**Introduction to Graphs**

**Graphs**

**Table of contents**

1. [Introduction](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#intro)
2. [Graph terminology](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#terminology)
   * [Directed and undirected graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#directed)
     + [Directed acyclic graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dag)
   * [Weighted and unweighted graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#weighted)
   * [Other graph terms](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#otherTerms)
3. [Data structures and representations](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dataStruct)
   * [Adjacency Matrix](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#adjMatrix)
   * [Adjacency List](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#adjList)
   * [Comparison of the representations](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dsCompare)
4. [Tree Graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#treeGraphs)
5. [Graph traversal](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#traversal)
   * [Breadth-first traversal](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#bfs)
   * [Depth-first traversal](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dfs)
6. [Some questions to ask of a graph](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#graphQuestions)
7. [Discussion](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#discussion)

**Introduction**

A graph is a set of nodes or vertices, with a set of edges that connect pairs of distinct nodes (with at most one edge connecting a pair of nodes). Formally, graph G is a pair (V, E), where V is a finite set (set of vertices) and E is a finite set of pairs from V (set of edges). We will often denote n = |V|, m = |E|.

Graphs are everywhere!

**Transportation networks**

How should you design the highway network in a country? What is the quickest way to drive from Badda to Naraynganj?

**Communication networks**

How to send a network packet from BRACU internet to Yahoo mail server?

**Information networks**

Is the World wide a directed or undirected network?

**Social networks**

Facebook, Myspace, Flickr, ...

**Dependency networks**

What courses must you take before you can take CSE-423?

**Mazes**

Is there way out of a maze? If so, what is a path from entrance to exit?

Often a complex system can be reduced to a simple graph keeping only the essential information needed to solve the problem at hand. For example, the labyrinth or maze shown in [Fig. 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig1) can be reduced to a much simpler graph that has all the information needed to answer questions about the maze. In the graph representation, a vertex represents a *location* in the maze, and an edge represents connection between two adjacent locations. We can describe this graph as:

|  |
| --- |
| G = **(V,E)**  = ({A,B,C,D,E,F,G,H,I,J,K,L},  {(A,B),(A,C),(A,D),(B,E),(B,F),(C,F),(D,G),(D,H),(E,I),(E,J),  (G,H),(I,J),(K,L)}) |

Using the graph representation, we can ask a question typical of a maze: Is there a path from A to J? If so, what is the path, that is, what is a possible sequence of steps that we may take from A to get to J?

|  |  |
| --- | --- |
| Maze | graph representation |
| **Figure 1**: A maze (left) and it's graph representation (right) | |

Before we can start discussing how to answer such questions of a graph, let's first look at the different types of graphs, and the various terminology that we need to be familiar with.

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)

**Graph terminology**

* [Directed and undirected graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#directed)
  + [Directed acyclic graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dag)
* [Weighted and unweighted graphs](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#weighted)
* [Other graph terms](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#otherTerms)

**Directed and undirected graphs**

A graph is **directed** if E consists of **ordered** pairs. Conversely, a graph is **undirected** if E consists of **unordered** pairs. The graph in [Fig. 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig1) is undirected since all edges ∈ E are undirected. For example, the edge (A,B) ≡ (B,A), so is written out just once. Any undirected graph can be turned into a directed one by replacing each undirected edge (x,y) ∈ E with two directed edges (x,y) and (y,x) (which doubles the number of edges obviously). [Fig. 2](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig2) shows a directed (left) and an undirected (right) graph.

|  |  |
| --- | --- |
| Maze | graph representation |
| **Figure 2**: A directed (left) and an undirected graph (right) | |

If (u,v) ∈ E, then vertices u and v are **adjacent**.

**Directed Acyclic**

If a directed graph has no cycles, then it's called *directed acyclic graph* or **dag** for short. Dags are a very important subclass of graphs, and we'll be seeing them in many different applications such as dependency graphs (such as the graph that describes your course pre-requisites!).

**Weighted and unweighted graphs**

We can assign weight function to the edges: w(e) is a weight of edge e ∈ E. The graph which has such function assigned is called **weighted**. [Fig. 3](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig3) shows a weighted graph.

|  |
| --- |
| A weighted graph |
| **Figure 3**: A weighted graph |

A graph that does not have a weight function assigned to any of its edges is an **unweighted** graph. An unweighted graph is equivalent to a weighted one in which each edge is assigned the unit weight, such that ∀ e ∈ E, w(e) = 1.

