Breadth first search

- Instead of going down the first branch of the tree, we could instead examine all children of a node first, before going deeper into tree
- In the simple case of no weights, we can stop as soon as we find a solution, since guaranteed to be shortest path

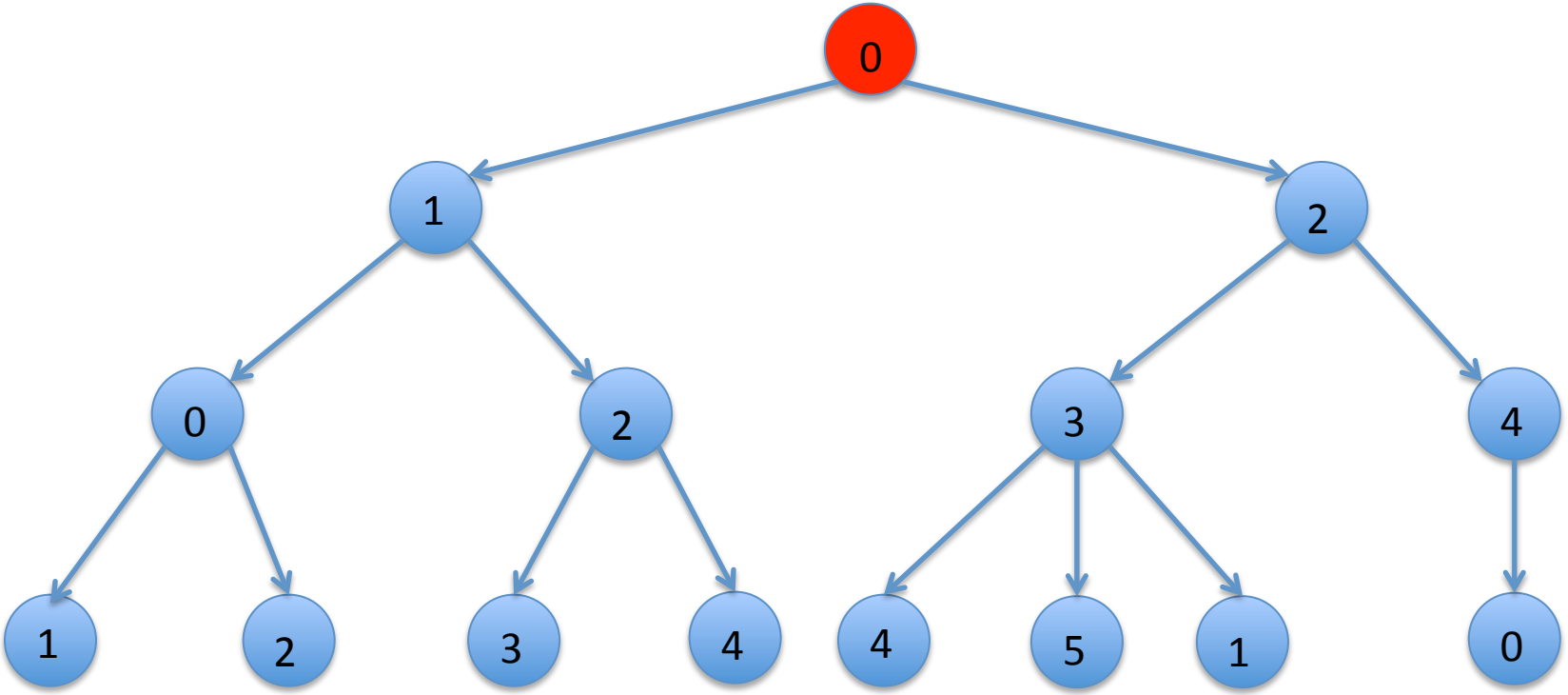
Breadth first search

- 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, but put at end of set (this uses a data structure called a **queue**)
  - 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

