Chapter 8
Data Structures

Computer has to process lots and lots of data. To systematically process those data efficiently, those data are organized as a whole, appropriate for the application, called a *data structure.*

The important issue is that different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.

An example

When we talked about algorithms, we looked at the example of 555-1212, where the phone records are organized as tables, i.e., a list of phone records.

We also saw that if we use a table where the phone records are not alphabetically sorted, only the sequential search algorithm can be applied. On the other hand, if those records are sorted, then the much faster binary search approach can be followed.

One of the two

It is long recognized that there is a close relationship between algorithms and data structure, indeed, as stated in Worth's famous book,

> "Algorithms + Data Structures = Programs"

Hence, data structures are used in every non-trivial program or software system. In many cases, an efficient algorithm is based on a cleverly designed data structure. An appropriate data structure is also the key to manage huge amounts of data, such as large databases, the bloodline of today's information age.

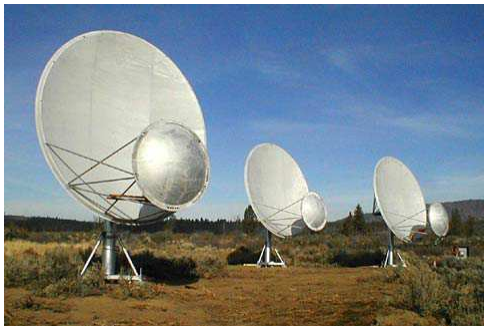
We now look at several simple, but often used, data structures.

Array

As we already discussed, the backbone of the 555-1212 system is a table of phone records. Technically, we call such a data structure as an *array* of data. Array is perhaps one of the most used data structure.

This data structure is also used in our daily life.

The following figure shows an array of telescopes.

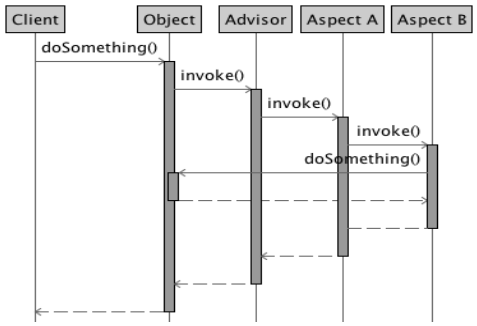


Stack

We once discussed in the topic of *Analytical Engine* about the concept of *sub-routine,* a group of steps that can be used as a whole.

For example, when we want to "doSomething", we will use the "Advisor", which uses "Aspect A", then "Aspect B", which again "doSomething" before it completes.

When "Aspect B" completes, we continue run "Aspect A", then return to the "Advisor", finally come back to "doSomething" until it is over.



Question: What is buried here?

Answer: The issue here is that we take off from "doSomething" first, but will come back to this as the last step.

Similarly, we come to the "Aspect B" as the last step of the whole sequence, but finishes it off as the first one. When we use a data structure to manage such a sequence of *FirstOut-LastIn* or *LastInFirstOut,* we will use the *Stack* data structure.

Stack is actually a restricted version of a *list,* but when we either add in something into a Stack, or remove something from it, we always do it in only one of the two ends.

A general list represents a sequence where we can either add something in, or take something out, *anywhere.*

No just here....

Stack is also used in our daily life.

For example, when we want to take a dish out of a deck of plates, as shown in the following figure, which one do you take?

Although you can take out any one, I bet you must take the one at the top. On the other hand, when we want to put a dish back to such a deck, where do you put it back to? Again at the top.

Thus, such a deck of dish is really a stack.

Queue

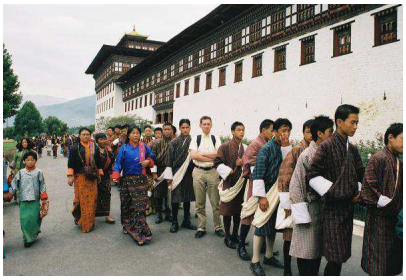
Queue is another very simple but useful data structure. In a computer lab when one printer is shared by all the computers in that lab, what happens if five people want to print out their papers? The answer is very simple: they take in turns. What happens is that those papers will be put into a queue according to the order they arrive, then wait there until it comes to the front.

Queue is also a restricted list where we add things in one end, and remove from the other. It is actually called *FirstInFirstOut* or *LastIn-LastOut* list.

Not just here again...

Queue is certainly also used quite a bit in our daily life.

