15-122 : Principles of Imperative Computation, Fall 2015

Written Homework 13

Due: Monday December 7, 2015 by 6:00 PM

Name: _____

Andrew ID: _____

Section: _____

This written homework covers graphs.

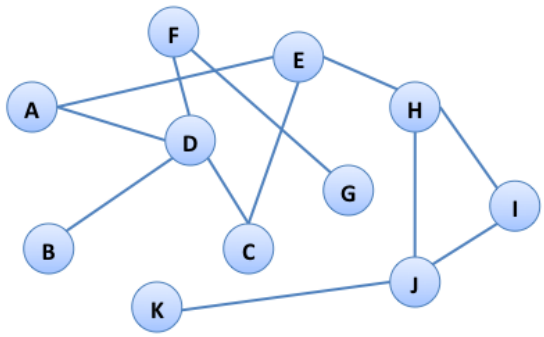
Print out this PDF double-sided, **staple** pages in order, and write your answers on these pages *neatly in ink or dark pencil*. You can hand in the assignment to your TA during lab or in the box outside of GHC 4117 (in the CS Undergraduate Program suite). **Warning: The box is removed promptly at 6PM.**

You must hand in your homework yourself;
do not give it to someone else to hand in.

Question	Points	Score
1	6	
2	9	
Total:	15	

1. **Graphs and Graph Traversals**

4pts (a) Consider the graph below:



Using a depth first traversal, list the vertices in the order that they are visited as we search from vertex J to vertex G. When we visit a vertex, we explore its outgoing edges in alphabetical order. Do not list a vertex again if you backtrack to it.

Solution:

List the vertices of the path found from J to G by the search.

Solution:

Using a breadth first traversal, list the vertices in the order that they are visited as we search from vertex J to vertex G. When we visit a vertex, we explore its outgoing edges in alphabetical order.

Solution:

List the vertices of the path found from J to G by the search.

Solution:

2pts (b) In an undirected graph with v vertices, what is the maximum possible number of edges? (This kind of graph is called a *complete graph*). Express your answer in closed form as a function of v.

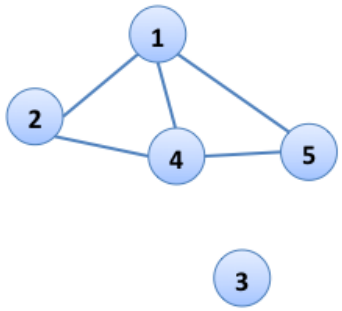
Solution:

A path in a graph is called a *simple cycle* if it lets you go from a vertex to itself without repeating an edge or any intermediate vertex. What is the maximum possible number of edges in a graph with v vertices that contains no simple cycles?

Solution:

2. **Graph representation**

1pt (a) Show the adjacency matrix that represents the graph drawn below (use the format shown in the lecture notes):



Solution:

3pts (b) Recall the *adjacency list* representation of a graph from class:

```
typedef unsigned int vertex;
typedef struct graph_header* graph;
typedef struct adjlist_node adjlist;
struct adjlist_node {
    vertex vert;
    adjlist *next;
};
struct graph_header {
    unsigned int size;
    adjlist *adj[];
};
```

Extend the graph interface with a function `graph_countedges(G, v)` that returns the number of edges at vertex v of graph G. Be sure to include appropriate `REQUIRES` and `ENSURES` contracts. You may call any functions given in the code in class posted on our website for the lecture on representing graphs. Your solution should be as efficient as possible, without making any changes to the definition of any data structure used in the graph representation.

Solution:

```
unsigned int graph_countedges(graph* G, vertex v) {









 }
```

1pt (c) Give the worst-case asymptotic complexity of your function for a graph of v vertices and e edges, as a function of v and e.

Solution:

3pts (d) Recall the interface to the graph library in `graph.h`:

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

graph_t graph_new(unsigned int numvert); // New graph with numvert vertices
void graph_free(graph_t G);
unsigned int graph_size(graph_t G);  // Number of vertices in the graph

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

void graph_addedge(graph_t G, vertex v, vertex w); // Edge can't be in graph!
  //@requires v < graph_size(G) && w < graph_size(G);
  //@requires v != w && !graph_hasedge(G, v, w);
```

Write another function to count the edges at a vertex. This must be a client function, that is, it must use only the types and functions provided in `graph.h`. You may use the fact that `vertex` is an integer type, and that it is the same type returned by `graph_size`.

Solution:

```
unsigned int countedges(graph_t G, vertex v) {




 }
```

1pt (e) Give the worst-case asymptotic complexity of your function for a graph of v vertices and e edges, as a function of v and e.

Solution: