Data Structures in C

Prof. Georg Feil

Graphs

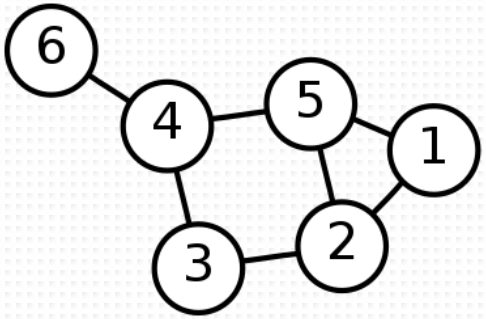
Summer 2017

Acknowledgement

- These lecture slides are based on slides by Professor Simon Hood
- Additional sources are cited separately

Graph definition

- A graph is a set of vertices along with a set of edges between pairs of vertices
- It represents pairs of objects which are related in some way
- The edges may be directed or undirected
- The image on the right shows a graph with 6 vertices and 7 edges that is undirected
  - The edges go in both directions
  - The vertices are also called nodes

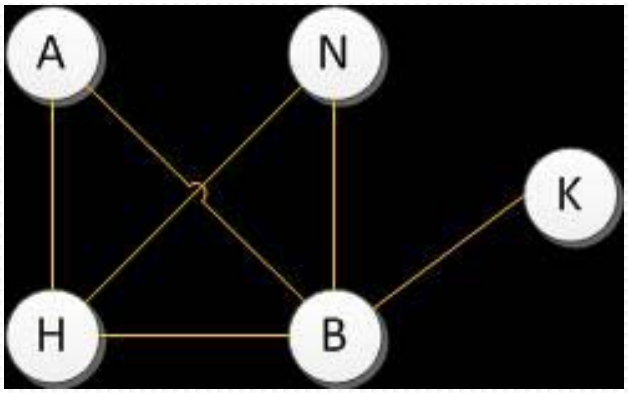


Graphs and Trees

- A tree is just a special case of a graph
- Graphs are more general than trees – an edge can connect to any node, not just nodes further down the tree
  - Unlike trees, graphs can have cycles
- Many difficult problems can be modeled and analyzed in terms of graphs

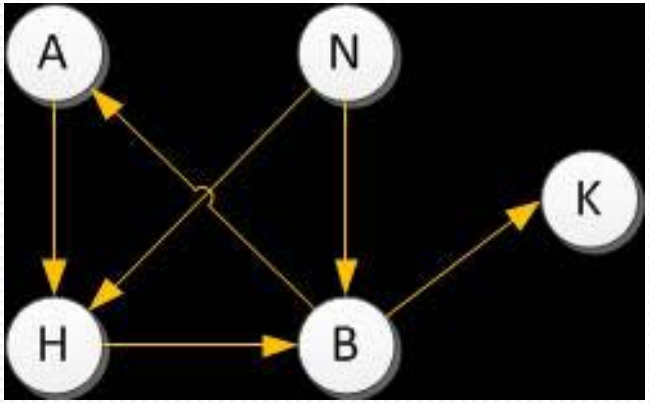
Undirected Graphs

- There are two types of graphs – undirected and directed
- In an undirected graph, we have two sets of information
  - Vertices {A, B, H, K, N}
  - The edges between the vertices {(A,H),(H,B),(B,K),(B,A),(N,H),(N,B)}



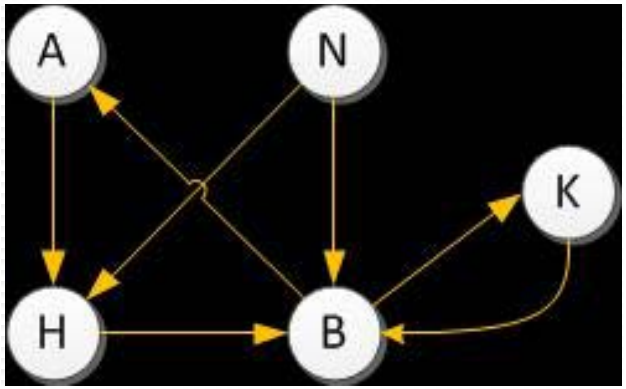
Directed Graphs

- A directed graph is just like an undirected graph (a set of vertices and edges), but the edges have a direction
- Note: this means that below there is an edge from A to H, but not from H to A



