Graph traversals

Depth-first search

DFS(*vertex*):
  precondition: *vertex* is coloured white; if a vertex is coloured black then it is reachable from *vertex*; an edge between black vertices is either black or grey; an edge between white vertices is white
  postcondition: if a vertex is reachable from *vertex*, then it is coloured black, otherwise is it coloured white; an edge between black vertices is either black or grey; an edge between white vertices is white
  input: source vertex of an undirected simple graph

```
colour vertex vertex gray
for each edge incident on vertex do
    if edge is white then
        other ← other endpoint of edge
        if vertex other is white then
            colour edge black
            DFS(other)
        else
            colour edge gray
colour vertex black
```

Breadth-first search

BFS(*vertex*):
  precondition: ...
  postcondition: ...
  input: source vertex

```
colour vertex gray
queue ← empty queue
enqueue vertex in queue
while queue is nonempty do
    node ← front of queue
    for each edge incident on node do
        if edge is white then
            other ← other endpoint of edge
            if vertex other is white then
                colour edge black
                colour other grey
                enqueue vertex other in queue
            else
                colour edge gray
    dequeue vertex node from queue
    colour vertex node black
```

Cycle detection

hasCycles(*vertices*, *edges*):
  input: undirected simple graph
  output: graph (*vertices*, *edges*) has cycles?

```
for each vertex in vertices do
    colour vertex white
for each edge in edges do
    colour edge white
for each vertex vertex do
    if vertex is white then
```

DFS(*vertex*)
return hasGreyEdge(*edges*)

hasGreyEdge(*edges*):
input: edges of a undirected simple graph
output: *edges* contains a grey edge?
found ← false
for each *edge* in *edges* **do**
    found ← *found* or (*edge* is grey?)
return *found*

Shortest path

shortestpath(*first*, *second*):
input: two vertices of the graph
output: a shortest path between *first* and *second*
BFS'(*first*)
sequence ← empty sequence
node ← *second*
add *node* to end of *sequence*
while *node* ≠ *first* **do**
    add edge between *node* and parent of *node* to beginning of *sequence*
    add parent of *node* to beginning of *sequence*
    node ← parent of *node*
return *sequence*

BFS'(*vertex*):
precondition: ...
postcondition: ...
input: source vertex
colour *vertex* gray
queue ← empty queue
enqueue *vertex* in *queue*
while *queue* is nonempty **do**
    node ← front of *queue*
    for each *edge* incident on *node* **do**
        if *edge* is white **then**
            other ← other endpoint of *edge*
            if vertex *other* is white **then**
                colour *edge* black
                colour *other* grey
                set *other*'s parent to be *node*
                enqueue vertex *other* in *queue*
            else
                colour *edge* gray
    dequeue vertex *node* from *queue*
    colour vertex *node* black