CS119 – Activity 10

Consider the abstract data types Stack and Queue:

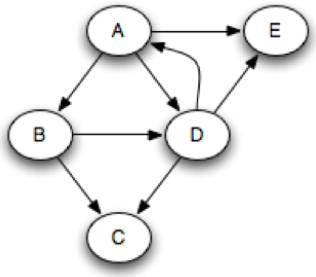
Function	Explanation
`empty :: Stack a`	gives an empty stack
`isEmpty :: Stack a -> Bool`	determines if a stack is empty
`top :: Stack a -> a`	gives the item that is at the top of the stack
`push :: a -> Stack a -> Stack a`	adds an item to the top of the stack
`pop :: Stack a -> Stack a`	removes an item from the top of the stack

Function	Explanation
`empty :: Queue a`	gives an empty queue
`isEmpty :: Queue a -> Bool`	determines if a queue is empty
`front :: Queue a -> a`	gives the item that is at the front of the queue
`enqueue :: a -> Queue a -> Queue a`	adds an item to the rear of the queue
`dequeue :: Queue a -> Queue a`	removes an item from the front of the queue

For each of the tasks below, determine whether we would want to use a stack, queue, or either.

1. A language runtime system that handles recursive function calls
2. Undo system in a text editor
3. A system for handling waiting printer jobs for a shared printer
4. Page visited history in a web browser for the back button
5. Operating system handling multiple processes that need execution
6. Visiting all the pages on a website via the hyperlinks

Let's examine visiting the pages on a website. Suppose we have a website with five pages A, B, C, D, and E. The diagram below indicates the hyperlinks on the pages. So page A has hyperlinks to B, D, and E.



Consider the following algorithm:

```
visitPages :: Website a -> [a]
visitPages website = (startPage website) :
                     visit (addHyperlinks (startPage website) empty) where
        visit q = if (isEmpty q) then
                        []
                  else
                        (front q) : visit (addHyperlinks (front q) (dequeue q))

addHyperlinks page q = <code which enqueues the unvisited pages reached
                                    from this page>
```

Complete the trace of this algorithm with the website above with start page A. (I am using `[[ ]]` to illustrate the queue ADT in the trace):

`A : (visit [[B,D,E]])` – `visitPages` on the website creates a list starting with the starting page and the rest of the list is created by the call to `visit` on the hyperlinks

`A : B : (visit [[D,E,C]])` – B is removed from the queue and the unvisited hyperlinks from B are added to the queue in the recursive call

`A : B : D: (visit_____)`

`A : B : D: ____ (visit_____)`

`[A, B, D, ____ ,____]`

Change the algorithm so that instead of using a Queue it uses a Stack and trace through it again, giving the list which would be produced.