Extension Material:
Graph algorithms based on traversal

1 Topological sorting

Topological Sorting is a linear sequence of all vertices in a Directed Acyclic Graph (DAG). And the sequence must meet the following two conditions:

- Each node appears exactly once.
- If there is an edge from node s to node t, then s is before t in the list.

How do we construct a topological sorting?

1.1 One solution

1. Save the indegree of each vertex in an array.
2. Find all nodes with zero incoming edges and insert them into a set. Remove one from the set and store it in the final sorted sequence, and then delete the node from the graph (i.e. all edges associated with it are deleted, and indegrees of adjacent vertices are reduced by 1).
3. Repeat the step 2 until set is empty.

1.2 Another method based on dfs

1. Call dfs function on each vertex.
2. After each vertex is finished, the time is recorded. Insert it to the front of a linked list.
3. return the linked list of vertices.

Why do we insert the node to the front of list?

From the nature of topological sorting we know that if there is an edge from node s to node t, then s is before t in the list.

We also know that DFS uses stack, which is last-in, first-out. When the vertex s is visited first in traversal, it will be inserted into the final sequence later than node t. Therefore, if we want to acquire the topology sequence in order, we should add the current vertex to the front of the list.

There are other ways to compute topological sorting, click here for more information.

2 Detecting cycles in a directed graph

Note that the above topological sorting only works with an acyclic graph. Then we could simply apply the same algorithm.

Another method is to use dfs. When an already visited node is encountered in traversal, a cycle is detected.

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge
while S is non–empty do
    remove a node n from S
    add n to tail of L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S
if graph has edges then
    return error   (graph has at least one cycle)
else
    return L   (a topologically sorted order)
```

Figure 1: Topological sort

3 Testing bipartite graph

For finite undirected graphs, a 2-colouring of a graph is an allocation of one of two colours (say, red and green) to each of the nodes of the graph, so that, whenever two nodes are linked by an edge, they are allocated different colours. Some graphs have a 2-colouring and some do not, e.g. a triangle.

How can a DFS algorithm be used to determine whether or not a finite undirected graph has a 2-colouring?

The main idea is to colour parent nodes alternate colour from child nodes in a DFS. When a coloured node is encountered: if it is same colour as parent - the graph is not 2-colourable - if it is different - continue colouring.

For more information about bipartite graph, click here.

4 Finding connected components of a graph

A connected component (sometimes, just a 'component') of a graph G, is a largest connected subgraph (i.e. one that cannot be expanded with additional nodes without becoming disconnected).

The idea to compute the connected components of a graph is simple. Using either DFS or BFS which begins at any vertex v will find the entire connected component containing v.

To find all the connected components of a graph, loop through its vertices, starting a new breadth first or depth first search whenever the loop reaches a vertex that has not already been included in a previously found connected component.

5 Eulerian cycle and Hamiltonian cycle problem

5.1 Seven Bridges of Konigsberg

The Seven Bridges of Konigsberg is a famous problem in graph theory. The problem is based on a real-life example:

In the 17th century there is a city of Konigsberg in Prussia (now Kaliningrad, Russia) city setting on both sides of the Pregel River, with two small islands in the center. The islands are connected to each other and to both sides of river by seven bridges. The problem was to devise a walk through the city that would cross each of those bridges exactly once.

In 1736, Euler published a paper (which becomes the first important document in the history of graph theory) that proved it is impossible to give a solution to the Seven Bridges of Konigsberg problem. He abstracted it into a mathematical problem, that is, is there a cycle in graph that pass through each edge exactly once. Later, people refer to a graph with such a loop as an Eulerian graph.

5.2 Eulerian cycle

An Eulerian path is a trail in a finite graph which visits every **edge** exactly once. Similarly, an Eulerian cycle is an Eulerian path which starts and ends on the same vertex.

5.3 Hamiltonian path problem

The general idea of Hamiltonian path problem is that in any given graph, can a path be found that goes through each **vertex** without repetition (not necessarily through each **edge**) and finally returns to the original starting vertex.