CMPT 125: Introduction to Computing Science and Programming II

Fall 2023

Week 8: Introduction to Graphs and their Operations

Instructor: Victor Cheung, PhD

School of Computing Science, Simon Fraser University

Fact of the day

Apple does not let bad guys use iPhones in movies

Still from Rian Johnson's "Knives Out." – Lionsgate

Assignment 3

- Assignment 3 is now available on Canvas
  - Read the description file carefully for the questions and submission instructions
- Due on Nov 10, 11:59p
- Learning objectives
  - Organize functions into source code files (include header files, implement in source code files)
  - Create a text-based interactive interface (looping through user inputs)

- DO NOT post the questions or share your code in any platform (e.g., Piazza, Discord, Canvas, Replit …anywhere)
  - Others might use what you post, our similarity report will catch you, both you and copiers get zero for cheating

Recap from Last Lecture

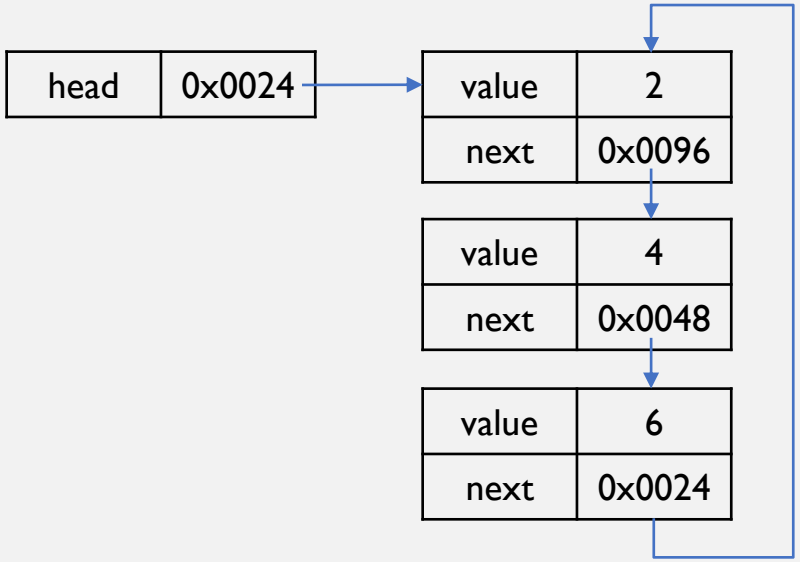
- How to insert/delete a node to/from the inside (not head/tail) of a linked list
  - Cases to consider and be careful with – empty list, list with 1 node, list with 2+ nodes
  - Doubly-linked lists – each node has one more pointer point to the previous node, allows reverse traversal and look up
- Revisiting recursion
  - Can greatly reduce complexity of code, but not always the best option (typically has a longer runtime due to creating a sub-routine in the computer memory stack)
    - Can be replaced by using a Stack ADT

Review from Last Lecture (1)

