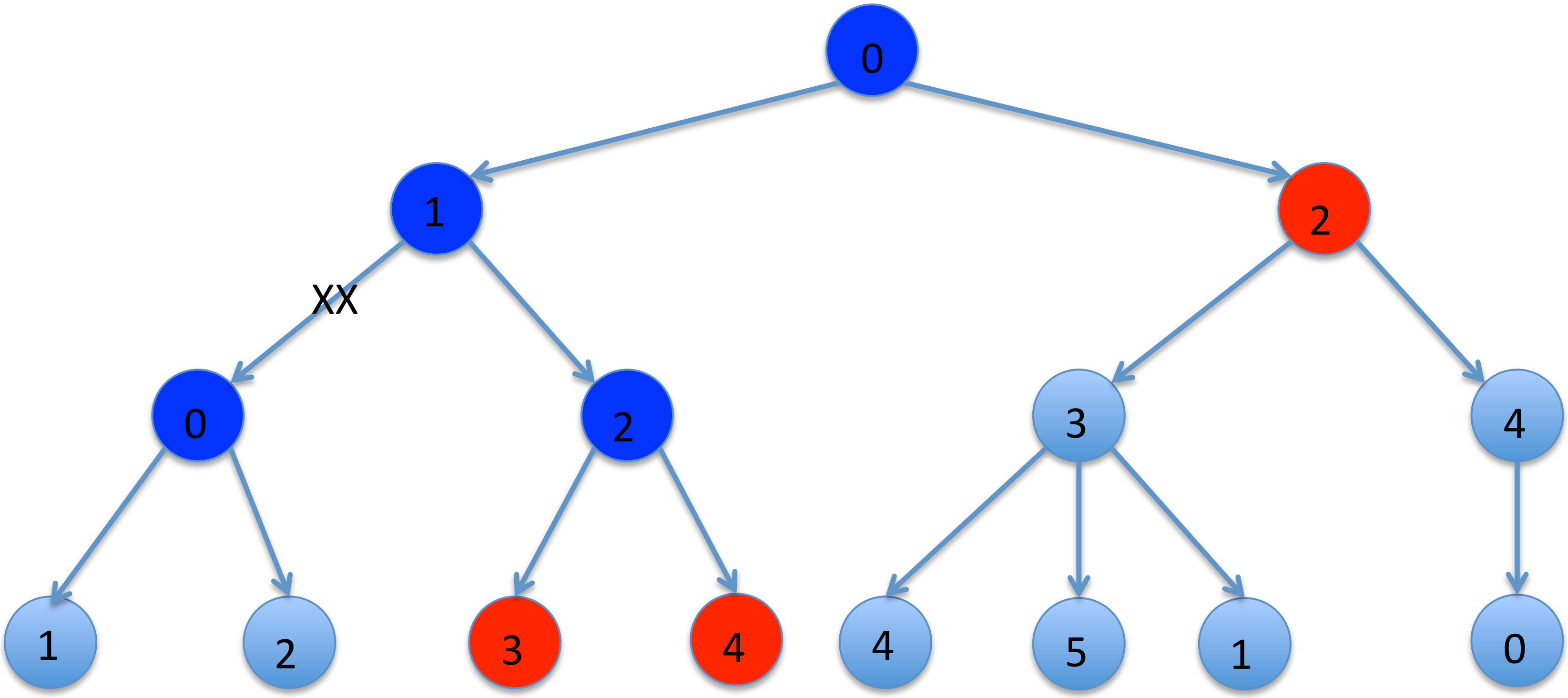


0

01 02

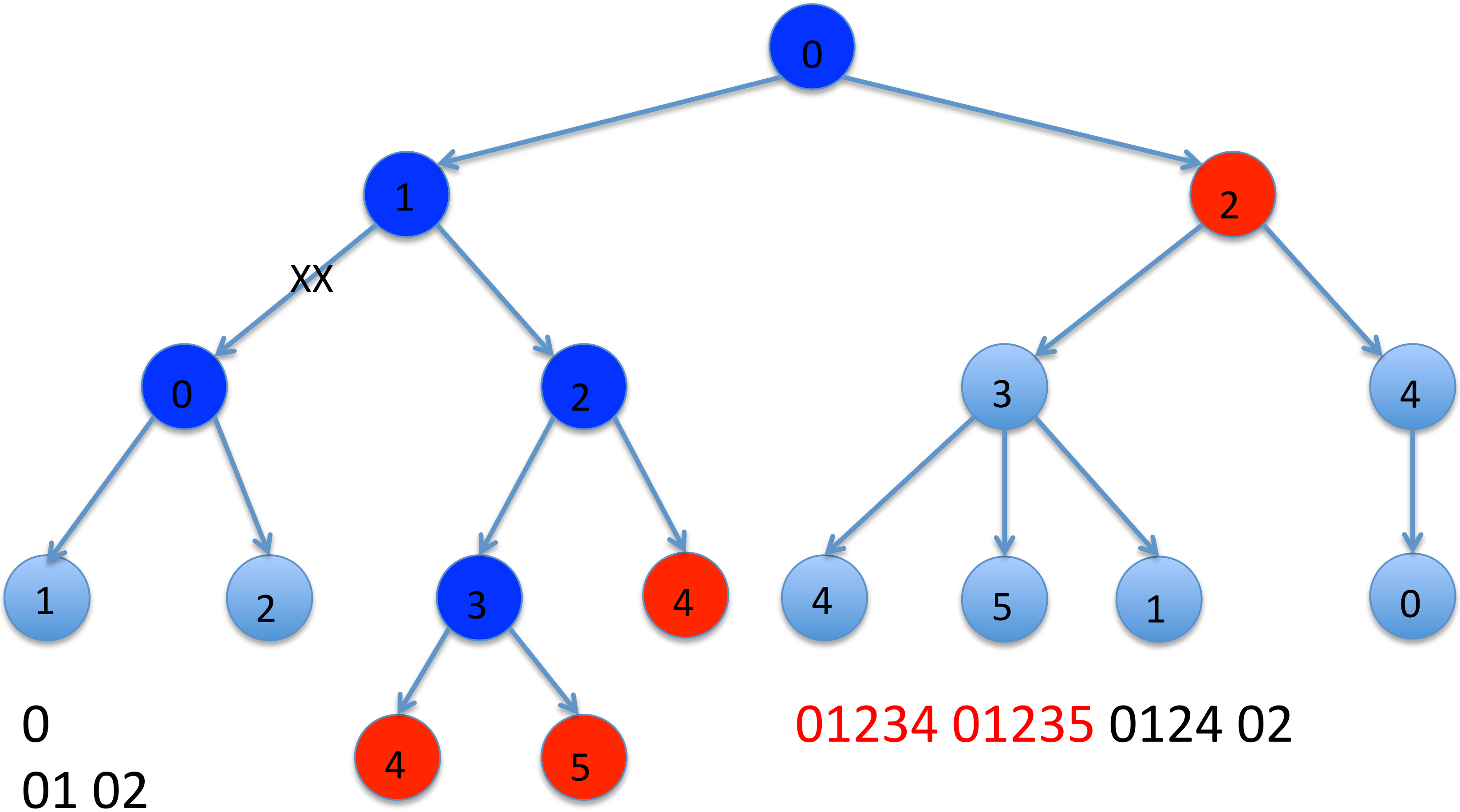
012 02

Simple depth first search



0
01 02
012 02
0123 0124 02

Simple depth first search



0

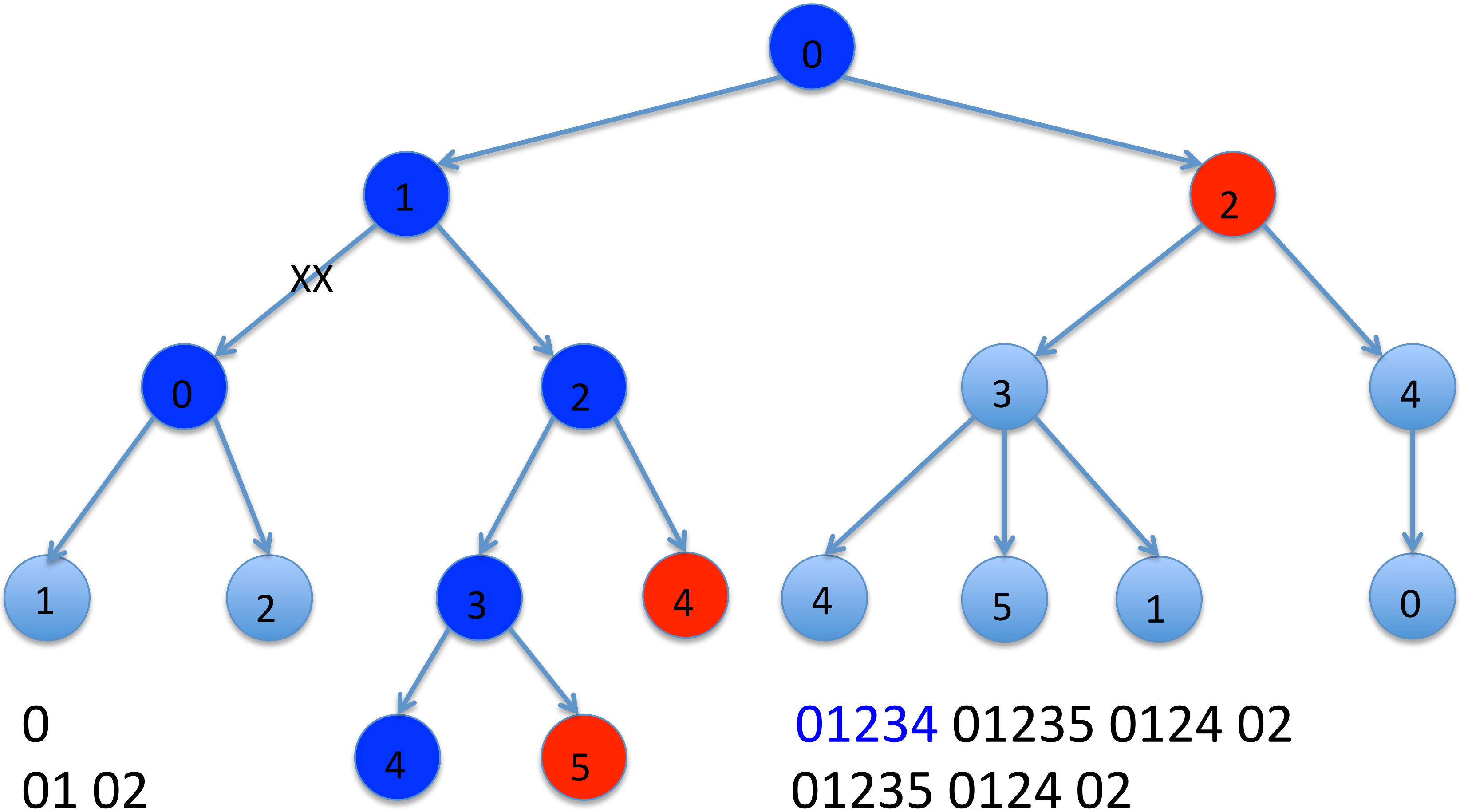
01 02

012 02

0123 0124 02

01234 01235 0124 02

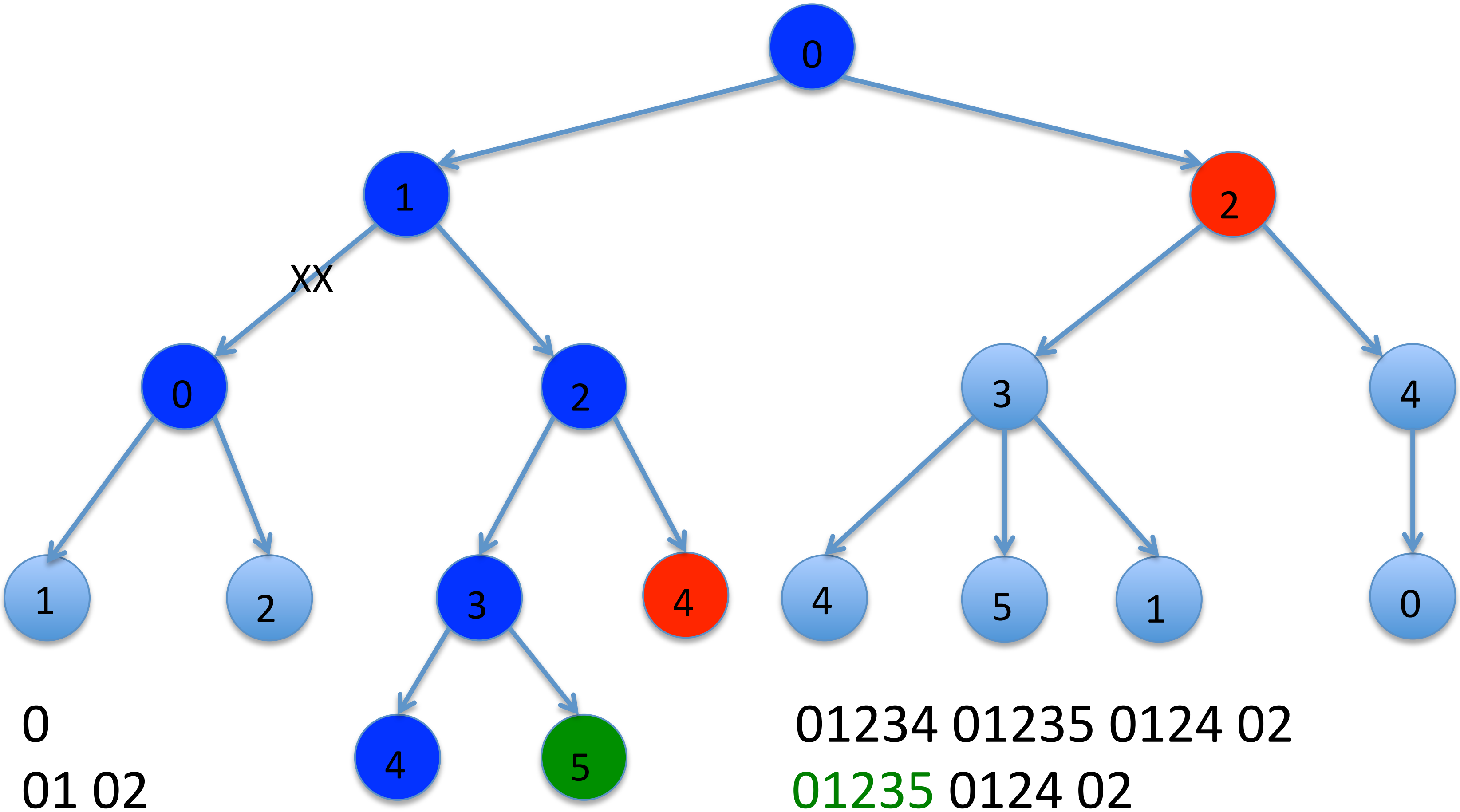
Simple depth first search



0
01 02
012 02
0123 0124 02

01234 01235 0124 02
01235 0124 02

Simple depth first search



0
01 02
012 02
0123 0124 02

01234 01235 0124 02