**Other graph terms**

A few other terms that we'll be using everywhere from now on are given below.

**Degree**

Degree of a vertex v is the number of vertices u for which (u,v) ∈ E or (v,u) ∈ E (denote deg(v)). The number of **incoming edges** to a vertex v is called **in-degree** of the vertex (denote indeg(v)). The number of **outgoing edges** from a vertex is called **out-degree** (denote outdeg(v)). For an undirected graph, deg(v) = indeg(v) = outdeg(v).

**path**

a path is sequence of edges starting from a source vertex leading to a target vertex.

**simple path**

a path is \*simple\* if no vertex appears more than once in the path. In a tree, every path is simple.

**cycle**

a cycle is a simple path that has the same first and last vertex. A tree cannot have a cycle by definition, since there must be a single simple path between every pair of nodes.

**connected-ness**

if there is a path between every pair of vertices, then it is **connected**. A tree is connected by definition. A graph has one or more **connected components**. [Fig. 4](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig4) shows a graph with 2 connected components.

|  |
| --- |
| Graph with 2 connected components |
| **Figure 4**: Graph with 2 connected components |

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)

**Data structures and representations**

* [Adjacency Matrix](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#adjMatrix)
* [Adjacency List](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#adjList)
* [Comparison of the representations](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dsCompare)

**Adjacency Matrix**

The graph is represented as a n × n matrix A = (ai,j), where ai,j = 1 if (vi,vj) ∈ E, or 0 otherwise.

[Fig. 5](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig5) shows a directed graph and it's adjacency matrix representation.

|  |  |
| --- | --- |
| Directed graph | Adjacency matrix representation |
| **Figure 5**: A directed graph and it's adjacency matrix representation | |

[Fig. 6](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig6) shows an undirected graph and it's adjacency matrix representation.

|  |  |
| --- | --- |
| Undirected graph | Adjacency matrix representation |
| **Figure 6**: An undirected graph and it's adjacency matrix representation | |

**Adjacency List**

This represents the graph as an array of lists, where each array slot represents a vertex, and the list represents its adjacent vertices (respecting directionality in case of a directed graph).

[Fig. 7](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig7) shows a directed graph and it's adjacency list representation.

|  |  |
| --- | --- |
| Directed graph | Adjacency list representation |
| **Figure 7**: A directed graph and it's adjacency list representation | |

[Fig. 8](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig8) shows an undirected graph and it's adjacency list representation.

|  |  |
| --- | --- |
| Undirected graph | Adjacency list representation |
| **Figure 8**: An undirected graph and it's adjacency list representation | |

**Comparison of the representations //didn’t understand**

[Table 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#tbl1) shows the performances for the given operations using the two different representations of a graph.

|  |  |  |
| --- | --- | --- |
| **Operation** | **Directed graph** | **Undirected graph** |
| Is (u,v) ∈ E? | ≈ constant | ≈ outdeg(u) |
| List edges outgoing from u | ≈ n | ≈ outdeg(u) |
| Space requirement | n × n | ≈ m + n |
| **Table 1**: Comparison of adjacency matrix and list representations | | |

For **dense** graphs (graphs for which the number of edges m ≈ n2), the space requirement is the same for both, so the matrix representation wins. In general however, most graphs are **sparse** (graphs for which m << n2), so the adjacency list representation is more appropriate.

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)

**Tree Graphs**

A graph G with n vertices is a **tree** if any of the following are satisfied (in other words, the following are all equivalent):

1. There are n-1 edges and no cycles.
2. There are n-1 edges and G is connected.
3. Exactly one path connecting each pair of vertices.
4. G is connected, but does not remain connected if any edge is removed.

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)

**Graph traversal**

The graph traversal problem is stated as follows:

|  |
| --- |
| Given a graph G = (V,E), and a *distinguished vertex* s, how do you visit each vertex v ∈ V **exactly** once. |

The two traversals we will study are **breadth-first** and **depth-first** traversals.