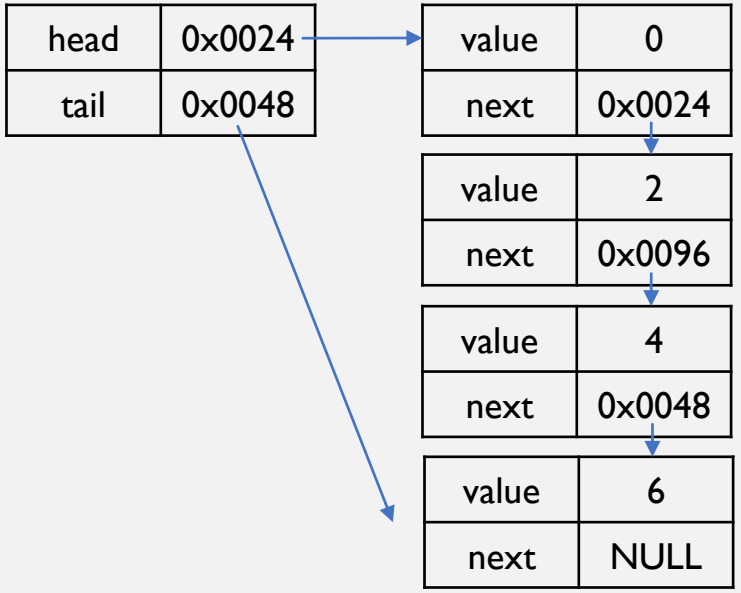
- Implement the linked list data structure
  - General idea: careful with cases when inserting/remove into/from an empty list and non-empty list to determine which pointers need to be updated, use "traveling pointers" as needed
  - Some implementations use node addresses instead of data values, for example:
    - when removing a value, return the node address instead of just the value
    - when searching for a value, use the node address instead of just the value
- Implement the Stack, Queue, Dynamic Array ADTs
  - Choose the correct basic operations to implement the associated policies (e.g., LIFO, FIFO, into/from valid indexes)
  - To store different data types (e.g., double instead of int), change the type the underlying Linked List stores, and the signatures of some of the ADT functions (or use void* to make things generic)

Review from Last Lecture (2)

- We've talked about a variation of linked list being the "doubly-linked list". Look up other variations.
  - Variation 1: Circular-linked list
    Allows traversal of the entire list from any node



Variation 2: Dummy node at the front
Allows a "lag" pointer for easy node removal



Today

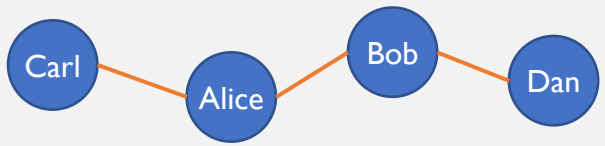
- Graphs
  - A way to describe relationships between data beyond a single order (array/linked list)
    - E.g., social circles, transportation networks
  - Adjacency matrices & adjacency lists

Graphs

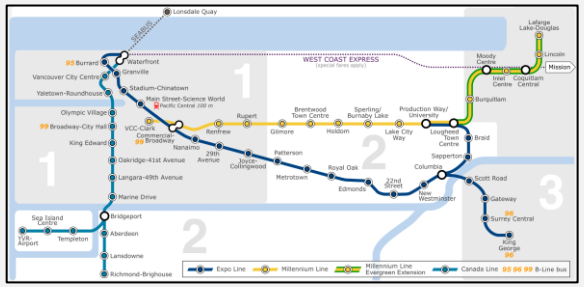
Connecting data points

Graphs

- A **graph** describes relationships amongst items in a collection
  - the items are called **vertices** (represented by dots or junctions)
  - the relations are called **edges** (represented by lines between vertices)
- Some examples:
  - Social circles/networks: Alice is friends with Bob & Carl, Bob is friend with Dan



  - Transit stops



Properties/Terminologies of Graphs

- Path – a sequence of edges which connect a vertex to another vertex
  - A cycle happens if there is a path connected a vertex back to itself
- Connected graph – a graph in which there is a path from any vertex to any other vertex
  - A graph is "disconnected" if there is at least one vertex that cannot be reached from any other vertices
- Complete graph – a graph in which there is an edge between any 2 vertices
- Weighted graph – a graph in which each edge has a value
  - E.g., distance between 2 stops in a transportation network
- Directed graph – a graph in which each edge only describes the relation from one vertex to the other (e.g., A follows B doesn't mean B follows A)
  - A graph is "non-directed" if each edge is bidirectional (e.g., if A is a friend of B then B is a friend of A)

How To Represent Graphs in C?

```
//Adjacency Matrix
#define NUM_OF_VERTICES

int adjMatrix[NUM_OF_VERTICES][NUM_OF_VERTICES];

//no edge between vertex 0 and vertex 1
adjMatrix[0][1] = 0;
adjMatrix[1][0] = 0;

//edge between vertex 2 and vertex 3
adjMatrix[2][3] = 1;
adjMatrix[3][2] = 1;
```

