Sliding Window Algorithms and Graph Theory

May 23, 2016

1 Sliding Window Algorithms

1.1 What is it?

Generally speaking a sliding window is a sub-list that runs over an underlying collection. Let's say you have an array like this:

[a b c d e f g h]

A sliding window of size 3 would run over it like this:

[a b c]
[b c d]
[c d e]
[d e f]
[e f g]
[f g h]

1.2 Methods to Solve

Let's imagine you are an alien in a UFO and below you, there are a row of *n* evenly spaced houses with residents living inside each house. You want to suck up as many residents as possible, but the technology you built to suck up the residents had bad design plans so you can only suck up the residents in *k* adjacent houses and you can only suck residents up once.

1.2.1 Brute Force

We can use brute force to add all the numbers in each sliding window and compare them to find the largest sum. However the overall time complexity is *O(nk)* which is not the most time-efficient method.

1.2.2 Store the Sum of Previously Calculated Numbers

Because the sum of *k-1* numbers has already been calculated for each sliding window, we can store the sum of them somewhere and then we can just add on one number each time. However, this is not a very memory-efficient method.

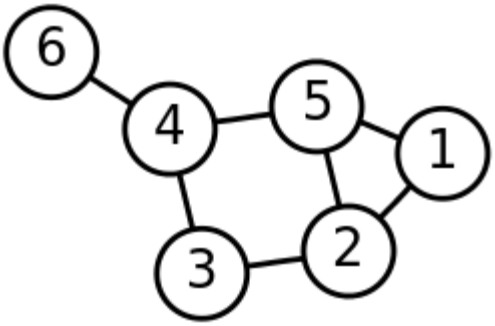
1.2.3 Subtract a Number and Add a Number

We can subtract the first number in the previous window and add the next number, to get the sum in each sliding window. The overall time complexity is of this method is *O(n)*.

2 Graph Theory

2.1 What is it?

A graph in the context of computer science is made up of *vertices*, *nodes* or *points* which are connected by *edges*, *arcs* or *lines*. A graph can be *undirected* which means that there is no distinction between the two vertices connected by each edge, or it can be *directed* which means that the edges point in a certain direction. This is an example of a simple undirected graph:



2.1.1 Weighted Graphs

Graphs can also be *weighted*; in a weighted graph, an edge is assigned a number, which can mean different things in different graphs. If we pretend that the nodes represent cities and we have an edge between the two nodes, the number can represent the amount of time it takes to travel from one city to another.

2.2 Graph Traversal

We need to be able to traverse graphs in a systematic order, in instances where we need to find whether a particular vertex is connected to another vertex.

2.2.1 DFS

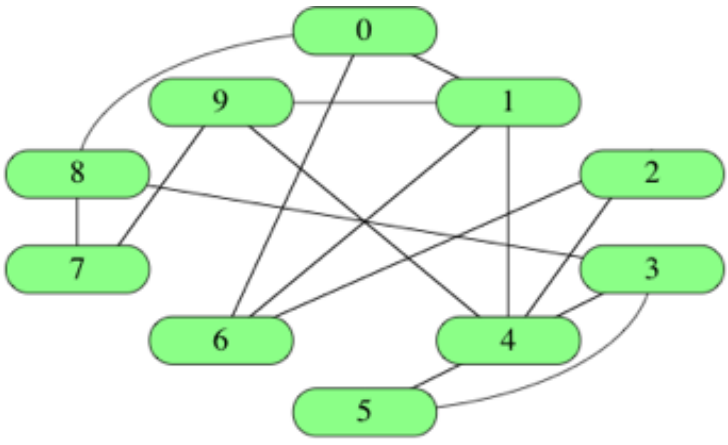
In a depth first search, we go from one vertex to an adjacent vertex which we have not visited yet. If we have visited all the adjacent vertices, we backtrack until we land on a vertex which has an adjacent vertex we have not yet visited. This continues until we have traversed through the whole graph. This follows a stack-like (FILO) structure. To prevent infinite loops, we need to make sure we visit each vertex only once.

2.2.2 BFS

This is a breadth first search. I don't think we talked about what this is, except Julian asked something like "if we use stacks for DFS, what do you think we use for BFS?" but I don't think he actually answered the question.

2.3 How is it represented?

Let's say we wanted to represent the graph below:



2.3.1 Edge List

One method to represent a graph is as an array of edges, which we call an edge list (who would've guessed). To represent an edge, we have an array of the two numbers on the vertex:

[[0, 1], [0, 6], [0, 8], [1, 4], [1, 6], [1, 9], [2, 4], [2, 6],

[3, 4], [3, 5], [3, 8], [4, 5], [4, 9], [7, 8], [7, 9]]

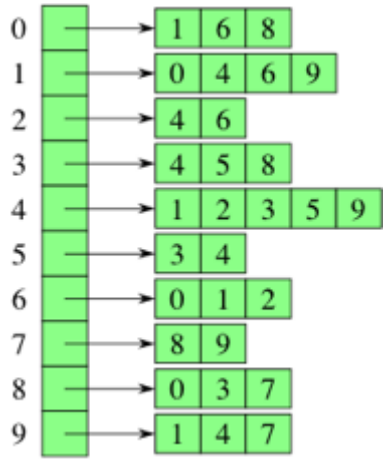
2.3.2 Adjacency Matrix

For a graph with V vertices, an adjacency matrix is a V x V matrix of 0s and 1s (TRUE and FALSE), for whether there is an edge between the two vertices. If we want to indicate an edge weight, we can use the edge weight for TRUE and use another value if there is no edge (i.e. -1 or NULL).

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1
2	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0	1	0
4	0	1	1	1	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	0

2.3.3 Adjacency List

In an adjacency list, we store an array, for each vertex, of the vertices adjacent to it.



We can get to each vertex's adjacency list in constant time, because we just have to index into an array.