Sidebar: a queue

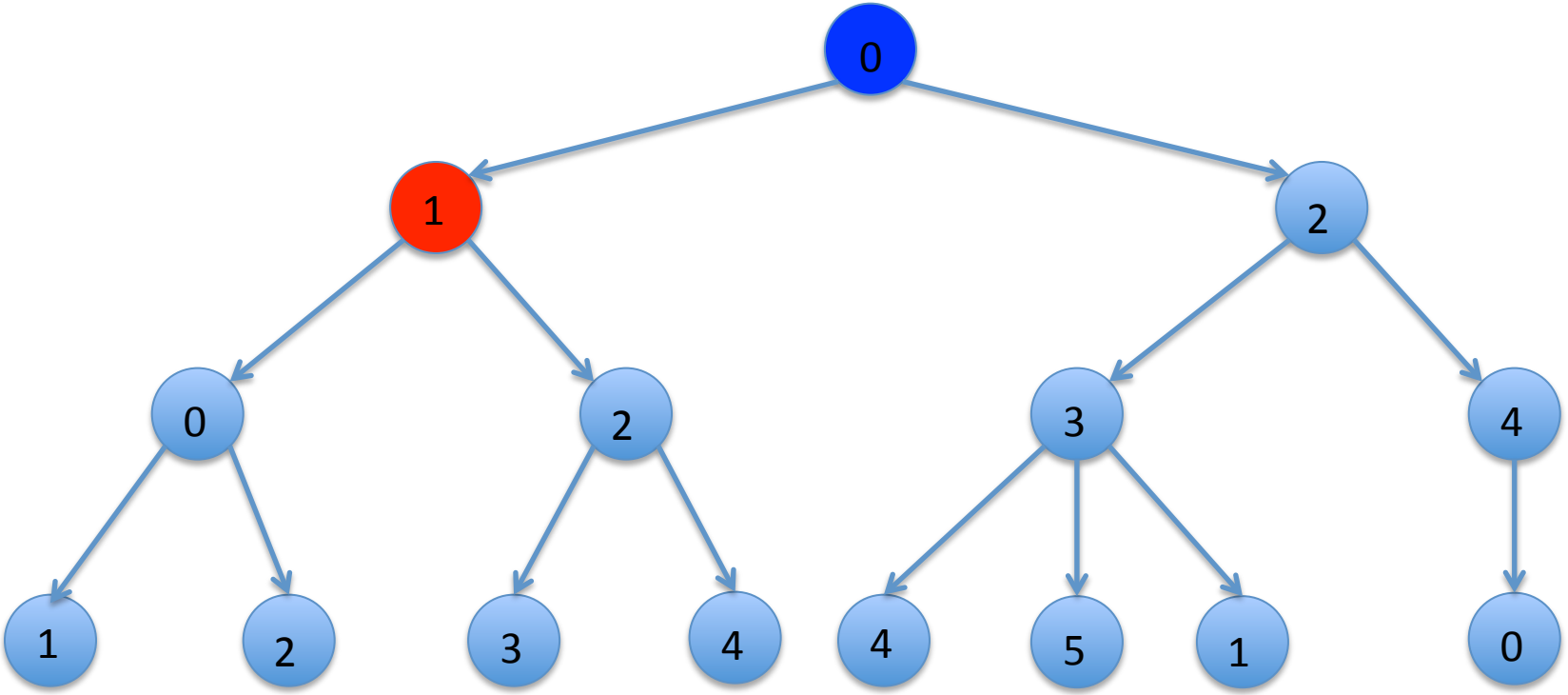
- A Queue is a data structure with a “first in, first out” behavior
  - We push items at the end of the queue
  - We pop items from the front of the queue
  - This maintains a set of items, where we explore each item in the order in which we create it