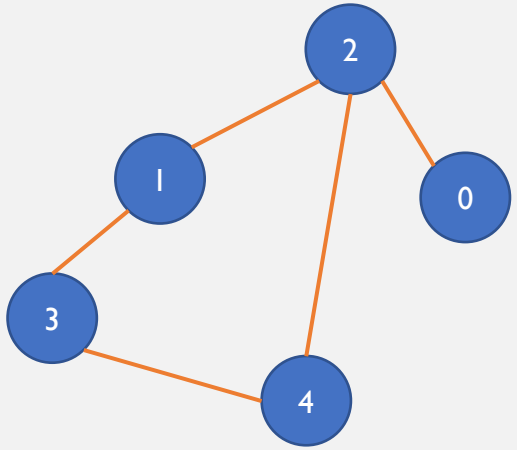
```
//Adjacency List
#define NUM_OF_VERTICES

//LList is a linked list
LList adjList[NUM_OF_VERTICES];

//no edge between vertex 0 and vertex 1
//do nothing

//edge between vertex 2 and vertex 3
insertEnd(adjList[2], 3);
insertEnd(adjList[3], 2);
```

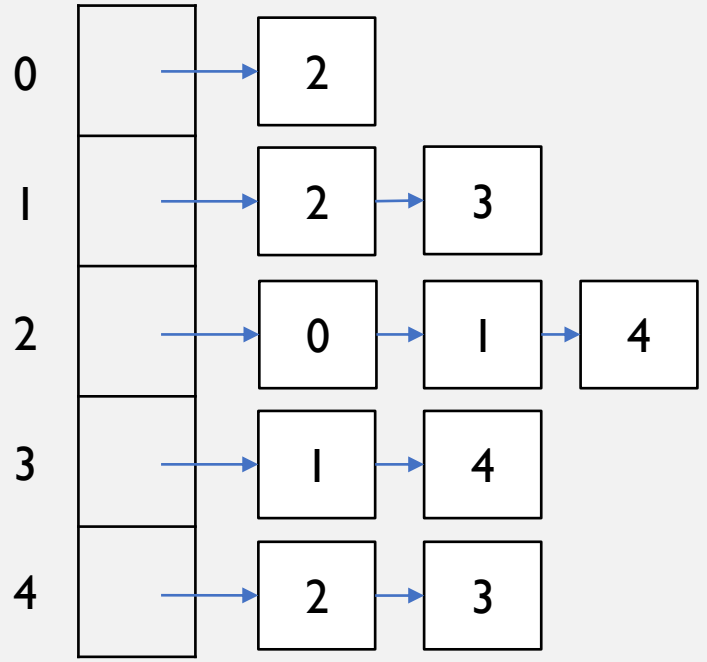
- There are algorithms that tells if a graph is connected or not, finds the shortest path from one vertex to the other, search for a vertex, print content of all vertices…etc.
  - Out of scope in this course, but will be in CMPT 225

Representing Graphs

	0	1	2	3	4
0	0	0	1	0	0
1	0	0	1	1	0
2	1	1	0	0	1
3	0	1	0	0	1
4	0	0	1	1	0

Adjacency Matrix





Adjacency List

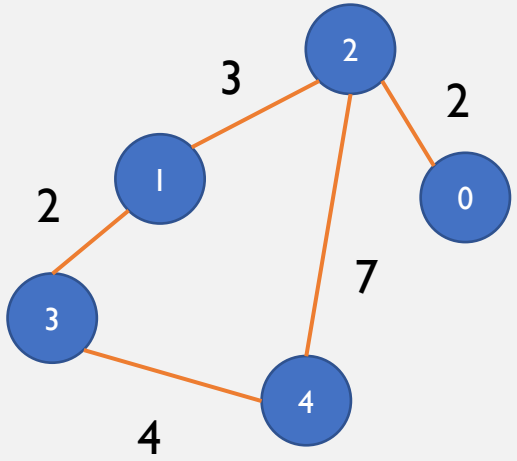
What about Weighted Graphs?

