Introduction to Data Structures

Aniel Nieves-González

Institute of Statistics

Spring 2014

Introduction

- Computer programs operate over “tables” of information.
- The information isn’t stored as an amorphous mass of numerical values, but they involve important structural relationships between the data elements.
- Some algorithms to solve specific problems (e.g. sorting) depend on the structural relation between the data elements, i.e. on how the data elements are stored.
- Each data stuctures is asociated to a series of operations over the given structure.
- We’ll talk about some fundamental data structures that are present either explicitly or implicitly in modern computer systems and high-level programming languages.

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.
2. Typical operations over the linear list are:

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.
2. Typical operations over the linear list are:
    1. Gain access to the kth node.

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.
2. Typical operations over the linear list are:
    1. Gain access to the kth node.
    2. Delete kth node.

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.
2. Typical operations over the linear list are:
    1. Gain access to the kth node.
    2. Delete kth node.
    3. Insert a node just before or after the kth node.

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.
2. Typical operations over the linear list are:
   1. Gain access to the kth node.
   2. Delete kth node.
   3. Insert a node just before or after the kth node.
3. In general linear list may or may not be stored consecutively inside the computer system. For example consider one-dimensional array vs. a linked list (more on this later).

Linear lists

1. A *linear list* is a sequence of $n \geq 0$ nodes $X[1], X[2], \ldots, X[n]$.
2. Typical operations over the linear list are:
    1. Gain access to the kth node.
    2. Delete kth node.
    3. Insert a node just before or after the kth node.
3. In general linear list may or may not be stored consecutively inside the computer system. For example consider one-dimensional array vs. a linked list (more on this later).
4. The item at each node may or may not be of the same type (more on this later).

Linear lists

There are linear list in which the typical operations occur at the first or last node. They occur so frequently in applications that they have special names.

- A *stack* is a linear list in which all typical operations occur at one end of the list (last-in-first-out, LIFO).

Linear lists

There are linear list in which the typical operations occur at the first or last node. They occur so frequently in applications that they have special names.

- A *stack* is a linear list in which all typical operations occur at one end of the list (last-in-first-out, LIFO).
- A *queue* is a linear list in which all insertions are made at one end of the list; all access and deletions at the other end (first-in-first-out, FIFO).

Linear lists

There are linear list in which the typical operations occur at the first or last node. They occur so frequently in applications that they have special names.

- A *stack* is a linear list in which all typical operations occur at one end of the list (last-in-first-out, LIFO).
- A *queue* is a linear list in which all insertions are made at one end of the list; all access and deletions at the other end (first-in-first-out, FIFO).
- A *dequeue* ("double-ended queue") is a linear list in which all typical operations occur at the ends of the list.

Linear lists: sequential allocation I

The simplest way to store a linear list inside a computer is to store the nodes in consecutive locations, one node after another. This is sequential allocation. Moreover, we can also restrict our linear list to store elements of the same kind (type). This lead us to:

- A *one-dimensional array* is a linear list in which the allocation of the elements is sequential and all elements are of the same type. For example: let X be a list and $j \in \mathbb{N}$

  $$\text{address}(X[j+1]) = \text{address}(X[j]) + c$$
  $$\text{address}(X[j]) = L_0 + cj$$

  where c is the number of bytes per node and L_0 is a constant base address (starting address).

Linear lists: sequential allocation II

- Mathematically a one-dimensional array can be view as an n-tuple.

The idea of a one dimensional array can be generalized to n dimensions.

- A 2-dimensional array is like a matrix. For example the $m \times n$ matrix A

$$A = \begin{pmatrix} A[1,1] & A[1,2] & \dots & A[1,n] \\ A[2,1] & A[2,2] & \dots & A[2,n] \\ \vdots & \vdots & & \vdots \\ A[m,1] & A[m,2] & \dots & A[m,n] \end{pmatrix}$$

- Observe that each node $A[i,j]$ belongs to two linear lists.

Linear lists: sequential allocation III

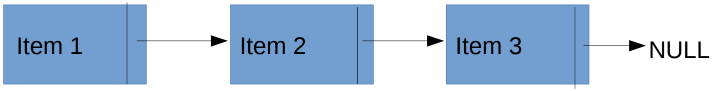
- The sequential allocation scheme for a 2-dimensional array is analog to the scheme for a one-dimensional array albeit more complicated. for example:

  $$\text{address}(A[i,j]) = a_0 + a_1 i + a_2 j$$

  where a_0, a_1, and a_2 are constants.
- Observe that although the sequetial allocation is easier to understand it present some challenges to the machine (operating system) at the moment of defining arrays and doing data transfers. What is the main challenge?

Linear lists: Linked allocation

Instead of keeping a linear list in sequential memory locations, we can use a more flexible scheme in which each node contains a datum and a link to the next node of the list. For example a simple linked list is:



There are more complicated variations such as double linked lists, circular linked lists etc.

Nonlinear Data Structures: Trees

The tree is an important nonlinear data structures that arise in computer algorithms. A tree structure means that there is a branching relationship between nodes. Formally, we define a tree as follows.

