Graphs
CS 106B

Programming Abstractions
Fall 2016
Stanford University
Computer Science Department
Huffman Encoding

Beautiful mathematically

Used in everyday life (both JPEG and MP3)

Sweet history

Great practice with trees
const is a keyword that promises that this method will not change the value of any instance variables. It also means you can’t call non-const helper methods.
Thanksgiving

* You can come to LAiR or Class. But we won’t be there! LAiR is back on the Sunday the 27th of November
Who Do You Love

And how does Facebook know?
Main CS Collections

- Vector
- Grid
- Stack
- PQueue
- HashMap
- HashSet
- Queue
- Map
- Set

Grid
Main CS Collections

Saw under the hood of the main CS collections
Today’s Goal

1. Introduction to Graphs
2. Graphs in C++
Tree Definition

Only One Parent  No Cycles
WHAT IF I TOLD YOU

THERE ARE NO "RIIIILES"
A **graph** is a mathematical structure for representing relationships using nodes and edges.

*Just like a tree without the rules*
We can have a family tree?
Family Tree

Tywin

Tyrion

Cersi

Jamie
Not a Tree

Tywin

Tyrion

Cersi

Jamie

Joffrey
Not a Tree
We can have a family tree?
Graphs Don’t Have Roots

- Catelyn
- Tywin
- Tyrion
- Cersi
- Jamie
- Joffrey
- The High Sparrow
Simple Graph

```cpp
struct Node{
    string value;
    Vector<Edge *> edges;
};

struct Edge{
    Node * start;
    Node * end;
};

struct Graph{
    Set<Node *> nodes;
    Set<Edge*> edges;
};
```
struct Node{
    string value;
    Vector<Edge *> edges;
};

struct Edge{
    Node * start;
    Node * end;
};

struct Graph{
    Set<Node *> nodes;
    Set<Edge *> edges;
};

We allow for more interesting edges
Simple Graph

```cpp
struct Node{
    string value;
    Vector<Edge *> edges;
};

struct Edge{
    Node * start;
    Node * end;
    double weight;
};

struct Graph{
    Set<Node *> nodes;
    Set<Edge *> edges;
}

We allow for more interesting edges
```
A graph consists of a set of nodes connected by edges.
A graph consists of a set of nodes connected by edges.
A graph consists of a set of **nodes** connected by **edges**.
A graph consists of a set of **nodes** connected by **edges**.
Directed Graph
Undirected Graph

CAN  MAN  RAN

CAT  SAT  RAT
Directed vs Undirected
Weighted graphs

**weight**: Cost associated with a given edge.

**example**: graph of airline flights, weighted by miles between cities:
Prerequisite Graph

- simple C++
- collections
- function calls
- recursion
  - fractals
  - exploration recursion
  - definition recursion
Social Network
The Internet
The Internet

10 to 20 billion
50,000 unique implementations of logistic regression in CS229
Chemical Bonds
Road Map
Corruption

“The Evolution of FCC Lobbying Coalitions” by Pierre de Vries in JoSS Visualization Symposium 2010
Partisanship
<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>N</th>
<th>C</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>P</td>
<td>Y</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>H</td>
<td>T</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Boggle
Some terms:
Paths

- **path**: A path from vertex \(a\) to \(b\) is a sequence of edges that can be followed starting from \(a\) to reach \(b\).
  - can be represented as vertices visited, or edges taken
  - example, one path from \(V\) to \(Z\): \(\{b, h\}\) or \(\{V, X, Z\}\)
  - What are two paths from \(U\) to \(Y\)?

- **path length**: Number of vertices or edges contained in the path.

- **neighbor** or **adjacent**: Two vertices connected directly by an edge.
  - example: \(V\) and \(X\)
• **cycle**: A path that begins and ends at the same node.
  – example: \{b, g, f, c, a\} or \{V, X, Y, W, U, V\}.
  – example: \{c, d, a\} or \{U, W, V, U\}.

  – **acyclic graph**: One that does not contain any cycles.

• **loop**: An edge directly from a node to itself.
  – Many graphs don't allow loops.
• **reachable**: Vertex \( a \) is *reachable* from \( b \) if a path exists from \( a \) to \( b \).

• **connected**: A graph is *connected* if every vertex is reachable from every other.

• **complete**: If every vertex has a direct edge to every other.
The Stanford C++ library includes a BasicGraph class.
– Based on an older library class named Graph

You can construct a graph and add vertices/edges:

```cpp
#include "basicgraph.h"
...
BasicGraph graph;
graph.addVertex("a");
graph.addVertex("b");
graph.addVertex("c");
graph.addVertex("d");
graph.addEdge("a", "c");
graph.addEdge("b", "c");
graph.addEdge("c", "b");
graph.addEdge("b", "d");
graph.addEdge("c", "d");
```
# BasicGraph members

```cpp
#include "basicgraph.h"  // a directed, weighted graph

// members

