Design & Analysis of Algorithms

20MCACC303-DAA

Unit-3: Greedy Methods & Dynamic Programming

Greedy Methods

Dynamic Programming

Greedy methods

The problem which are solved using greedy method is of following structure:

- There are ***'n' inputs*** of some values and ***objective functions***.
- The method gives an optimal solution to the problem by considering the ***inputs 1 (one) at a time***, checking to see if it can be included in the set of values which give an ***optimal solution***, and than check if it is a ***feasible solution***.
- All of 'n' inputs may ***not be included***, only those needed to form the ***optimal solution will be included***.
- Each input may consume some ***resources*** which is generally available in ***limited quantity***.

Greedy methods…

The flow of data is as in the figure below:

- Greedy method depends upon ***local maximum***.
- The strategy adopted for achieving the ***optimal solutions*** is called the ***"Greedy Methods".***
- The word greedy refers to ***allocating the maximum possible*** values of some limited resource to the first element which enter the optimal solution.
- Feasibility of solution means, ***to find solution by obeying the constraints***.

Greedy methods…

Below methods come under the head of Greedy methods:

1. Knapsack problem
2. Job sequencing with deadline
3. Minimum spanning tree
4. Shortest path method

1). Knapsack problem

- Given a set of items, each with a weight and a value, determine the number of each item to include in a ***collection (Sack)*** so that the total weight is ***less than or equal to a given limit*** and the total ***value (profit)*** is ***as large as possible***.
- The problem often arises in ***resource allocation*** where the decision makers have to choose from a set of ***non-divisible projects*** or tasks under a ***fixed budget*** or ***time constraint***, respectively.
- The problem refers to the commonplace problem of ***packing the most valuable or useful items*** without overloading the luggage.
- Knapsack problems appear in real-world decision-making processes in a wide variety of fields, such as finding the ***least wasteful way to cut raw materials***, ***selection of investments and portfolios.***

2). Job sequencing with deadline

- In job sequencing problem, the objective is to ***find a sequence of jobs***, which is ***completed within their deadlines*** and ***gives maximum profit***.
- Let us consider, a set of n given jobs which are associated with deadlines and profit is earned, if a job is completed by its deadline.
- These jobs need to be ordered in such a way that there is ***maximum profit***.
- Constraints:
- Each job required ***1 unit of time.***
- No 2 jobs can be done simultaneously (***one job at a time***).

3). Minimum Spanning tree

- Graphs are represented as a ***set of edges or values***.
- Spanning tree is a ***sub graph of a graph***, since it is a ***subset of a graph***.
- Assume that a graph is given, having 6 (six) vertices and 6 (six) edges.
- Like, vertices (v) = (1,2,3,4,5,6)
- And edges (e) = { (1,2), (2,3), (3,4), ...}
- While preparing a spanning tree from a graph, we must ***take subset of edges*** but ***vertices must be as it is***.
- We must include all the vertices, than what about the edges?

Minimum Spanning tree…

- |V| = n = 6
- |E| = n-1 = 5
- There will be 6 vertices and 5 edges in a spanning tree (if the graph is containing 6 vertices and 6 edges).
- Definition: “Spanning tree is a sub graph of a graph having all the vertices and only n-1 edges.
- From a given ***weighed graph***, we can draw spanning tree in so many ways.
- But we are interested in the spanning tree which is having the ***minimum cost.***
- This could be achieved manually by drawing all the possible spanning tree from a given graph.

Minimum Spanning tree…

- But this is not a feasible solution.
- We are having two methods by which we can achieve a minimum spanning tree at just one try.

1. Prim's algorithm
2. Kruskal's algorithm

Prim's algorithm:

- This is a greedy method which can find minimum spanning tree.
- This method starts with the ***edge with the minimum cost.***
- Then ***select minimum weight*** out of all the ***connected edges.***

Minimum Spanning tree…

- Care is to be taken while selecting edges that, it does ***not form the cycle***.
- Prim's algorithm ***can't find spanning tree*** from the ***un-connected graph***.

Kruskal's algorithm:

- This is a greedy method which can find minimum spanning tree.
- This method starts with the ***edge with the minimum cost.***
- Than it again ***finds the minimum cost edge*** and draw the tree further.
- This will be ***continued until spanning tree is complete***.
- Spanning tree ***can be formed from the un-connected graph***.