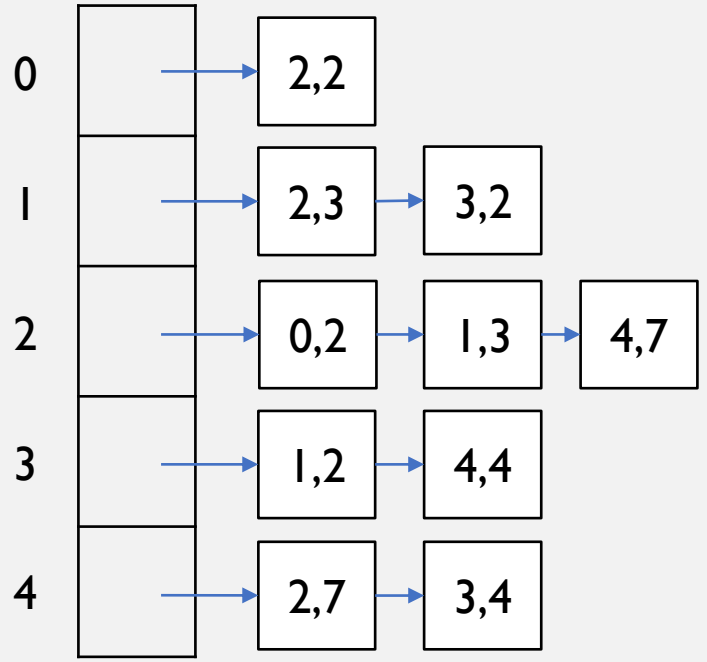
- In a weighted graph each edge has a value, for example:
  - Distance between 2 stops in a transportation network
  - Time of travel between 2 airports in a flight map
- How to represent weights in an adjacency matrix?
- How to represent weights in an adjacency list?

Representing Weighted Graphs

	0	1	2	3	4
0	0	0	2	0	0
1	0	0	3	2	0
2	2	3	0	0	7
3	0	2	0	0	4
4	0	0	7	4	0

Adjacency Matrix





Adjacency List

Today's Review

- Graphs
  - A way to store relationships between data beyond a single order (array/linked list) – represented by vertices & edges
    - E.g., social circles, transportation networks
  - Adjacency matrices & adjacency lists – implemented by 2D arrays & Linked Lists

Homework!

- We've learned that in a complete graph there is an edge between every 2 vertices. How many edges are there if we have a graph with 3 vertices? How about 4 vertices? Is there a formula to calculate that?
- Complete the code in p11 to represent the graph shown in p12

Midterm Prep

Midterm on Nov 3 during Lecture (D100: 12:30p – 2:20p)

- 1 hour and 30 minutes
  (if you requested accommodation via CAL, contact them to arrange writing the exam at their facilities)
  - 3 questions each with several items (short questions & answers, coding questions)
- In-person (come on time and spread across the classroom, we might ask you to change seats)
  - Closed books. No electronics. Only pens/pencils are allowed
  - Bring your student IDs. Keep everything else in your bag. We are not responsible for your belongings
  - Includes all the materials we learned so far (not including graphs)
  - All work must be done by yourself (we might interview you later if we find anything suspicious)
- Instructor/TAs will be available for questions
  - Generally will not answer any clarification questions (write down your assumptions), only respond to typos

Tips for Preparing for Midterm

- Anything that are covered up to and including Week 08 could appear in the midterm
- Revise lecture materials (including the exercises/homework), do the weekly self-tests, and study the materials posted in our Canvas course website (e.g., assignments, practice problems, past exams)
- Don't try to "predict" what the questions will be from the past exams
  - Instead, learn about the format and what is expected in the answers
- Read the questions carefully… for example:
  - If it says you cannot use functions from a library, you cannot
  - If it asks you to show steps, you need to show them