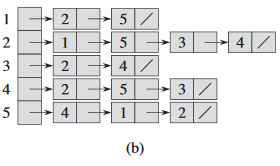
**What does it mean to search a graph?**

It means you trace the edges of it to find the vertices (Points where multiple lines/curves meet). Methods for doing this consist of using input graphs to analyse the nature of the output graphs, or just through other methods.

**How do we represent these graphs?**

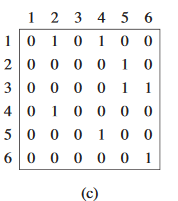
Assuming they follow the format (Don’t ask what that means), they can either be represented as adjacency lists or adjacency matrices. Lists are generally used as they are compact (I guess in memory?), and function well with sparse graphs. If the graph is dense however, matrices are recommended. The difference between sparse and dense graphs is based on E’s relation to V2. The closer E is to V2, the denser the graph is.

**More on the lists please**

In list representations, you have an array *Adj* of lists which size is determined by |V|. Now assume we have some variable u, then the array Adj[u] contains all the vertices v that fulfil the condition . To make it easier to understand, look at the image to the right. There, a bunch of arrays are listed in an orderly fashion. The first number in each array correspond to u, and all the following numbers are the ones that u can go to / become / are adjacent to.

In the book on figure 22.1(a) and 22.2(b), are respectively examples of undirected graphs and directed graphs. If G is a directed graph, the sum of the lengths of all adjacency lists (the arrays in the image above) will equal |E|. If the graph is undirected however, then it will be 2|E| instead.

Fun fact: the memory cost of an adjacency list representation is always going to be .

**Now gimme matrices!**

They are 2 dimensional arrays, which uses even more memory. Instead of having a set of number representing adjacency, this method instead uses a coordinate representation. If you want to check if a value v is adjacent to a value u, then you check if the vth value on the uth row is a 1. This can be seen in the image of a directed graph matrix to the right. Mind you, if the graph was undirected, you would also be able to tell the adjacency through the uth value on the vth row.

Fun fact: Unlike the list representation, the memory cost of a matrix is always .

**Weighted graphs**

So this is just a side note, but you can actually use a variety of functions to change adjacency, one of which being the weighted function. It’s pretty simple, you have some function w which takes inputs u and v. When calculating the adjacency of v, you use the function w to alter v, which can alter whether it is adjacent. At least, I think that is how it works, the book doesn’t delve too heavily into it, so it does kind of sound like arcane bullshit.

**Breadth-first search**