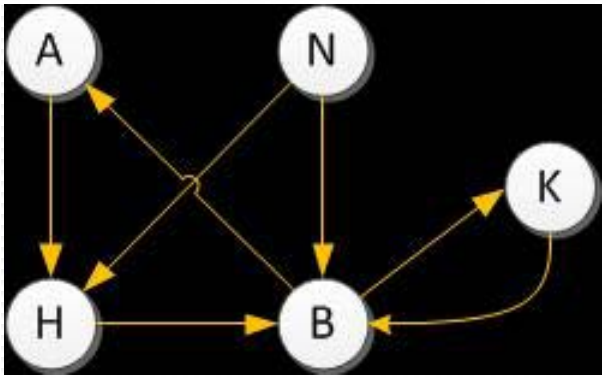
Directed Graphs

- There's an edge from B to K
- If we want to draw an additional edge from K back to B, in a directed graph we often do so with an additional line
- Notice there are two lines instead of one line with two arrows



Paths and Cycles

- A simple “path” through our graph might be N -> H -> B -> A
  - This is a path of length 3 (three edges are traversed)



- Many graphs contain cycles (this is one of the main points where graphs differ from trees)
- A cycle in this graph is A -> H -> B-> A

How to represent a graph

- There are two common ways to represent a graph
  - An n x n adjacency matrix (where n is the number of vertices)
  - n adjacency lists
- A 5 x 5 adjacency matrix for the previous graph might be drawn like this

	A	B	H	K	N
A			(A,H)		
B	(B,A)			(B,K)	
H		(H,B)			
K		(K,B)			
N			(N,H)	(N,K)	

How to represent a graph

- The same graph can also be represented as a series of ‘n’ adjacency lists
- Our 5 adjacency lists are as follows
  - Don’t read these as linked lists –they denote that the first vertex points to each other vertex to the right, but not in any particular order

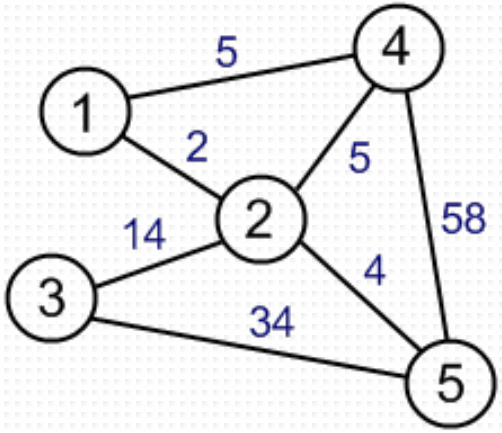
1. A -> H
2. B -> A -> K
3. H -> B
4. N -> B -> H
5. K -> B

Which representation?

- What do we use and when? Adjacency matrix or adjacency lists?
- It depends on the number of edges and the kinds of operations we may need to perform
  - You can use the implementation you prefer
- To be efficient, if the graph is dense (has a lot of edges), the matrix representation takes less space, but for a sparse graph, the list is more space efficient

Weighted graphs

- We can assign each edge a number or weight
  - It could represent a cost or length
  - For example if the vertices are cities the weights could be driving distances along highways between the cities



Building a graph

- Let's see how to build a graph with the following data – imagine it's stored in a file called "graph.txt"