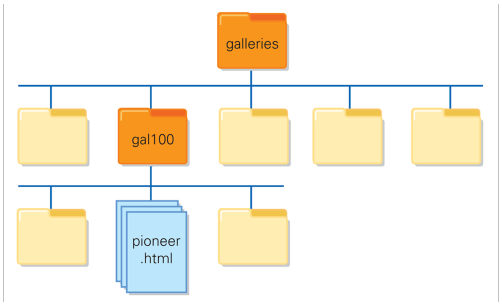
For example, in a civilized society, we would wait in a line for our turns, but not cut in as some of the students did when buying tickets for the Drake show in last year's Spring Fling weekend.



Tree

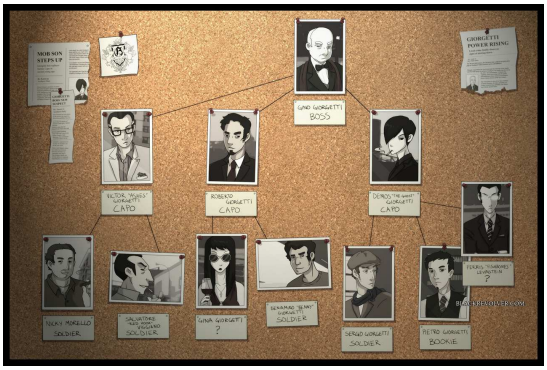
Array, stack and queue are examples of linear data structures, where everything is lined up. Tree is an example of a more complicated data structure.

Indeed, everything inside the computer is organized as a tree. For example, the following figure shows the composition of a web site: The folder *galleries* contains a few folders, one of which is called *gal100,* which contains a few folders and a few documents, one being *pioneer.html.*



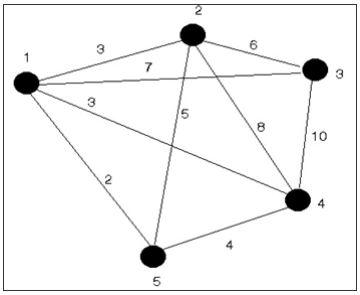
A key property of using the tree structure to organize such data is that we can provide a "unique path" for the computer to find the *pioneer.html* file.

Tree structure is also widely used in our life. For example, the following figure shows the relationship among persons in a family.



Graph

Graph is the most complicated data structure. The following shows a simple graph that connects five vertices with nice edges.



In the above graph, we represent the association between various vertices as well as some information attached to such association.

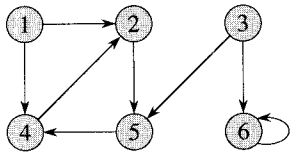
Graph representation

Usually, a graph can be represented as either an *adjacency matrix,* or an *adjacency list.*

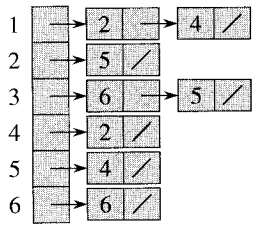
The adjacency matrix of an $n-$vertex directed graph, G, is an $n \times n$ matrix, such that,

$$A(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

An alternative approach is to use a linked list to represent, for every vertex, all the vertices it is adjacent.



(a)



(b)

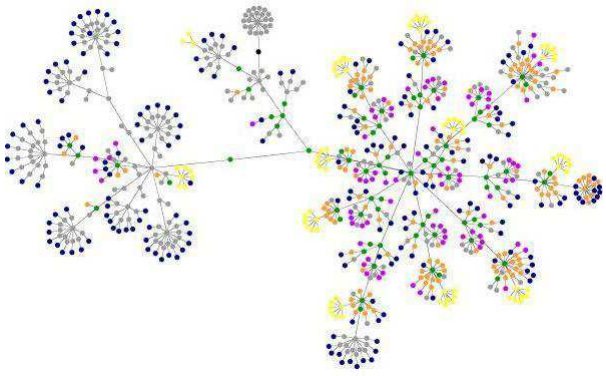
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Applications

The graph data structure is used in many real applications. For example, before you do any travel, you want to google a shortest way to get there, where a map is always represented as a graph.

Internet is also organized as a graph.

