Chapter 11

Algorithms

Def.:

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

Deterministic and non-deterministic algorithm

Deterministic algorithms can be defined in terms of a state machine: a *state* describes what a machine is doing at a particular instant in time. State machines pass in a discrete manner from one state to another. Just after we enter the input, the machine is in its *initial state* or *start state*. If the machine is deterministic, this means that from this point onwards, its current state determines what its next state will be; its course through the set of states is predetermined. Note that a machine can be deterministic and still never stop or finish, and therefore fail to deliver a result.

A non-deterministic algorithm is one in which for a given input instance each intermediate step has one or more possibilities. This means that there may be more than one path from which the algorithm may arbitrarily choose one.

What makes the algorithms non-deterministic?

- If it uses external state other than the input, such as user input, a global variable, a hardware timer value, a random value, or stored disk data.
- If it operates in a way that is timing-sensitive, for example if it has multiple processors writing to the same data at the same time. In this case, the precise order in which each processor writes its data will affect the result.
- If a hardware error causes its state to change in an unexpected way.

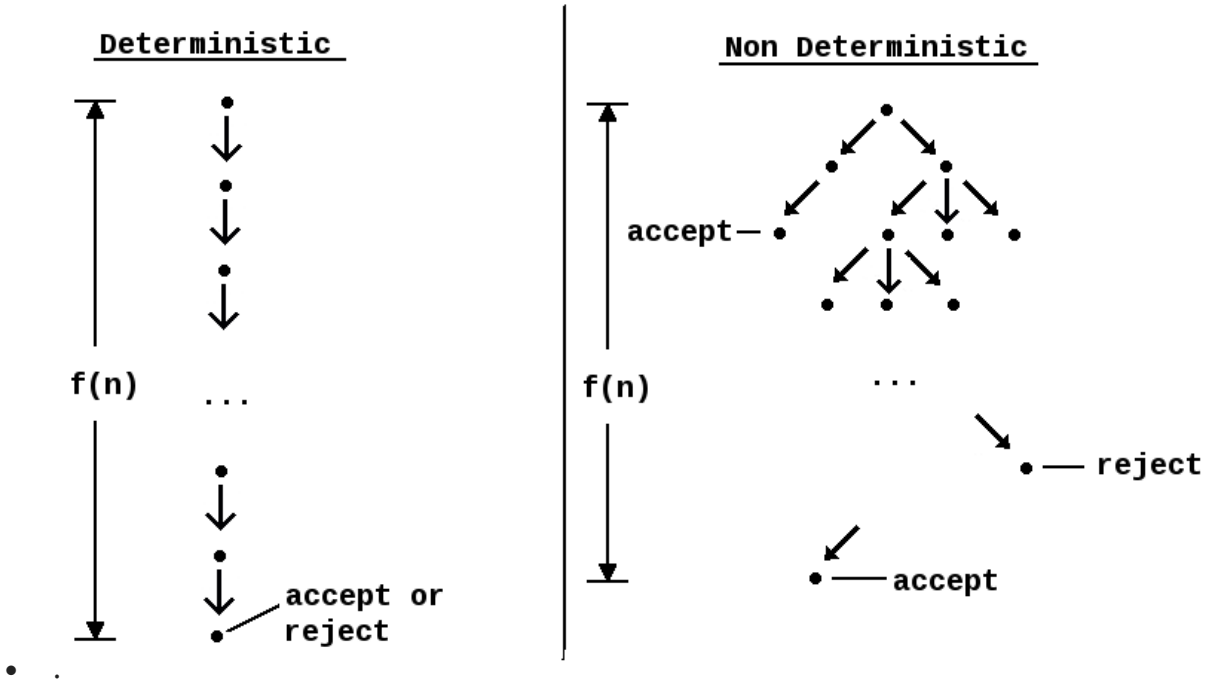

Fig. Deterministic vs. Non deterministic

Divide & Conquer algorithm

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. It involves three steps:

Divide a problem into smaller sub-problems

Conquer the sub problems by solving them recursively.

Merge the answers together so as to obtain the answer to the larger problem

Some examples are:

1. Thinking the solution of tower or Hanoi with 'n' disks as; one task to move n-1 disks to the temporary peg, move 'nth' disk to destination, and again move n-1 disks to the destination.
2. Merge sort: Dividing a file into sub files and then sort the sub files. After sorting sub-files, merge them.
3. Traversing only one half of the given range in the binary search tree.