Definition (Tree)

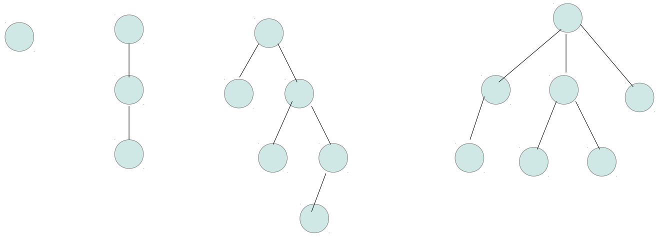
Let T be a nonempty finite set of nodes such that:

1. *There is a specially designated node called the root of the tree, root(T).*
2. *The remaining nodes (excluding the root) are partitioned into $m \geq 0$ disjoint sets $T_1, \ldots, T_m$, and each of these sets is in turn a tree. The trees $T_1, \ldots, T_m$ are called subtrees of the root.*

Observe that the definition is recursive (more on this later).

Nonlinear Data Structures: Trees

A few example of trees are:



Nonlinear Data Structures: Trees I

Observe the following:

1. The number of subtrees of a node is the degree of that *node*.
2. A node with degree zero is called a *leaf*.
3. Each root is said to be the *parent* of the roots of its subtrees. The roots of such subtrees are called *siblings*. The siblings are the *children* of their parent.
4. Tree structures can be represented graphically in other ways that bear no resemblance to actual trees.
5. In a *binary tree* each node has at most two subtrees and when only one subtree is present we distinguish between left and right subtree.

Nonlinear Data Structures: Trees II

6. An equivalent definition of tree uses terminology from graph theory.
7. Many algorithms make use of the tree data structure to solve specific problems, in particular, sorting. For example the following sorting algorithms have the virtue that are efficient in terms of running time:
   1. Binary tree sort.
   2. Heap sort.
   3. Quick sort.

An heterogeneous container: “the struct”

With the exception of the n-dimensional array the items contained in the above mentioned data structures may or may not be of the same type.

- By definition, the elements n-dimensional array are of the same type.
- In many programming languages it is possible to define “containers” that hold items of different type.
  - In C and MATLAB the struc can hold data of different type.
  - In MATLAB the cell array can hold data of different type.
  - In R the list and data frame can hold data of different type.

Basic Data structures in R I

All objects in R have some basic data types, or modes in R parlance, which are: numeric, character, logical, list, and function. These modes or types indicate how the object is stored in memory. The *mode* function give us that information. For example, mode(5) should return numeric as the result. mode('5') should return...
Here a some of the basic data structures that are defined in R.

1. **Vectors.**
   - Vectors are homogeneous, i.e., all elements have the same mode. Moreover vectors are indexed by position as one-dimensional arrays. For example $v[3]$ returns third element of v.
   - Vectors can be indexed by multiple position, returning a vector. For example $v[2:3]$ and $v[c(2,3)]$ return 2nd and 3rd element of v.

Basic Data structures in R II

- Vector elements can be named using the name function and then the elements can be selected by name. As an exercise issue the following commands: $x = c(2, 3, 4)$ names$(x) = c("A", "B", "C")$
- Constants and atomic variables or scalars are viewed in R as vectors with one element.

2. **Lists** (the analog of lists in other programming languages are hash or lookup tables).
   - Lists are heterogeneous, i.e., list elements can be of different types. Moreover lists can contain lists and other structured objects such as data frames.
   - Lists can be indexed by position or multiple position akin to vectors. For example $L[[2]]$ refers to the 2nd element of list L. Observe the double brackets.

Basic Data structures in R III

- Similar to vectors, lists elements can be named. Thus, L[[Tito]] and L\$Tito both refer to an element named "Tito".

3. **Matrices**: In R a matrix is a vector that has the attribute of dim set from NULL to some numeric value. For example, the instructions $A = 1:4$, $\dim(A) = c(2,2)$ returns

$$\begin{array}{cc} 1 & 3 \\ 2 & 4 \end{array}$$

one can also use the function *matrix* to create a matrix. Matrices can also be constructed from lists.

4. n-dimensional arrays. They are a generalization of matrices. As an exercise issue the two commands: $B = 1:12$ and $\dim = c(2,3,2)$.

Basic Data structures in R IV

5. **Data frame**: Data frames are intended to mimic a data set. It is a rectangular data structure but not implemented with matrices but it is rather a list. The function “data.frame” can be used to create data frames.
   - A data frame is a list whose elements are vectors. Those vectors are the columns of the data frame.
   - The vectors (columns) must have the same length.
   - The columns must have names.
   - One can use list operations such as Dframe[[i]] and Dframe$Tito to extract columns.
   - One can use matrix-like notation for accessing content: Dframe[i, j], Dframe[$i,$], or Dframe[$, j$].
6. Factors are another data structure (more on this later).

References I

- Donald D. Knuth.
  The art of computer programming. Volume 1: Fundamental Algorithms.
  Third edition. Addison-Wesley, 1997.

- Paul Teetor
  R cookbook
  O'Reilly, First edition 2011.