01235 0124 02

Some details

- Depth first search explores the first option in its set of possibilities
- It replaces the current first option by adding an ordered set of new options, extending the current path to each child node of the current node, and placing those options at the front of the set of possibilities (this is called a **stack**)
- It does not visit any child node already in the path
- As we have created the algorithm so far, it will stop once it finds a path

Sidebar: a stack

- A Stack is a data structure with a “last in, first out” behavior
  - We push items onto the top of the stack
  - We pop items off of the top of the stack
  - This maintains a set of items to be explored, where the top of the stack is the next item, and where new items are placed at top of the stack

Sidebar: a stack

- An example from our search
  - 0
  - 01 02 – pop 0 and push 01, 02
  - 012 02 – pop 01 and push 012
  - 0123 0124 02 – pop 012 and push 0123 and 0124
  - 01234 01235 0124 02 – pop 0123 and push 01234 and 01235

A simple DFS algorithm

```
def DFS(graph, start, end, path = []):
    # Assumes graph is a Digraph
    # Assumes start and end are nodes in graph
    path = path + [start]
    print 'Current dfs path:', printPath(path)
    if start == end:
        return path
    for node in graph.childrenOf(start):
        if node not in path: # Avoid cycles
            newPath = DFS(graph,node,end,path)
            if newPath != None:
                return newPath
    return None
```