* [Breadth-first traversal](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#bfs)
* [Depth-first traversal](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#dfs)

**Breadth-first traversal**

In breadth-first traversal, we start with the source vertex and *fan* out from there. It finds the distance (in terms of number of edges or hops) of each node from the source vertex. [Fig. 9](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig9) shows the breadth-first tree of the graph shown in [Fig. 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig1), using **A** as the source vertex. The vertices are annotated with the shortest distance from the source vertex, and the arrows show the ancestor of each vertex after the traversal. The solid lines represent the tree edges, and the dashed lines represent the non-tree edges, which are back edges in the case of this undirected graph. The figure on the right shows the BFS tree in a more *tree-like* fashion.

|  |  |
| --- | --- |
| BFS of undirected graph | BFS of undirected graph |
| **Figure 9**: Breadth-first search tree of graph in [Fig. 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig1). | |

Since we've only performed a single BFS traversal starting at A, the resulting BFS tree only contains the vertices that are in the same connected component. Vertices K and L are not **reachable** from A, so are not visited in this traversal. If we wanted to visit **all** the vertices, then we would need to perform a BFS traversal starting at **each** vertex v ∈ V in the graph. After each BFS traversal starting at any vertex, all the vertices in its connected component are marked as *visited*; a BFS traversal that starts at a vertex marked *visited* returns immediately, so there is not much loss of performance.

**Depth-first traversal**

In depth-first traversal, we start with a source vertex and recursively visit each adjacent vertex, backtracking only when all the choices are exhausted. [Fig. 10](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig10) shows the depth-first tree of the graph shown in [Fig. 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig1), using **A** as the source vertex. The vertices are annotated with the discovery and finish times of each vertex. The solid lines represent the tree edges, and the dashed lines represent the non-tree edges, which are back edges in the case of this undirected graph. The figure on the right shows the DFS tree in a more *tree-like* fashion.

|  |  |
| --- | --- |
| DFS of undirected graph | DFS of undirected graph |
| **Figure 10**: Depth-first search tree of graph in [Fig. 1](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#fig1). | |

Why do we care about these **discovery** and **finish** times for each vertex? We can use these numbers to **topologically sort** the graph. More (and much more!) on that next semester …

Since we've only performed a single BFS/DFS traversal starting at A, the resulting BFS/DFS tree only contains the vertices that are in the same connected component. Vertices K and L are not **reachable** from A, so are not visited in this traversal. If we wanted to visit **all** the vertices, then we would need to perform a BFS/DFS traversal starting at **each** vertex v ∈ V in the graph. After each BFS/DFS traversal starting at any vertex, all the vertices in its connected component are marked as *visited*; a DFS traversal that starts at a vertex marked *visited* returns immediately, so there is not much loss of performance.

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)

**Some questions to ask of a graph**

Here are a few typical questions to ask of a graph, and all of these can be answered using either BFS or DFS or both.

1. Starting from a given *source* vertex, can I reach a *target* vertex? This is the **s-t connectivity** problem.

Solution: Start a BFS or DFS starting at the *source* vertex, and if at any time during the traversal, we reach the *target* vertex, we know that the *target* is reachable from *source*. Remember that BFS and DFS visit all the vertices in the same connected component as the source vertex, which are all the vertices reachable from it.

1. Starting from a given *source* vertex, how can I visit all the other vertices reachable from the source?

Solution: See the previous solution.

1. Is the graph *connected*? If not, how many connected components does it have (and what are these)?

Solution: The first part can be answered by simply performing a BFS or DFS starting at any vertex. If, after the traversal is done, all the vertices have been visited, then the graph is connected.

For the second part, we initialize the number of components to be 0, and perform a BFS or DFS starting at each vertex v ∈ V in the graph. Every time we encounter a starting vertex that is yet unvisited (means it's either the first time we're performing a BFS/DFS, or it's not in the same component with another vertex that was used as a starting vertex), we increment the number of connected components.

1. Is there cycle in this graph? The presence of one or more **back edges** in the BFS or DFS-tree represent a cycle. Can be answered with either BFS or DFS traversal/search.

Solution: If, during a BFS or DFS traversal, we encounter **back edges**, the graph has at least one cycle. For an undirected graph, **all** non-tree edges are back edges; for directed graphs, there are 3 types of non-tree edges: (1) back, (2) forward, and (3) cross. Only a back edge represents a cycle.

1. Starting from a given *source* vertex, what is the shortest distance (in terms of the number of edges) of each vertex from the source vertex?

Solution: Perform a BFS, and the **discovery time** annotation of each vertex reachable from the *source* vertex show the shortest distance from the source. Note that this **cannot** be answered with DFS traversal.

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)

**Discussion**

Back to [Table of contents](http://moodle.bracu.ac.bd/mod/page/view.php?id=5314#toc)