Breadth first search



0

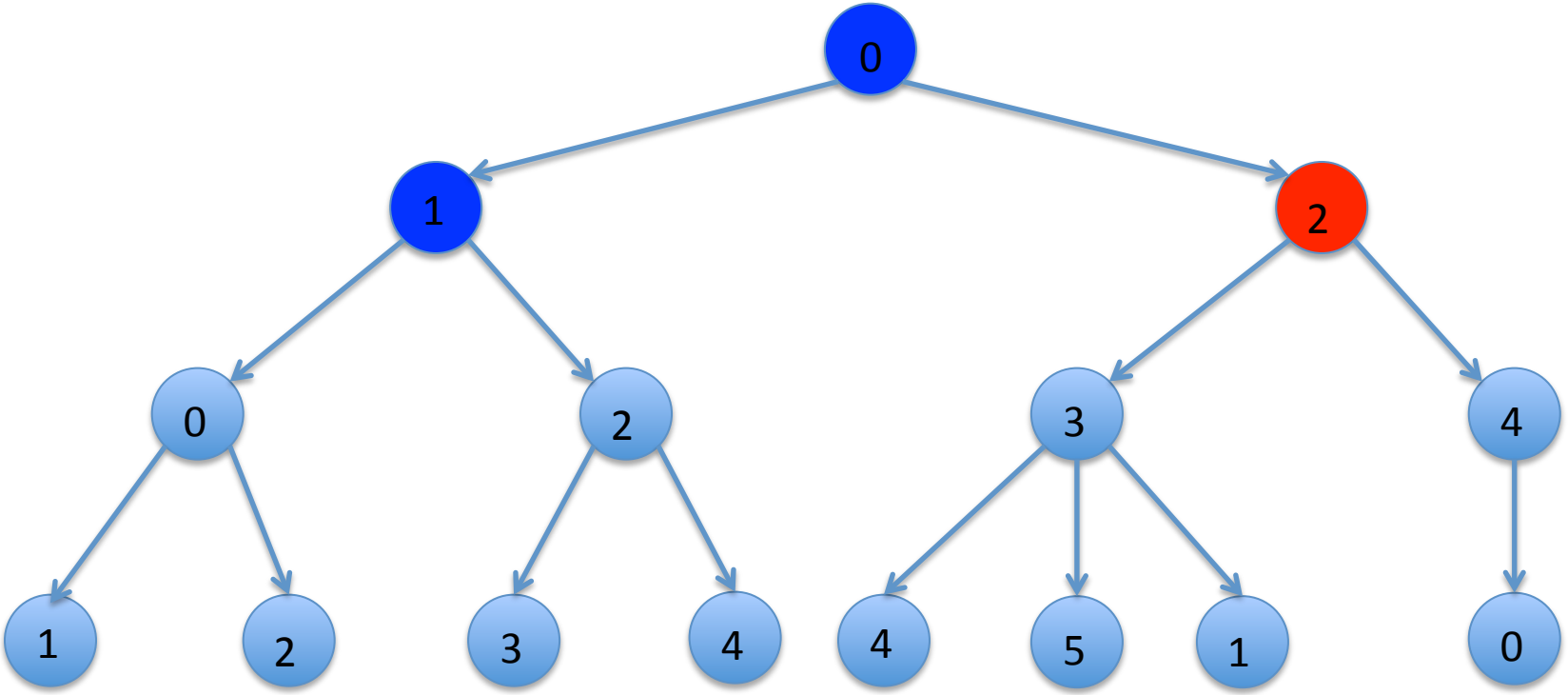
Breadth first search



0

01

Breadth first search

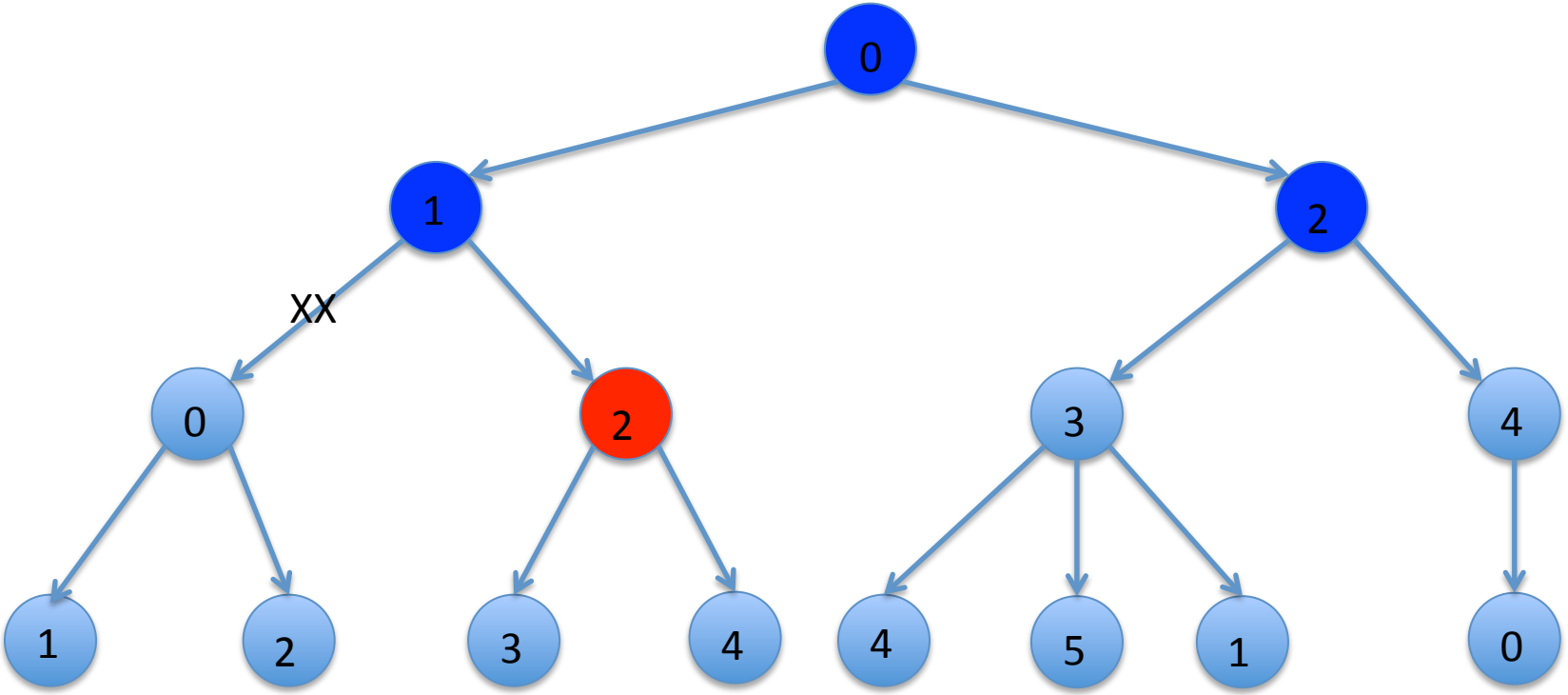


0

01

02

Breadth first search



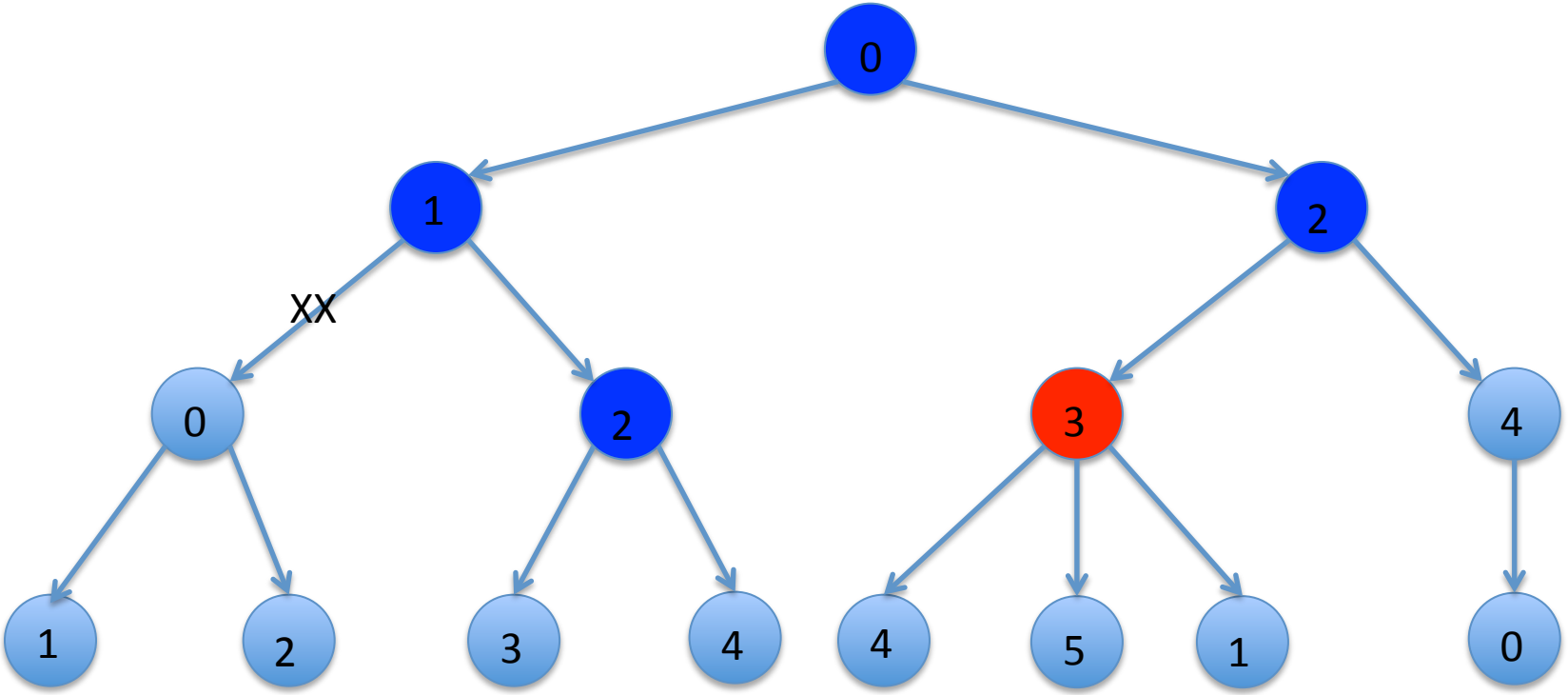
0

01

02

012