4). Shortest path method : Dijkstra's

- In the shortest path problem, we are interested in ***the length of a particular path*** and not the total length.
- The problem is to ***determine the shortest pa***th from the source node to ***all the remaining node of the graph G.***
- A greedy strategy for this problem require a ***multi stage solution*** with an ***optimization measure***.
- The path can be ***built edge by edge***, the ***sum of the length of the path built until now can be used as the optimization measure.***
- Dijkstra's algorithm is well known and a popular algorithm for solving shortest path problem.
- It is implemented in ***high speed computer***, ***network routers*** and ***switches.***

Dynamic programming

- The idea of dynamic programming is quite simple, ***avoid calculating the same thing twice.***
- This is achieved by ***keeping a table of known results*** that feels up as sub-instances of the ***problem that is solved.***
- Dynamic programming is a ***bottom up technique.***
- It starts with the ***smallest*** and ***simplest sub-instances***.
- The dynamic programming strategy drastically (extremely) ***reduces the unnecessary calculations***, by ***avoiding the sequence which cannot possibly lead to an optimal solution.***
- In dynamic programming, an optimal sequence of decisions are arrived by the ***"principle of optimality".***

Dynamic programming…

Principle of Optimality:

An optimal sequence of decisions has a property that whatever be the initial state and decision, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

1. Rod Cutting problem
2. Multistage graph
3. Travelling salesman problem
4. Longest common subsequence
5. Matrix multiplication / Matrix chain multiplication

1). Rod Cutting problem

- This technique gives the answer to the question : ***How to cut a metal rod into pieces, so that the revenue (profit) obtained by selling them is maximum?***
- The rod has an ***integer unit length*** and ***cut lengths are also to be an integer units***.
- It is assumed that there is ***no cutting cost.***
- We are always interested in the ***unique cuts***.

2). Multi stage graph

- A multi stage graph is a ***weighted directed graph*** in which, ***vertices are grouped into several stages.***
- This method is ***applicable in the project management***, where in order to complete a project, we have to ***perform several activities in different stages***.
- At ***each stage***, we have to ***take several decisions*** on the ***basis of weights available***.
- We have to ***select the minimum cost edges***, in order to have ***minimum cost to complete a particular task.***
- That is why, it is called ***optimization problem (minimization problem)***, and this is solved through ***dynamic programming***.

3). Travelling salesman problem

- A salesman has to ***visit various cities from his hometown***.
- He has to visit all the cities ***exactly once.***
- Means he ***can not revisit the city nor he can miss any of it***.
- He has ***to return to his hometown*** after the journey.
- We have to find the ***route which costs minimum***.
- ***Cost adjacency matrix*** will be given with the problem itself.
- This algorithm can be applicable in ***logistic, deciding the route, planning and scheduling***.

4). Longest Common Subsequence

- A subsequence is a subset of elements from the sequence with ***strictly increasing order*** (not necessarily continuous).
- There can be ***more than one common sub sequences*** for the given two strings.
- We have to find out the ***longest common subsequence***.
- It means we have to find common subsequence that is of ***maximum length***.
- We have to follow the ***relative order of character positions***.
- It means that, ***while comparing*** we must look to the ***forward direction*** and ***not backward***.
- It can be used in the '***diff-utility', revision control system***, etc.

5). Matrix chain multiplication

- Suppose, we have more than 2 matrices (A1, A2, A3); we have to multiply these matrices such that the cost of multiplication is minimum.
- We have to find the sequence of multiplication in which the cost is minimum.
- We can first multiply A1 with A2 and than the answer is multiplied with A3, or we can multiply A2 and A3 than the answer is multiplied with A1.
- This algorithm is applicable in signal processing and network industry.

Unit-2: Divide and Conquer Strategy

Divide and Conquer

Decrease and Conquer

Divide and Conquer Strategy

- In divide and conquer strategy, problem is ***divided into smaller sub-problems*** and then ***each problem is solved independently.***
- When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where ***no more division is possible.***
- Those ***"atomic"*** smallest possible sub-problem are solved.
- The solution of all sub-problems is finally ***merged*** in order to obtain the solution of an ***original problem.***
- Divide and conquer is a ***recursive strategy***.
- It is important to note that, ***sub-problems must be similar to the main problem***.

Merge sort

- Merge sort algorithm is considered as one of the good example of Divide and Conquer method.
- We assume that sorting is to be done in the ascending order.
- Given a set (array) of unsorted numbers, idea is to split this set (array) in to two almost equal parts.
- Than, each of these is individually sorted and the two arrays obtained as a result are merged to obtain a single sorted array.
- This operation can be carried out recursively on the half sized array till we are left with arrays of size 2 or 1.

Pros and Cons of Divide and Conquer

Pros

- It can solve difficult problems easily as, it divides the problem into similar sub-problems.
- Methods which use divide and conquer is much easier to understand as the same process is repeated many number of times.
- Algorithm designed with divide and conquer strategy does not require any modifications, as the sub-division of problem is parallel and can be processed by parallel processing systems.
- It makes efficient use memory cache, because when the problem gets divided, they are so small that they can be easily solved in the cache itself.

Pros and Cons of Divide and Conquer…

Cons

- Divide and conquer strategy uses recursion that makes it a little slower and if a little error occurs in the code, than the program may enter into infinite loop.
- Usage of stack may make use of extra space.
- Performing recursion for number of time greater than the stack in the CPU than the system may crash.
- If all the sub-problems are working properly but, error persist in any single sub-problem than it may lead to error.
- Sometimes it happens that sub-problem may increase the solving time and it may consume extra memory.

Divide and Conquer vs. Dynamic Prog.