7

A B C D E F G

A 2 B 25 C 45

B 3 D 50 E 60 F 55

C 2 B 10 G 70

D 0

E 3 C 65 D 20 F 30

F 1 D 15

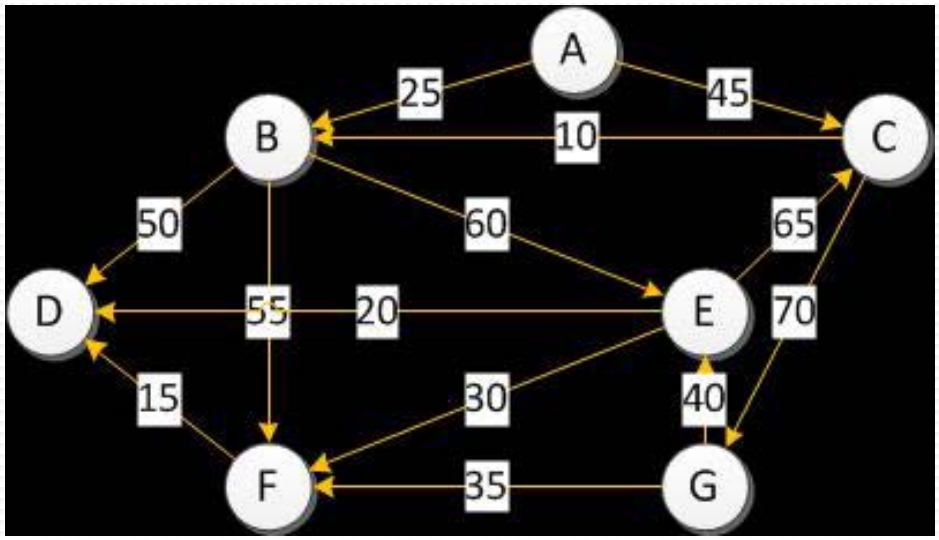
G 2 E 40 F 35

Building a graph

- The first line of the file says there are seven vertices. The next line gives the names or labels of the vertices
- It's followed by seven more lines, each of which describes the edges leaving a vertex
  - The first item of these seven lines is the name of the vertex, followed by the number of edges leaving it
- The next items in the seven lines are a series of pairs of values describing each individual edge
  - The vertex the edge leads to, and its weight

Building a graph

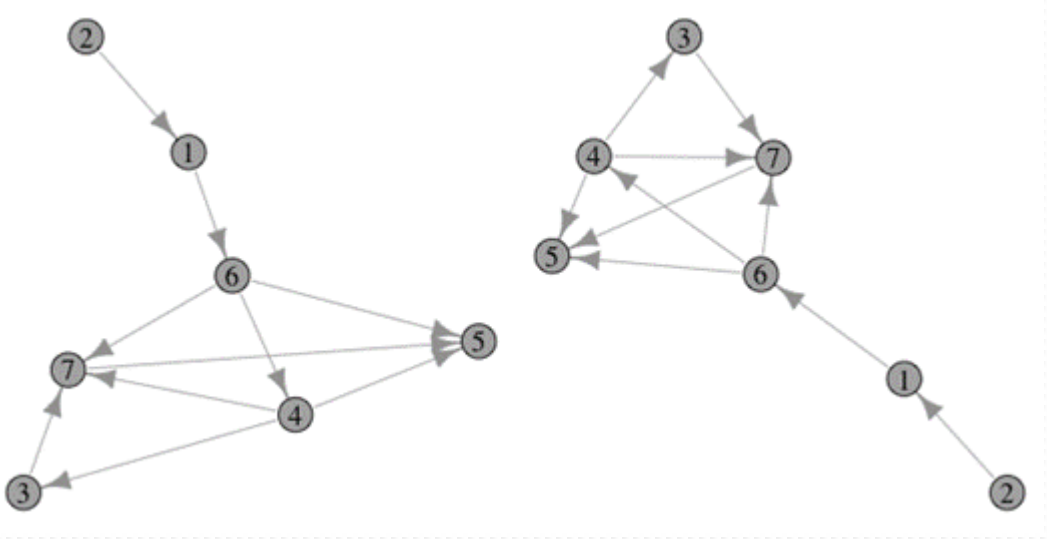
- So what does this look like?



- That's a pretty complex graph!
- It only took a small file to represent it

Different forms

- Note that there are many ways to draw the same graph
- How you draw it can make it look simpler or more complicated



C code for a graph

- I won't show a complete implementation of a graph, but let's look at the data structure (struct) definitions
  - We'll need a list of vertices, and a list of edges
- Vertices:

```
typedef struct {
    char id[MaxWordSize];
    int parent, cost, discover, finish, inDegree;
    GEdge* firstEdge;
} GVertex;
```

- The id field holds the name of the vertex, and firstEdge points to the first edge node

C code for a graph

- An edge leading from a vertex to a “child” vertex
- It has a next pointer so we can make a linked list of all edges for a particular vertex

```
typedef struct gedge {
    int child; // child is a vertex array index
    int weight;
    struct gedge* nextEdge;
} GEdge;
```

C code for a graph

- We can store the graph vertices in an array
- We'll also need a variable (numV) to store the number of vertices
- Let's create a graph structure

```
typedef struct graph {
    int numV;
    GVertex vertex[MaxVertices + 1];
} Graph;
```

Exercise 1

- Draw the directed graph denoted by the following adjacency list
- This is as much an exercise in art as it is understanding the graph file structure we are using
  - There are no incorrect answers as long as your graph follows the structure below

```
7
A B C D E F G
A 3 D 20 F 15 G 10
B 0
C 2 B 60 E 35
D 2 B 30 E 50
E 2 A 40 B 25
F 2 E 55 G 45
G 1 C 65
```

- Expect your page to be a complete mess of arrows and weights! After you have a correct version try to optimize the layout on a new page.

Traversing a Graph

Depth-First Traversal,
Breadth-First Traversal,
Dijkstra's Algorithm for Shortest Path

Traversing a graph

- See next set of slides by Simon Hood...