Breadth first search



0

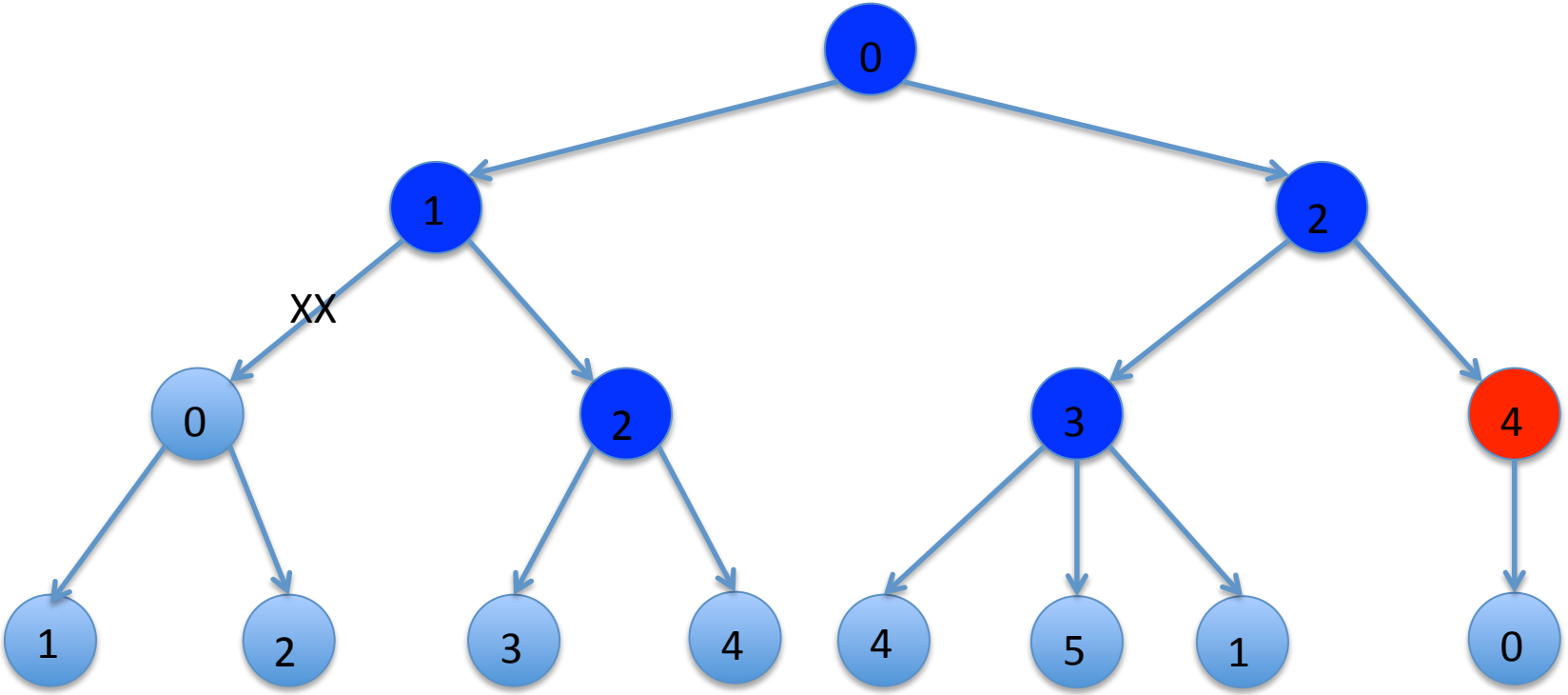
01

02

012

023

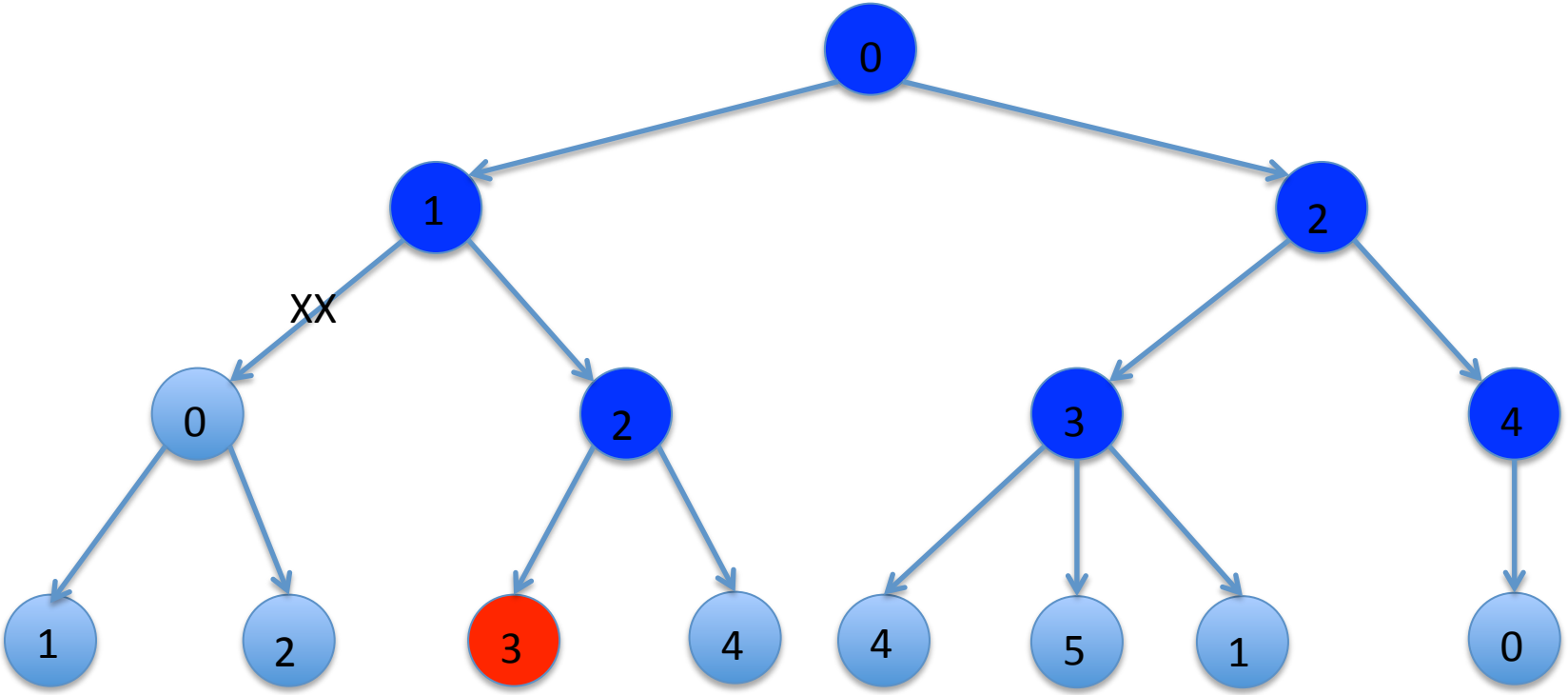
Breadth first search



0
01
02
012

023
024

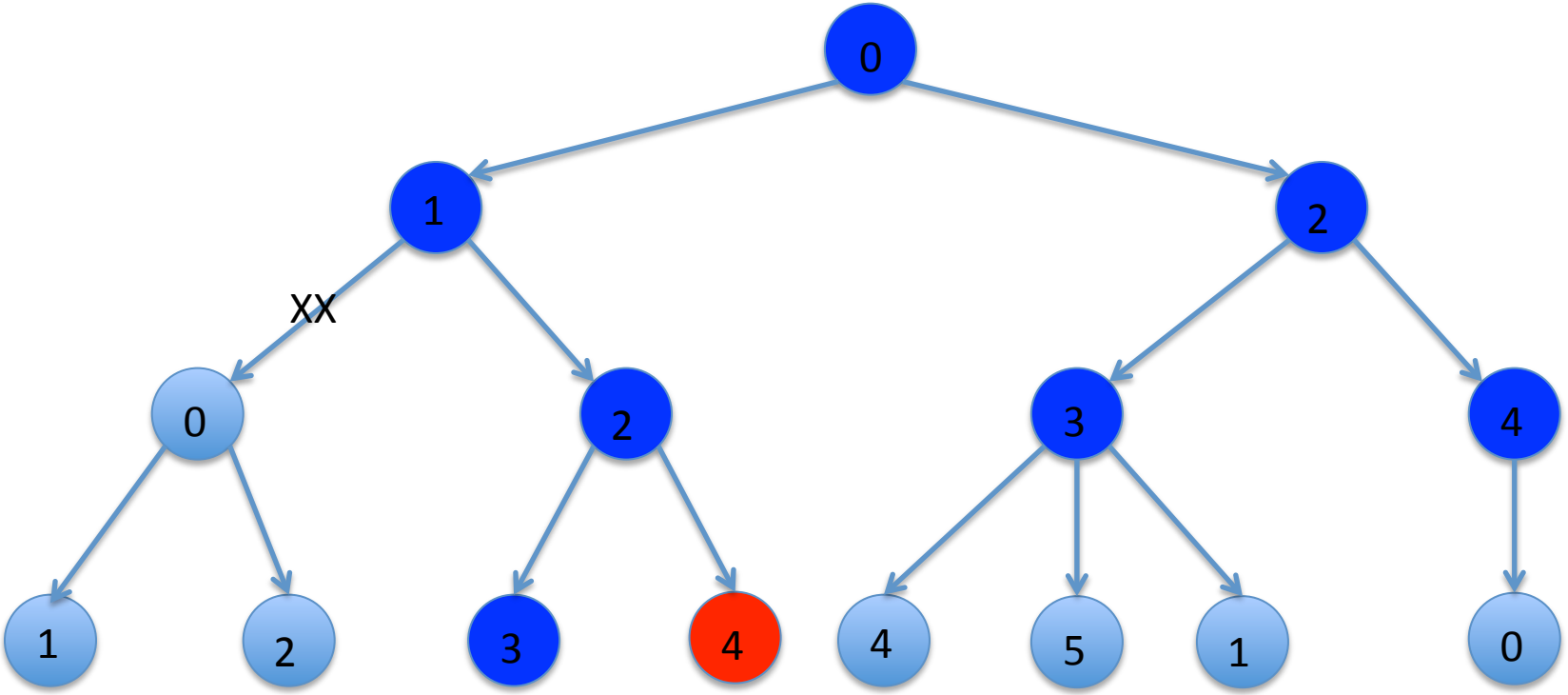
Breadth first search



0
01
02
012

023
024