Advantages:

- **Parallelism:** If an algorithm can be made as divide and conquer, the sub-problems can be passed to other machines. Other machines work on the sub-problems and send result back to the main machine. The job of main machine is to divide the problems into sub-problems and assemble the result. Some of the problems that we applied this technique are binary search, binary tree search, merge sort etc
- **Memory access**: Divide-and-conquer algorithms naturally tend to make efficient use of memory caches. The reason is that once a sub-problem is small enough, it and all its sub-problems can, in principle, be solved within the cache, without accessing the slower main memory.

Disadvantages:

1. This approach uses recursion which makes the process little slow.
2. Another problem of a divide-and-conquer approach is that, for simple problems, it may be more complicated than an iterative approach, especially if large base cases are to be implemented for performance reasons. For example, to add *N* numbers, a simple loop to add them up in sequence is much easier to code than a divide-and-conquer approach that breaks the set of numbers into two halves, adds them recursively, and then adds the sums.

Series & parallel algorithm

- Algorithms are usually discussed with the assumption that computers execute one instruction of an algorithm at a time. This is a serial algorithm, as opposed to parallel algorithms, which take advantage of computer architectures to process several instructions at once. They divide the problem into sub-problems and pass them to several processors. Iterative algorithms are generally parallelizable. Sorting algorithms can be parallelized efficiently.

Heuristic & approximate algorithm

- The term **heuristic** is used for algorithms which find solutions among all possible ones, but they do not guarantee that the best will be found, therefore they may be considered as approximately and not accurate algorithms.

- These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this best solution is proven to be the best.
- The method used from a heuristic algorithm is one of the known methods, such as greediness, but in order to be easy and fast the algorithm ignores or even suppresses some of the problem's demands.
- Approximation algorithms are efficient algorithms that find approximate solutions to NP-hard optimization problems with provable guarantees on the distance of the returned solution to the optimal one.
- Approximation algorithms naturally arise in the field of theoretical computer science as a consequence of the widely believed P ≠ NP conjecture.

Big O Notation

- Big-O Notation measures the complexity of an algorithm or order of magnitude of the number of operations required to perform a function.
- In other words, how the number of processing steps increase as the number of items being processed increases. Processing may not increase in a constant manner. For instance if processing one item takes 2 seconds, processing 100 item does not necessarily take 200 seconds (2 * 100).
- Big-O notation is a type of language or notation or terminology to discuss the the reality which is - algorithms need to process efficiently. Without knowing Big-O notation, an experienced programmer can look at an algorithm and understand that a step that can be moved to process outside of a loop will reduce the processing steps by the number of loops the algorithm has to perform less one. Instead of processing once for each item in the list, it can process that step only one time. For me, this underlying understanding of processing steps is more practical than reciting Big-O Notation, but we must succumb to industry standards when applying for jobs, and to communicate about complexity of an algorithm with other programmers. So I conform.

Common Big-O notation orders:

O(1) represents function that runs in constant time

The time to run an algorithm doesn't change based if the number of items being processed changes. Whether running 1 or 1000 items the time remains constant. This is rare. Looking up an element in an array usually takes the same amount of time no matter how many items are in the array and would be said to run in constant time.

O(N) represents function that runs in linear time

Operations to run directly proportional to number of items processed. For instance if it take 3 minutes to process 1 item it will take 3 X 10 = 30 minutes to process 10 items.

O(N^2) represents a function that runs in quadratic time.

The equation for quadratic time is (N^2 - N) / 2. Or in other words 0+1+2+...+(N-1). In Big-O notation constants are dropped so we have N^2 - N. As the number of items increases, subtracting N from the result becomes a negligible difference so we can skip that and end up with N^2. So if you have 2 items a function that runs in O(N^2) roughly takes 4 processing steps and with 10 items takes 100 processing steps. An example would be inserting items being processed into an array in the first position and having to move every single item of the array each time to insert the new item. Functions running in quadratic time are typically not acceptable for interactive applications.

O(log N) and O(N log N) represent a functions that runs in logarithmic time.

The running time of an algorithm increases with the log of the number of items being processed. These generally mean that the algorithm deals with a data set that is partitioned into small groups of data as it is processed, like a balanced binary tree.

For instance if your asked to find the number I'm thinking of out of 100 you could ask, is it greater than or less than 50. I say greater. You say is it greater or less than 75. I say greater. You say is it greater or less than 87.5. I say greater...and continues until you get to the number I am thinking of which is 88. This is more efficient than saying, "Is it one? Is it two? Is it three?..." etc.

Questions to emphasis

1. Write short notes:
    - Big O notation
    - Divide and conquer algorithm
    - Serial & parallel algorithm
    - Divide & Conquer algorithm