Example Applications for BFS and DFS

Algorithms and Data Structures

Transitive Closure

A set of software packages are listed in a text file together with their dependencies. The format of the file is as follows:

```
pkg1 pkg2 pkg3
pkg3 pkg2
pkg2 pkg4
pkg5 pkg2 pkg4
pkg6
```

This says that *pkg1* depends on *pkg2* and *pkg3*, that *pkg3* depends on *pkg2*, which depends on *pkg4*, etc.

Write a function `all_dependencies (p)` that, once the dependency file has been correctly read, given a package `p` returns an array containing all and only those packages upon which `p` depends, either directly or indirectly.

You my use global variables to store the input graph. Or you may also pass the graph to the `all_dependencies` function through additional parameters.

Connected Components in an Undirected Graph

A *connected component* of an undirected graph G is a maximal subgraph H of G such that all nodes in H are reachable from each other. Write a function `count_connected_components(f)` that reads an undirected graph G from an input file `f` and returns the number of strongly connected components in G. The format of the input is the same as in the *Transitive Closure* example. However, notice that the input graph is *undirected.* Therefore, an input that contains a single line `A B` indicates an undirected edge between vertexes `A` and `B`.

Dependency Cycles

Write a function `contains_a_cycle()` that, given a dependency graph like the one described in the *Transitive Closure* exercise, returns `True` if and only if the input graph contains a cycle.

Topological Sort

Write a function `topological_sort()` that, given a dependency graph like the one described in the *Transitive Closure* exercise, returns an array of the nodes sorted in *topological* order. A topological order is such that, if u precedes v, then u must not depend on v, either directly or indirectly. If it is impossible to obtain a valid topological order of all the nodes, then the output must be `None`.