11.2 graph implementation and algorithms

Friday, November 18, 2022

Reminder: lab 8 is due on Monday!

TODAY: implementation details for the Graph ADT

REVIEW: we've seen

- LISTS to represent linear relationships
- TREES to represent hierarchical relationships
- GRAPHS to represent general relationships

Graphs get used for many applications: maps, social networks, the web, etc.

graph $G = (V, E)$ — V: set of vertices, E: set of edges

- edges are directed/undirected
- edges are weighted/unweighted

We will focus on SIMPLE graphs:

X no self-loops

X no duplicate edges

examples we will not consider

GRAPH ADT

templated on 3 things:

- V vertex label type (must be unique)
- E edge label type (can have duplicates)
- W edge weight type (usually numerical)

methods:

```
void insertVertex(V vertex)
void removeVertex(V vertex)
void insertEdge(V source, V destination, E label, W weight)
void removeEdge(V source, V destination)
vector<Edge<V,E,W>> getEdges()
vector<V> getVertices()
bool containsVertex(V vertex)
bool containsEdge(V source, V destination)
```

Edge<V,E,W> getEdge(V source, V destination)

vector<Edge<V,E,W>> getOutgoing(V vertex)

vector<Edge<V,E,W>> getIncoming(V vertex)

vector<V> getNeighbors(V vertex)

IMPLEMENTING THE GRAPH ADT

Adjacency matrix

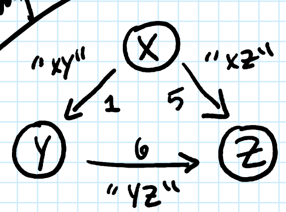
idea: store a 2D array of bits

destination

source		X	Y	Z
	X	0	1	1
	Y	0	0	1
	Z	0	0	0

1 means "there is an edge from this source to this destination"

example



types:

V char/string

E string

W int

Problem: not clear how to store edge labels and edge weights

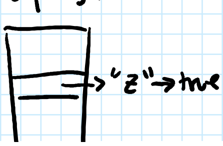
Adjacency dictionary: 2 data members

- STLHashTable < V, bool > vertices;
- STL HashTable < V, STLHashTable < V, pair < E, W >> * > edges;

source — destination — edge label & weight

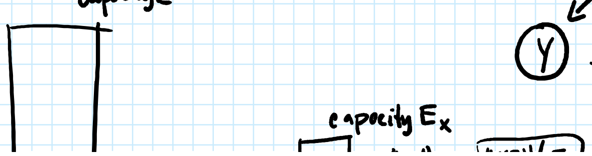
vertices hashtable

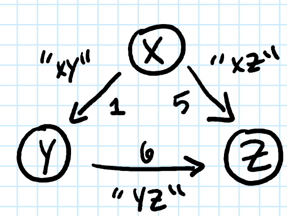
capacity V

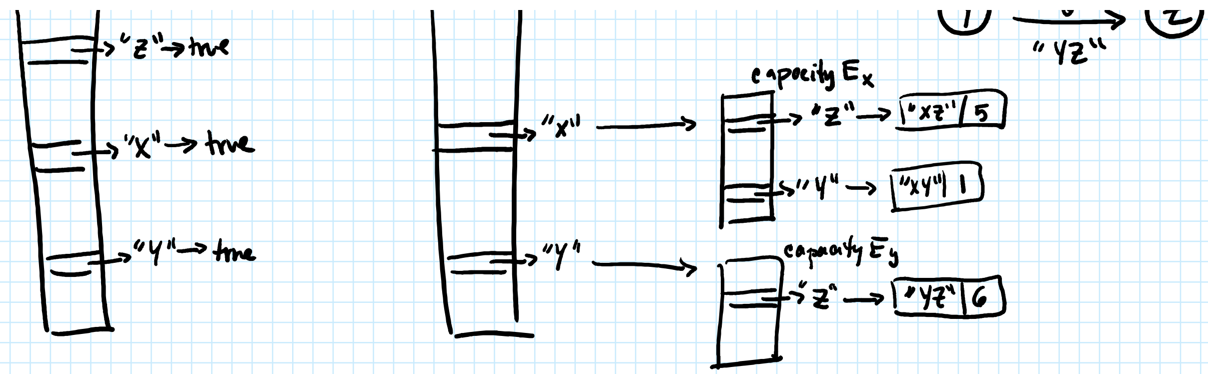


edges hashtable

capacity E







We store both the vertices & edges as data because there may be some vertices with no outgoing edges (like Z in the example).

