Tute 4: Graphs & Standard Library

Review: Graphs

What is a graph?

A graph is a data structure consisting of a finite set of nodes (or vertices), connected by edges (or arcs), and edge(A,B) is said to connect node A and B. A graph is similar to a tree data structure, except that a node can have any number of nodes joined to it (unlike a binary tree, where each node can have a max of 2 children and 1 parent).

What is an adjacency list?

An Adjacency List is one potential mechanism for determining what nodes are connected to each other in a graph, ie, every node in the graph stores a list of nodes, and any nodes that appear in that list are said to be connected to the original node. Adjacency Lists can be ordered or unordered (ie, if node A appears in node B's list, then node A can be reached from node B. In an unordered graph, node A would appear in node B's list, and node B would appear in node A's list).

How would you store a graph in a data structure to ensure safe automatic freeing of memory?

It really does depend on the configuration of the data. If there are no cyclic dependencies (pointers to data that then point back to the original data), unique_ptr may be sufficient. However if there are cyclic dependencies, a combination of shared_ptr and weak_ptr may be required. A shared_ptr maintains a reference count of how many objects point at it and when that count reaches 0, it is safe to free the memory. A weak_ptr is a copy of the shared_ptr that does not increment the reference count.

Programming Concepts: Vectors

What is a vector?

A vector is a sequence container (stores data in order) that represents an array that can dynamically change in size. Vectors use contiguous memory to store their elements, just like arrays, which means their data can be accessed and used in the same way as an array. Internally vectors use a dynamically allocated array to store their elements, which is reallocated any time the vector runs out of space, typically doubling in size each time, which means that vectors will not necessarily reallocate memory every time an element is added, only when needed. This gives vectors an amortized constant time for data insertion (amortized meaning, average time taken per operation, if you do many operations).

How can a vector be used to store and retrieve data?

Data can be added to a vector using the push_back() method. Vectors can also be resized using the resize() method, and then data can be inserted directly at a specific location using the subscript operator [], ie vector[10] = value;

How can data from a file be read to/from a vector?

Reading from a binary file (with an already opened stream object called “in”:

```
vector<int> ints;
std::size_t num_ints;
in.read(reinterpret_cast<char*>(&num_ints), sizeof(int));
ints.resize(num_ints);
in.read(reinterpret_cast<char*>(&ints[0], sizeof(int) * num_ints);
```

read expects a char* (ie, an array of bytes), followed by the number of bytes to read from the file.

How can the standard library be used to select random elements from a vector?

The C++ Standard Library provides different tools for generating random numbers. For the assignment we’ll be using the Mersenne Twister, which can be created as:

```
#include <random>
std::mt19937 rng(seed);
```

It can then be used to generate a random number by calling it directly (the rng above is a functor):

```
auto randomNumber = rng();
```

To use it to retrieve a random element from a vector, you simply use the modulus of the vector’s size:

```
edge e = edges[rng() % edges.size()];
or
uniform_int_distribution<int> dist(0,edges.size());
edge e = edges[dist(rng)];
```

The main advantage of using mt19937 instead of the standard rand() function provided by C is that you can create multiple random number generators with different seeds, and have them generating random numbers simultaneously. The mersenne twister also repeats itself less often so you get a better distribution.

Compilation: None this week

Errors: Seg Faults

What is a seg fault?

A segmentation fault is an attempt to access a memory location that the program is not allowed to access (ie, writing to a red only location, or reading from a location that cannot be addressed). A common seg fault is to attempt to access a member of a null pointer, or attempting to dereference a null pointer.

What is an easy seg fault you can introduce to a program?

An easy seg fault is to create a pointer and have it point to null memory, then dereference it when attempting to use it, ie:

```
int *i = nullptr;
std::cout << *i << “\n”;
```

Debugging: Stack Frames

How can we see a list of the current stack frames when a program crashes?

When a program has crashed due to a seg fault, or paused for any other reason inside gdb, a list of stack frames can be produced with the command "bt" (backtrace). This will allow you to move between stack frames to analyse the execution of a program.

How can we analyse an individual stack frame?

After running the backtrace command, the command "f" can be used, followed by a number, where the number represents which frame you want to examine.

How can we analyse an individual variable when inside a stack frame?

Once inside a stack frame, the relevant section of code that is being executed will be displayed. Variables can be accessed and their values printed with the command "p" followed by the variable name. Note that you can also dereference variables when printing, or even access particular members of a variable.

Exercises:

Write a program to iterate through a vector of ints, pushing them onto a stack. Then pop them off the stack, printing each to the console.

Make sure to explain what a stack is and how to use it, since it is needed for the assignment.

```
#include <vector>
#include <stack>
#include <iostream>
#include <cstdlib>

int main(int argc, char **argv)
{
	std::vector<int> intVector = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
	std::stack<int> intStack;

	for (auto i : intVector)
		intStack.push(i);

	while (!intStack.empty())
	{
		std::cout << intStack.top() << ", ";
		intStack.pop();
	}

	std::cout << "\n";

	return EXIT_SUCCESS;
}
```