Welcome back!

Assignment 9:

- released Wednesday
- due next Sunday

Today:

- graphs
  - high level recap
  - ADT
  - vertex and edge implementations
    - STL vectors
  - Building a graph
    - inserting a vertex
    - inserting an edge

Recap:

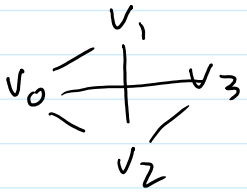
What is a graph?

- A collection of vertices connected by edges.

$$G = \{V, E\}$$

- each vertex contains a "key" and a list of edges (adjacent vertices)

- list of edges is stored in an adj. matrix or adjacency list



V_0	V_1	V_2	V_3
key	key	key	key
V_1	V_0	V_0	V_0
V_2	V_2	V_1	V_2
V_3		V_3	

- unlike a BST, there are no set relationships b/w vertices

$V_0 \not\rightarrow V_1$

- all relationships have to be explicitly set

ADT

- undirected

- weighted

private:
    vertices

public:

    init( )
    insertVertex(value)
    insertEdge(startVertex, endVertex, weight)
    deleteVertex(value)
    deleteEdge(startValue, endValue)
    printGraph( )
    search(value)

struct vertex {

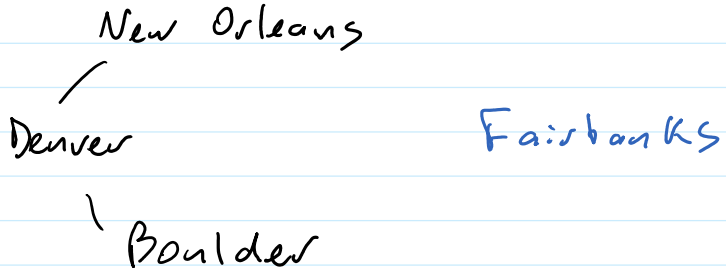
```
string key;
vector<adjVertex> adj;
};

struct adjVertex {
  vertex *v;
  int weight;
}
```

STL Vectors

- Include the vector library
- allows to index into elements like an array
- can append and grow length where memory allocation happens automatically
- can define container type to be either a primitive type or object (class or struct)

Insert vertex e.g addVertex("Fairbanks")



If using vectors:

1) search to ensure no duplicate keys exist

2) append to the end (push_back(x))

Add Edge: e.g. addEdge("Boulder", "Fairbanks", 4)

goal: find two given vertices in graph and insert edge to both adjacency lists

loop across all vertices

if "Boulder" is found

↳ loop across all vertices to find "Fairbanks"

1) Add an entry to Boulder's adjacency list w/ pointer to Fairbanks

2) Add entry to Fairbanks adj list w/ pointer to Boulder

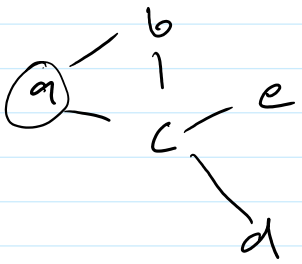
Next up

Displaying and searching graphs in various orders

- Breadth first

e.g. start w/ a

6



e.g. start w/ a

- Breadth first

a, c, b, e, d

- Depth first

a, c, d, e, b