# Module 0112: Introduction to graphs

Tak Auyeung, Ph.D.

December 13, 2006

## 1 About this module

- Prerequisites:
- Objectives: This module discusses the implementation of graphs as a data structure.

## 2 Graphs, in general

A graph consists of vertices and edges among the vertices. Let us use $V$ represent a set of vertices. An edge is a tuple $(n, m)$, in which both $n$ and $m$ are elements of $V$. Note that edges cannot be represented as a function because a vertex corresponds to many other vertices. The set of all edge tuples is represented by $V$. Thus, a graph can be represented as $G = (V, E)$. It only makes sense that $\forall(x, y) \in E : x, y \in V$. In other words, $E$ cannot contain an edge such that at least one of its vertices is not in $V$.

In terms of representation, each vertex is associated with any number of other vertices. Consequently each vertex has a *container* associated with it to represent vertices that are reachable directly from the vertex.

In this case, a container can be a linked list, an array or a tree. However, it is usually best to use a container that is efficient at lookup operations. A sorted array or a tree (AVL tree) are efficient containers.

## 3 Terms related to a graph

### 3.1 Path

A path is tuple of edges. A path $(e_1, e_2, e_3, \ldots e_n)$ has the special property that $\forall i \in [1 \ldots n-1] : e_i = (v_i, v_{i+1}), e_{i+1} = (v_{i+1}, v_{i+2}) \in E$. In English, a path is a tuple of edges so that the destination vertex of one edge is the source vertex of the next edge in the tuple.

Note that there may exist multiple paths between two vertices.

### 3.2 Cycle

A cycle is a path such that the begin vertex is also the destination vertex of the path. In fact, in a cycle, there is no begin and there is no end.

### 3.3 Degree of vertex

The degree of a vertex is the number of edges sourcing from a vertex. If we use $d(v)$ denote the degree of vertex $v$. We can define it as $d(v) = |\{(v, w)|(v, w) \in E\}|$.

### 3.4 (Un)Directed edges

In some graphs, edges are directed. In other graphs, edges are undirected. Whether edges are directed all depend on how a graph is used, and what it represents.

It is possible, however, to represent all graphs using directed edges. If a graph does not require directed edges, then we can used edge pairs. In other words, if vertex $v$ and $w$ are connected, and there is no need to indicate direction, then we can use $(v, w)$ and $(w, v)$ to represent the connection.

### 3.5 Subgraph

Given that we start with a graph $G = (V, E)$, a subgraph $G' = (V', E')$ can be defined such that $V' \subseteq V$ and $E' \subseteq E$.

### 3.6 Connectivity

Two vertices $v$ and $w$ are connected if there exists at least one path between $v$ and $w$.

## 4 Graph traversal

### 4.1 Depth first

The pseudocode to perform depth first search is in listing 1.

Listing 1: depth first search

```
define sub dfs
  by reference G = (V, E) : graph
  by reference Visited : set of Vertices
  by value x : vertex
  if (not (x in Visited)) then
    add x to Visited
    for each (x, y) in E do
      invoke dfs G <-> G, Visited <-> Visited, y <-> x
    end for
  end if
end define sub
```

No stack is explicitly used because the subroutine is recursive. The set `Visited` is necessary so we know when to stop in the case of cycles. The initial call should supply an empty set for `Visited` and a start vertex for `x`.

### 4.2 Breadth first

The pseudocode to perform breadth first search is in listing 2.

Listing 2: breadth first search

```
define sub bfs
  by reference G = (V, E) : graph
  by value x : vertex
  local Visited : set of vertices
  local Boundary : set of vertices
  Visited = {}
  Boundary = { x }
  NextBoundary = { }
  while (not(Boundary = {})) do
    while (not (Boundary = {}) do
      select y from Boundary
      Boundary <- Boundary - y
      Visited <- Visited + y
      if not(y in Visited) then
        for each (y, z) in E do
          NewBoundary <- NewBoundary + z
        end for
      end if
    end while
    Boundary <- NextBoundary
    NextBoundary <- {}
  end while
end define sub
```

### 4.3 Applications

Depth-first-search (DFS) is useful when only one “front” can be maintained. For example, when a robot explores an environment (such as an office environment), DFS makes sense because the robot can only be at one place at any particular time.

Breadth-first-search (BFS) is useful when we want to find a least cost path from a vertex to another vertex.