Graph<V,E,W> methods

void insertVertex(V vertex) - use hash table insert - $O(1)$ amortized

void removeVertex(V vertex) - use hash table remove and also remove edges - $O(1 + \text{capacity}\, E)$

void insertEdge(V source, V destination, E label, W weight) - use hash table insert $O(1)$ amortized

void removeEdge(V source, V destination) use hash table remove $O(1)$ assuming a good hash function

vector<Edge<V,E,W>> getEdges() - scan all of edges hash table, then scan each hash table it points to $O\left(\text{capacity}\, E + \sum_{i=1}^{n} \text{capacity}\, E_i\right)$

vector<V> getVertices() - getKeys on vertices hashtable $O(\text{capacity}\, V)$

bool containsVertex(V vertex) - hash table contains $O(1)$ with a good hash fn

bool containsEdge(V source, V destination) - use hash table contains $O(1)$ with a good hash fn

Edge<V,E,W> getEdge(V source, V destination) - use hash table get $O(1)$ with a good hash fn

vector<Edge<V,E,W>> getOutgoing(V vertex) - use getItems on the specific vertex's hash table $O(\text{capacity}\, E_{vertex})$

vector<Edge<V,E,W>> getIncoming(V vertex) - scan all of edges hashtable, then lookup $O(\text{capacity}\, E)$

vector<V> getNeighbors(V vertex)

APPLICATIONS OF GRAPHS

We will use graphs to answer many sorts of questions:

- What's the shortest path between two vertices?
- What's the least expensive path between two vertices?

- Is it possible to get from one vertex to another specific vertex?
- Is it possible to reach every other vertex from a specific starting vertex?
- Is it possible to reach every vertex from every other vertex?

We've already seen some algorithms we could use! DFS can answer "is there a path between these two vertices?"

<u>pseudocode</u> for reachability

```
bool reachableDFS(V start, V end, Graph<V,E,W> * g)
    Stack<V> stack
    Dictionary<V, bool> visited
    stack.push(start)
    visited.insert(start, true)
    while (! stack.isEmpty())
        V current = stack.pop()
        if current == end
            return true
        for each n in g->getNeighbors(current)
            if (! visited.contains(n))
                stack.push(n)
                visited.insert(n, true)
    return false
```

We used this in the maze lab!

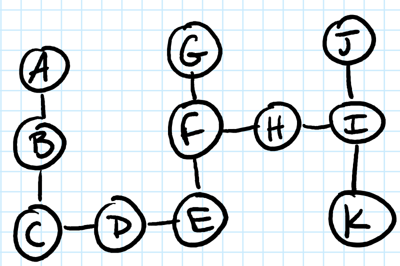
example1.map

start →

.	#	.	#	.
.	#	.	.	.
.	.	.	#	.

← exit

We can convert this to a graph g



\# wall
. open space

reachableDFS(A, K, g)

Let's run reachableDFS:

~~K~~
J
~~I~~

Let's run reachableDFS:

J
~~I~~
~~H~~
G
~~F~~
~~E~~
~~D~~
~~C~~
~~B~~
~~A~~
Stack

current: ~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ ~~H~~ ~~I~~ K
returns true!

visited: A B C D E F G H I J K

We will implement three graph algorithms:

1. bool reachableDFS(V start, V end, Graph<V,E,W>* g)
2. vector<V> shortestLengthPathBFS(V start, V end, Graph<V,E,W>* g)
   //ignores weights and returns the path with fewest number of edges
3. Dictionary<V,W>* singleSourceShortestPath(V start, Graph<V,E,W>* g)
   //finds the length of the shortest path from start to every vertex in graph