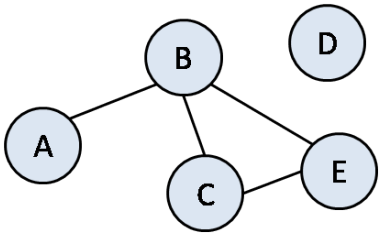
Recitation 12: Computing on the Edge

Thursday April 18th

Graphs, vertices, and edges

A *graph* G is a set of vertices V and a set of edges E, where each edge is a pair of vertices.



For example, the graph above would be described by giving the vertex set and the edge set as in

$$\begin{aligned} V &= \{A, B, C, D, E\} \\ E &= \{(A, B), (B, C), (B, E), (C, E)\} \end{aligned}$$

(For brevity, only edges in one direction are listed. This is okay because the graph above is *undirected*.)

The *subgraph* consisting of only vertices B, C and E is called a *complete graph* because an edge exists between every two of those vertices. In general, a complete graph with v vertices has $\frac{v(v-1)}{2}$ edges.

Checkpoint 0

Recall from lecture that we discussed two ways of representing graphs: adjacency matrices and adjacency lists. Draw the adjacency matrix and the adjacency list for the graph above.

Checkpoint 1

Fill in the following table with the worst-case bounds for both the adjacency matrix and adjacency list representations. Express your answers in terms of the number v of vertices and e of edges in a graph.

	Adjacency Matrix	**Adjacency List**
Space	$O($_____$)$	$O($_____$)$
`graph_hasedge`	$O($_____$)$	$O($_____$)$
`graph_addedge`	$O($_____$)$	$O($_____$)$
`graph_get_neighbors`	$O($_____$)$	$O($_____$)$

We call a graph *sparse* if it has relatively few edges and *dense* if it has relatively many edges. Specifically, a graph with v vertices and e edges is sparse if $e \in O(v)$).

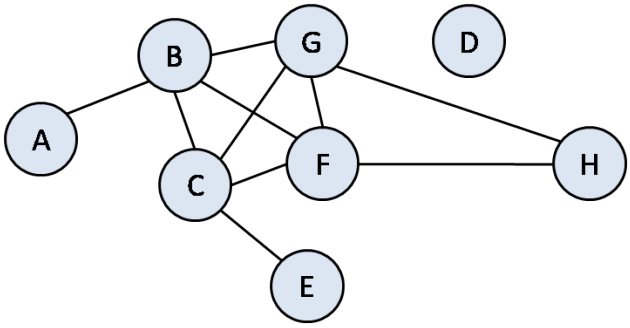
Checkpoint 2

Which representation might you want to use for a sparse graph? What about for a dense graph? Why?

Graph search

We've discussed two algorithms for traversing a graph: depth-first search (DFS) and breadth-first search (BFS). DFS always visits the first neighbor of a vertex before visiting the rest of the neighbors, whereas BFS visits all neighbors of a vertex before continuing the search in a neighbor's neighbor.

Here's again the graph seen earlier:



Checkpoint 3

Assume you wish to search the above graph for vertex H starting at vertex A. List the vertices of the graph in the order visited by DFS and BFS. Assume that the algorithms consider neighbors in alphabetical order.

Now list the vertices in order of distance from A. Do you notice anything?

Checkpoint 4

In class we studied a recursive implementation of DFS. Finish the following iterative implementation of DFS. Note that the structure of this function is different from what we saw for BFS.

```
1  bool dfs(graph_t G, vertex start, vertex target) {
2    REQUIRES(G != NULL);
3    REQUIRES(start < graph_size(G) && target < graph_size(G));
4
5    if (start == target) return true;
6
7    bool mark[graph_size(G)];  // what kind of array is this?
8    for (unsigned int i = 0; i < graph_size(G); i++)
9      mark[i] = false;
10
11   _____________________________________________________; // initialize data structure
12
13   _____________________________________________; // put start into our data structure
14
15   while (_____________________________________) { // until our data structure is empty
16
17     vertex u = _____________________________; // retrieve a vertex from our data structure
18     if (!mark[u]) {          // examine u only if unmarked
19       mark[u] = true;        // mark it
20       if (u == target) {     // if u is the target return true
21         printf("   Found!!!\n");
22
23         _____________________________________________; // free our data structure
24         return true;
25       }
26       neighbors_t nbors = graph_get_neighbors(G, u);
27
28       while (_____________________________) { // while there are unvisited neighbors
29
30         vertex w = _____________________________________; // grab the next neighbor
31
32         _____________________________________; // add it to our data structure
33       }
34       _____________________________________________________; // free neighbors
35     }
36   }
37   _____________________________________________; // free our data structure
38   return false;
39 }
```

Checkpoint 5

What is the time complexity of this function

- using an adjacency list representation? $O($_____$)$
- using an adjacency matrix representation? $O($_____$)$

REFERENCE: Graph Interface

```
typedef unsigned int vertex;
typedef struct graph_header *graph_t;

graph_t graph_new(unsigned int numvert);
//@ensures \result != NULL;

void graph_free(graph_t G);
//@requires G != NULL;

unsigned int graph_size(graph_t G);
//@requires G != NULL;

bool graph_hasedge(graph_t G, vertex v, vertex w);
//@requires G != NULL;
//@requires v < graph_size(G) && w < graph_size(G);

void graph_addedge(graph_t G, vertex v, vertex w);
//@requires G != NULL;
//@requires v < graph_size(G) && w < graph_size(G);
//@requires v != w && !graph_hasedge(G, v, w);


typedef struct neighbor_header *neighbors_t;

neighbors_t graph_get_neighbors(graph_t G, vertex v);
//@requires G != NULL && v < graph_size(G);
//@ensures \result != NULL;

bool graph_hasmore_neighbors(neighbors_t nbors);
//@requires nbors  != NULL;

vertex graph_next_neighbor(neighbors_t nbors);
//@requires nbors  != NULL;
//@requires graph_hasmore_neighbors(nbors);

void graph_free_neighbors(neighbors_t nbors);
//@requires nbors != NULL;
```