0123

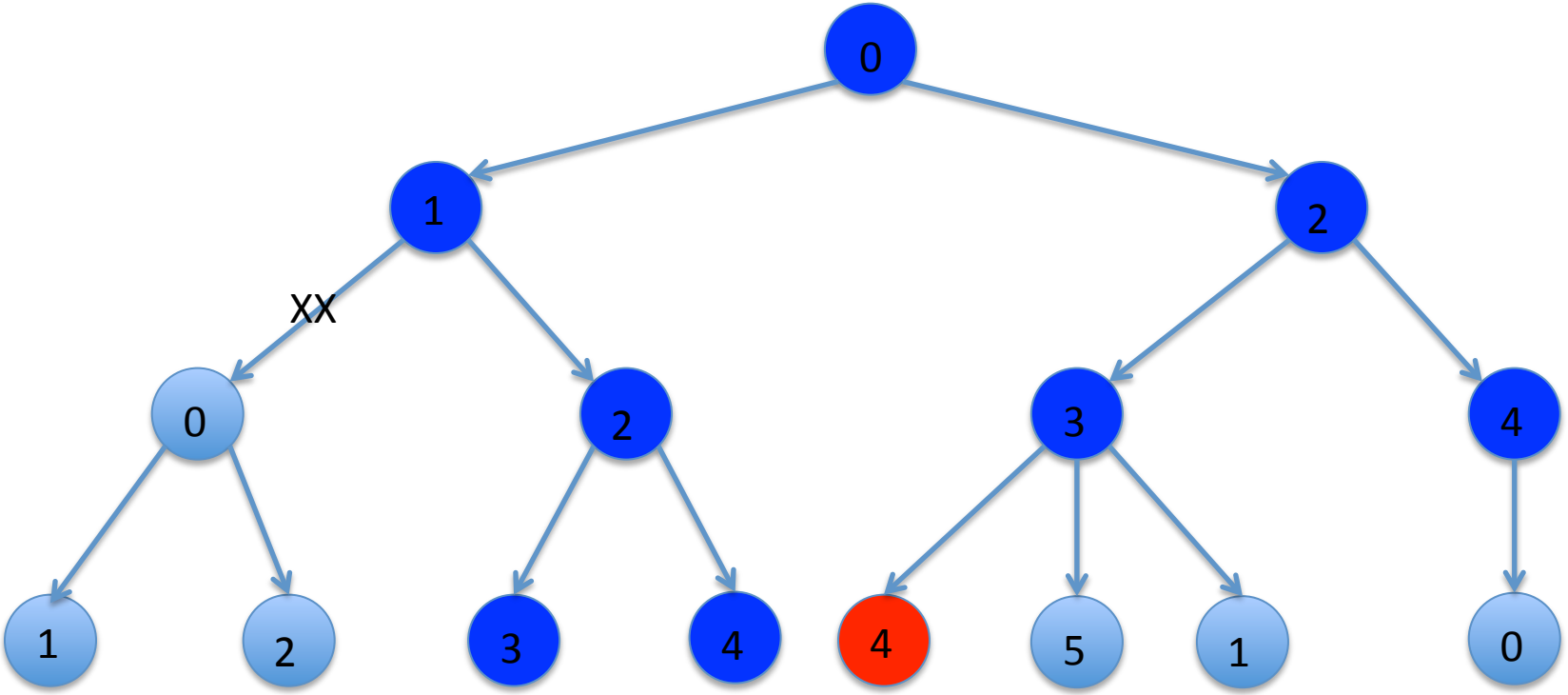
Breadth first search



0
01
02
012

023
024
0123
0124

Breadth first search

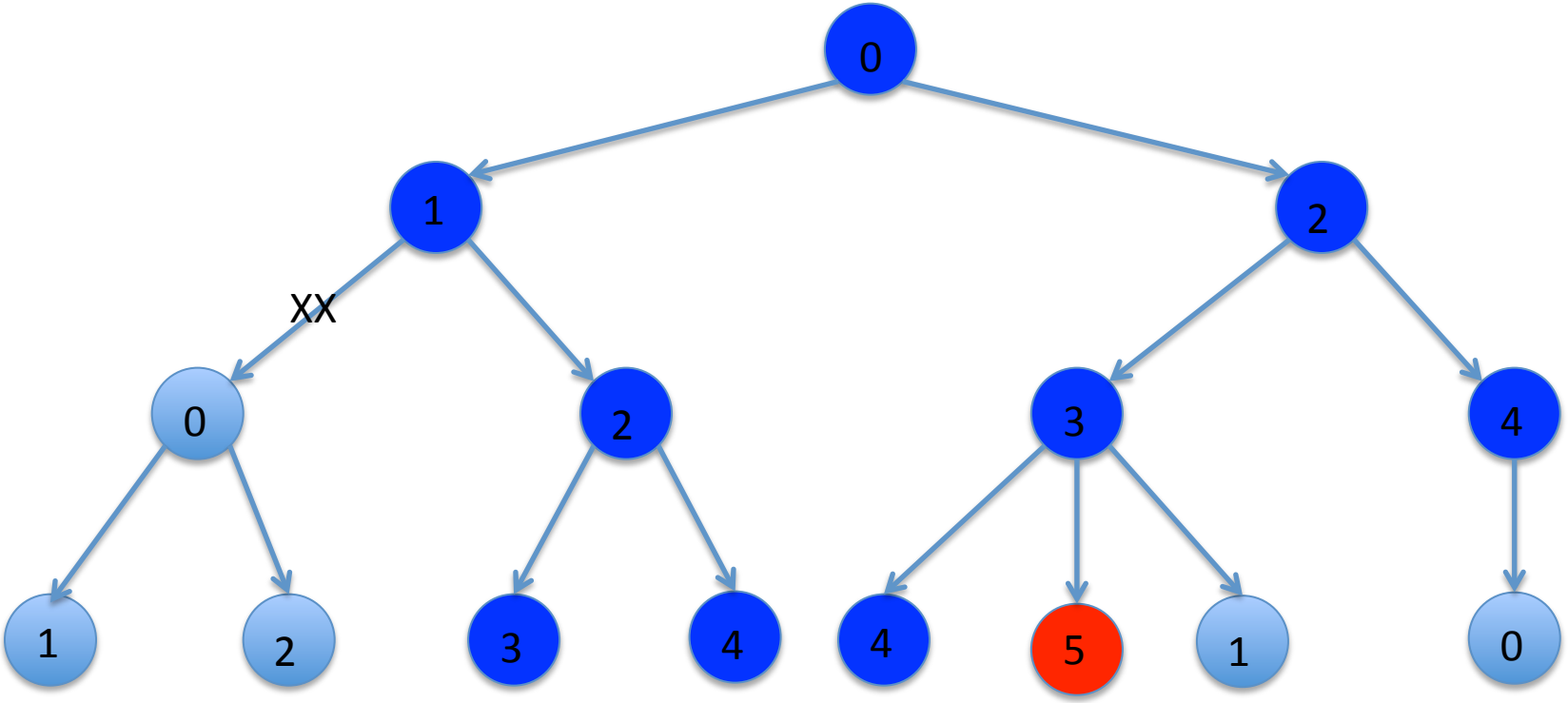


0
01
02
012

023
024
0123
0124

0234

Breadth first search



0
01
02
012

023
024
0123
0124

0234
0235

Sidebar: a queue

- An example from our search
  - 0
  - 01 02 – pop 0, insert 01, 02 at back of queue
  - 02 012 – pop 01, insert 012 at back of queue
  - 012 023 024 – pop 02, insert 023, 024
  - 023 024 0123 0124 – pop 012, insert 0123, 0124

Breadth first search algorithm

```
def BFS(graph, start, end, q = []):
    initPath = [start]
    q.append(initPath)
    while len(q) != 0:
        tmpPath = q.pop(0)
        lastNode = tmpPath[len(tmpPath) - 1]
        print 'Current dequeued path:', printPath(tmpPath)
        if lastNode == end:
            return tmpPath
        for linkNode in graph.childrenOf(lastNode):
            if linkNode not in tmpPath:
                newPath = tmpPath + [linkNode]
                q.append( newPath)
    return None
```