

Data Structures and Algorithms II

Lecture 9: Graph Algorithms

- Motivation
- Terminology
- Graph Abstract Data Type.

1

Motivation

These can all be graphically represented:

Each of these is a graph structure.

3

Motivation

Many problems can be formulated in terms of

- a set of entities.
- relationships between them.

Examples:

- Route finding:
objects = towns, relationships = road/rail links.
- Course planning:
objects = courses, relationships = prerequisites.
- Circuit analysis:
objects = components, relationships = wire connections.
- Game playing:
objects = board state, relationships = moves.

2

Definition

A graph is a datastructure consisting of:

- a set of *vertices* (or nodes).
- a set of *edges* (or links) connecting the vertices.

ie, $G = (V, E)$ where V is a set of vertices, E = set of edges, and each edge is formed from pair of distinct vertices in V

If we represent our problem data using a graph data structure, can use standard graph algorithms (often available from code libraries) to solve it.

4

Graph Algorithms

Graph algorithms that we will look at include:

- Searching for a path between two nodes.
 - Can be used in game playing, AI, route finding, ..
- Finding shortest path between two nodes.
- Finding a possible *ordering* of nodes given some constraints.
 - e.g., finding order of modules to take; order of actions to complete a task.

5

Graphs and Trees

Graphs are more general than trees. (Trees are a special kind of connected graph, with no cycles and a “root”).

Tree terminology: children, parents, ancestors, siblings, decendants, root, leaf.

7

More Terminology

Need to be familiar with a number of terms, which can be explained graphically: directed/undirected; cyclic/acyclic; labelled; connected/unconnected; adjacent; neighbours; path;

6

A Graph ADT: Operations

Need to define:

- Operations for modifying and inspecting graph.
- Data structure for graph itself.

For simple *undirected, unlabelled* graph, a small set of operations is enough, to:

- Create a graph.
- Add and remove edges to the graph.
- Check if an edge exists.

If we assume all nodes are identified by a number, following C++ functions can be used:

```
graph();           // constructor
~graph();         // and destructor
                  // (may be empty)
void addedge(int n1, int n2);
void removedge(int n1, int n2);
logical edgeexists(int n1, int n2);
```

8

Graph ADT: Operations (cont)

For a labelled graph we might also want operations to add, remove and inspect labels. In C++ we might have:

```
void setlabel(int n, label l);  
label getlabel(int n);
```

where label is a suitable datatype

9

A Simple Graph ADT using C++ Classes

(Note: Friday will revise classes and inheritance)

Using above representation we can have following very simple class definition:

```
class graph()  
{  
public:  
    graph();  
    ~graph();  
    void addedge(int n1, int n2);  
    void removedge(int n1, int n2);  
    logical edgeexists(int n1, int n2);  
private:  
    logical g[MaxSize][MaxSize];  
}
```

- This uses a fixed size array. Not ideal as may want graphs of varying size. May use pointers to allow variable sized arrays.

- Also can improve with private variable to denote size of graph, and constructor argument to set size.

11

Graph ADT: Implementation

What built in or user defined datatypes can we use to represent a graph? Two methods:

1. Adjacency matrix method.

Use N x N array of boolean values:

	0	1	2	3
0	F	T	T	F
1	T	F	T	F
2	T	T	F	T
3	F	F	T	F

(or can just use integer, and 1/0)

If array name is G, then $G[n][m] = T$ iff edge exists between node n and node m.

10

Graph ADT

Write the function definition for "edge exists" based on this way of representing graphs.

12

Graph ADT: Implementation

2. Edge List Method.

Use 1d array, but make each item in array be a *list* of adjacent nodes:

```
0 (1 2)
1 (0 2)
2 (0 1 3)
3 (2)
```

Linked lists (e.g., IntList) may be used to represent these lists, so we could have a datastructure:

```
private:
    IntList edges[MaxSize];
```

But now to check whether edge exists between n1 and n2 need to look at list contained in edges[n1] and see if has n2 in the list.

13

Summary

- Graphs are used for problems where data consists of objects and relationships between objects.
- Graph = set of nodes (vertices) and set of edges between nodes.
- ADT needs operations to modify and inspect edges.
- Two main implementations: edge lists and adjacency matrix.

15

Comparison

Adjacency matrix is:

- Easy to implement
- Most operations are efficient.

but it uses a large array, and inefficient for *sparse* graphs.

Edge lists are:

- A little harder to implement.
- Some operations are less efficient, as require list traversal.

but it is much more efficient in terms of space, especially for sparse graphs.

14