```g.addEdge(v1, v2);``` | adds an edge between two vertexes |
| g.addVertex(name); | adds a vertex to the graph |
| g.clear(); | removes all vertexes/edges from the graph |
| g.getEdgeSet() | returns all edges, or all edges that start at v, as a Set of pointers |
| g.getEdgeSet(v) | |
| g.getNeighbors(v) | returns a set of all vertices that v has an edge to |
| g.getVertex(name) | returns pointer to vertex with the given name |
| g.getVertexSet() | returns a set of all vertexes |
| g.isNeighbor(v1, v2) | returns true if there is an edge from vertex v1 to v2 |
| g.isEmpty() | returns true if queue contains no vertexes/edges |
| g.removeEdge(v1, v2); | removes an edge from the graph |
| g.removeVertex(name); | removes a vertex from the graph |
| g.size() | returns the number of vertexes in the graph |
| g.toString() | returns a string such as "\{a, b, c, a \rightarrow b\}" |
BasicGraph members

```
#include "basicgraph.h"  // a directed, weighted graph

```g`.addEdge(v1, v2);` | adds an edge between two vertices
`g`.addVertex(name);` | adds a vertex to the graph
`g`.clear();` | removes all vertexes/edges from the graph
`g`.getEdgeSet()` | returns all edges, or all edges that start at `v`, as a Set of pointers
`g`.getEdgeSet(v)`
`g`.getNeighbors(v)` | returns a set of all vertices that `v` has an edge to
`g`.getVertex(name)` | returns pointer to vertex with the given name
`g`.getVertexSet()` | returns a set of all vertexes
`g`.isNeighbor(v1, v2)` | returns true if there is an edge from vertex `v1` to `v2`
`g`.isEmpty()` | returns true if queue contains no vertexes/edges
`g`.removeEdge(v1, v2);` | removes an edge from the graph
`g`.removeVertex(name);` | removes a vertex from the graph
`g`.size()` | returns the number of vertexes in the graph
`g`.toString()` | returns a string such as "\{a, b, c, a -> b\}"
Using BasicGraph

The graph stores a struct of information about each vertex/edge:

```c
struct Vertex {
    string name;
    Set<Edge*> edges;
    double cost;
    // other stuff
};
```

```c
struct Edge {
    Vertex* start;
    Vertex* finish;
    double weight;
    // other stuff
};
```

You can use these to help implement graph algorithms:

```c
Vertex * vertC = graph.getVertex("c");
Edge * edgeAC = graph.getEdge("a", "c");
```
Our First Graph

A

B

C

B → A

C ← B

C → B
There are other representations...
... this is the one we are going to use.
Algorithms
Who Do You Love

And how does Facebook know?
Ego Graph
Maybe I Love These People?
But I Actually Love This Person

Your significant other

376
Romantic Partnerships and the Dispersion of Social Ties: A Network Analysis of Relationship Status on Facebook

Lars Backstrom
Facebook Inc.

Jon Kleinberg
Cornell University

ABSTRACT
A crucial task in the analysis of on-line social-networking systems is to identify important people — those linked by strong social ties — within an individual’s network neighborhood. Here we investigate this question for a particular category of strong ties, those involving spouses or romantic partners. We organize our analysis around a basic question: given all the connections among a person’s friends, can you recognize his or her romantic partner from the network structure alone? Using data from a large sample of Facebook users, we find that this task can be accomplished with high accuracy, but doing so requires the development of a new measure of tie strength that we term ‘dispersion’ — the extent to which two people’s mutual friends are not themselves well-connected. The results offer methods for identifying types of structurally significant people in on-line applications, and suggest a potential expansion of existing theories of tie strength.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database applications—Data mining
Keywords: Social Networks; Romantic Relationships.

they see from friends [1], and organizing their neighborhood into conceptually coherent groups [23, 25].

Tie Strength.
Tie strength forms an important dimension along which to characterize a person’s links to their network neighbors. Tie strength informally refers to the ‘closeness’ of a friendship; it captures a spectrum that ranges from strong ties with close friends to weak ties with more distant acquaintances. An active line of research reaching back to foundational work in sociology has studied the relationship between the strengths of ties and their structural role in the underlying social network [15]. Strong ties are typically ‘embedded’ in the network, surrounded by a large number of mutual friends [6, 16], and often involving large amounts of shared time together [22] and extensive interaction [17]. Weak ties, in contrast, often involve few mutual friends and can serve as ‘bridges’ to diverse parts of the network, providing access to novel information [5, 15].

A fundamental question connected to our understanding of strong ties is to identify the person’s social network,

Dispersion: The extent to which two people’s mutual friends are not directly connected.
Dispersion: The extent to which two people’s mutual friends are not directly connected.
Dispersion: The extent to which two people’s mutual friends are not directly connected.
Dispersion: The extent to which two people’s mutual friends are not directly connected

Dispersion: 0
Dispersion: The extent to which two people’s mutual friends are not directly connected
Dispersion: The extent to which two people’s mutual friends are not directly connected
Dispersion: The extent to which two people’s mutual friends are not directly connected.
Dispersion: The extent to which two people’s mutual friends are not directly connected.

Dispersion: 4
Dispersion: The extent to which two people’s mutual friends are not directly connected.
Dispersion: The extent to which two people’s mutual friends are not directly connected
Who Do You Love?

Your significant other

376
Today’s Goals

1. Introduction to Graphs
2. Graphs in C++