CSCI 204: Data Structures & Algorithms

Revised based on textbook author's notes.

Breadth First Search with Queues

Compared with Stacks

SOLUTION WITH STACK

Backtracking

- Here is an example HPAir of using stack for backtracking
  - Given
    - A set of cities that HPAir serves
    - Pairs of city names, each pair represents the origin and destination of one flight
    - Pair of cities names each of which represents a request to fly from an origin to a destination
  - Find whether or not a path exists between two cities requested by a passenger

General ideas

1. Start from the origin city
2. Find a city that is connected to the current city
3. If the next city is the destination, we are done. Report a route has been found
4. Otherwise
   1. If we found next city, push it onto stack
   2. If we found none, just pop the stack
5. If we visited all cities and no route is found, we declare the failure

When we pursue a path, we explore a path as deep as possible. This type of solution is called "depth-first search" or **DFS**.

Source code

```
def dfs_paths(graph, start, goal):
    """Depth first search using a stack"""
    visited = unvisit_all()
    stack = Stack()

    stack.push((start, [start]))    # push both the city and the path to stack
    mark_visited(visited, start)

    while stack.is_empty() == False:
        (vertex, path) = stack.pop()
        for next in graph[vertex]:
            if visited[next] == False:
                mark_visited(visited, next)
                if next == goal:
                    return path + [next]
                else:
                    stack.push((next, path + [next]))
```

Note: both the city (vertex) and the path (as a list) are on stack!

Try out dfs.py

SOLUTION WITH QUEUE

General ideas

1. Start from the origin city
2. While the queue is not empty
3. For each connected city,
   1. If next is the destination, return success
   2. If not, add next and current path to queue
4. Continue in Step 2
5. If we visited all cities and no route is found, we declare the failure

When we pursue a path, we explore a path as wide as possible. This type of solution is called "breadth-first search" or **BFS**.

Source code

```
def bfs_paths(graph, start, goal):
    """Breadth first search using a queue"""

    visited = unvisit_all()

    queue = Queue()
    queue.enqueue((start, [start]))     # put both the city and path on queue
    mark_visited(visited, start)

    while queue.is_empty() == False:
        (vertex, path) = queue.dequeue()
        for next in graph[vertex]:      # try each city that is connected
            if visited[next] == False:
                mark_visited(visited, next)
                if next == goal:
                    return path + [next]
                else:
                    queue.enqueue((next, path + [next]))
```

Note: both the city (vertex) and the path are on queue!

Try out bfs.py