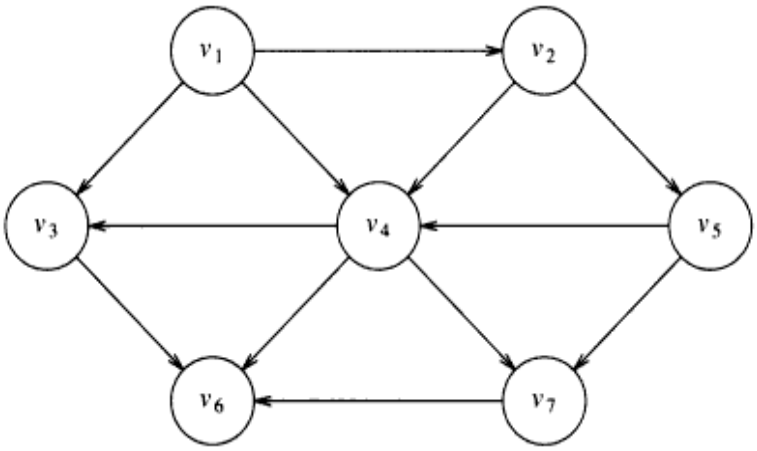


Whatever we do with the computer network, we have to rely on this graph structure to come up with algorithms, such as *topological sort*, *shortest paths*, *minimum spanning tree*, *maximum network flow*, etc..

Course registration

It is about time for us to register courses for Spring 2011. How could we make sure that all the prerequisite courses will be taken first?

For examples, the following graph represents the prerequisite structure of part of a curriculum.



Course v_1 is a prerequisite of both v_2 and v_4, and v_1, v_2 and v_5 are all prerequisites of v_4, etc..

Topological sort

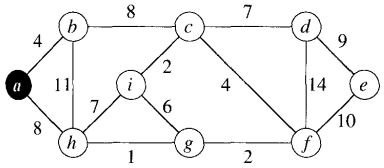
The following is an algorithm looking for a *topological sorting* in G, where we label all the vertices to come up with an order in which these courses will be taken:

```
1. Set n to 1
2. Wile not all vertices labeled yet
3.  Find v, s.t. there is no edge coming into v;
4.  Label v with n;
5.  n<-n+1
6.  Retract v together with all the edges
     outgoing from v;
```

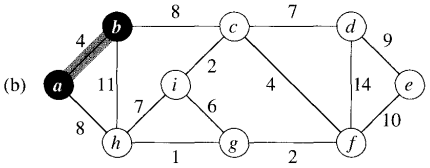
Let's apply this algorithm to the above graph.

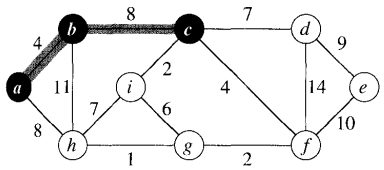
Minimum spanning tree

Let's assume that we want to hook up a collection of sites, houses, buildings on a campus, with Internet lines, each connection comes at different cost. How could we do it with minimum cost?

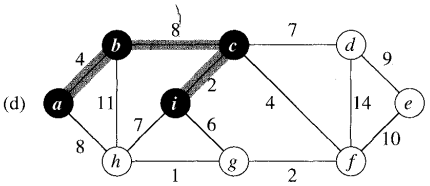


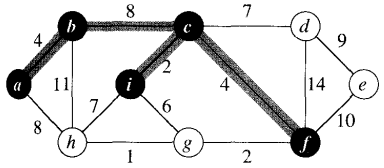
(b)



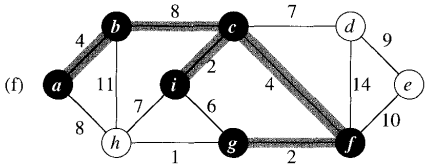


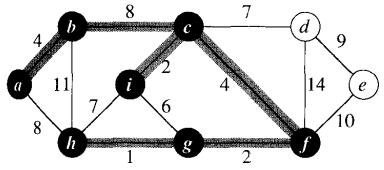
(d)



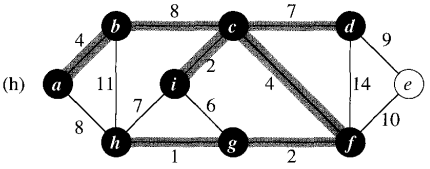


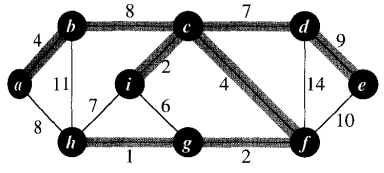
(f)





(h)





Activities

1. Find out other examples of applications of the data structures that we have discussed here, i.e., stack, queue, list, tree, graph, etc., in our lives.

2. Find out examples and applications of other data structures that people have been using with computers applications.

3. How do you organize data, in the most general sense, in your life? For example, how do you organize all the clothing in your closet so that it will take you little time to find out the one that you want?