Divide and Conquer	Dynamic Programming
This is an algorithm that recursively breaks down a problem into two or more sub-problems until, it becomes simple enough to be solved directly.	This is an algorithm that helps to efficiently solve a class of problems that has overlapping sub-problems.
Sub-problems are independent of each other.	Sub-problems are inter dependent.
This algorithm works on recursive approach.	It is non-recursive approach.
It is time consuming as it solves each sub-problems independently.	Requires lesser time as it uses the data of the previous stage.
Comparatively, this approach is considered less efficient.	Considerably more efficient approach.

Decrease and Conquer

- The decrease and conquer approach works on following steps:
  - Step 1: Decrease
  - Step 2: Conquer
  - Step3: Extend

  Decrease:
  - Reduce the problem instance to smaller instance of the same problem and extend the solution.
  - Example: 1000 * 1000.
  - Reduce the number by removing two zeros (00).

  Conquer:
  - Now, conquer the problem by solving a smaller number (10 * 10) of a problem.

  Extend:
  - Take the answer from the conquer and add removed number (0000) to the answer.
  - So that you can have an actual answer.

Decrease and Conquer…

- It is suggested that the name “Divide and Conquer” should be used only when each problem may ***generate two or more sub-problems.***
- The name “Decrease and Conquer” has been proposed for the ***single sub-problem class.***
- According to this definition, ***merge sort comes under “Divide and Conquer”*** strategy and ***binary search comes under “Decrease and Conquer” strategy.***

Unit-4: Backtracking

- Backtracking is an algorithmic-technique for solving problems recursively by trying to ***build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints*** of the problem at any point of time.
- Backtracking found its first use in ***games and many artificial intelligence applications***.
- It was found that such problems ***require traversal of a large search tree***.
- It requires ***exponential time.***
- In order to reduce the search time, strategy used is ***"Back tracking".***
- Back tracking is a systematic method ***to examine all possible configuration of a problem space***.
- In this method, ***we generate each possible configuration exactly once, avoiding repetitions and missing configuration.***

Combinatorial search

- The selection of a given number of element from a largest number without considering the arrangements.
- The benefits that can occur from analyzing and improving algorithms for problems which implements combinatorial search (search in non-systematic manner) the time of which grows exponentially in the size of the problem, which can be very high.
- So, by using exhaustive search techniques ***we can optimally solve small problems although the tie complexity may be very high***.
- In certain situation, ***it would be good to spend more time to get an optimal solution.***

Search and Traversal

- The most problems with graphs and trees require ***searching for a node with same specific property***.
- When the search requires necessarily ***visiting every node in the structure,*** then it is called a traversal.
- There are three methods of traversal in binary tree.

1. **In-order traversal**

   -Traverse left sub-tree, Visit and use the root, Traverse the right sub-tree

2. **Pre-order traversal**

   -Visit and use the root, Traverse left sub-tree, Traverse the right sub-tree

3. **Post-order traversal**

   -Traverse left sub-tree, Traverse the right sub-tree, Visit and use the root

8 Queens problem

- In general form, this is of “n-queens” problem, but to understand it in the precise manner we understand it with 8 queens or 4 queens problem.
- In the 4 queens problem, we have 4*4 chess board, we have to place all the queens such that no queen can attack each other.
- We have to take care that no two queens are placed on the same column and/or on the same row.
- No two queens should be placed in the diagonal cells of the chess board.
- We will have 4*4 chess board and we have to place 4 queens on the chess board as per the constraints discussed.
- It helps you build quality like constraint programming, or evolutionary algorithms.
- We found that ‘N’ problem can be solved with just in the form of ‘4’.
- It can give answer to question like: Can I schedule ‘N’ work items using shared resources, within time limit?

m-Coloring problem

- In m-coloring problem, a graph G=(V,E) is a set of m colors with its constraints.
- We want to determine if the nodes of the graph can be colored with distinct colors, so that no two adjacent nodes have the same color.
- M-color optimization problem finds the minimum number of colors required to paint the graph.
- This minimum number is known as “chromatic number” of a graph.
- This technique is applicable in making a schedule, a time table, sudoku.
- There are many ways to color all the vertices in different color.

Hamiltonian Circuit

- A circuit is a path in an undirected or directed graph that visits each vertex ***exactly once.***
- To determine whether such paths and cycles exists in a graph is the “Hamiltonian path problem”.
- ***Definition***: “A Hamiltonian path is a path that visits each vertex of the graph exactly once and return to the starting vertex.”
- As per the rule of the backtracking, we will explore all the possibilities which form a Hamiltonian circuit for the given graph.
- This problem looks similar to Travelling salesman problem up to some extent. But, in TSP we need to find the smallest path.
- This may applied into Computer Graphics, electronic circuit design, etc.

Constructing Typical State Space

- To understand how backtracking work, we must see how objects such as permutations and subset can be constructed by determining the right state space.

❖ **Constructing all subsets**

- To design a suitable state space for representing a collection of objects, it is important to know the number of objects we will need to represent.
- It means, how many subset of n elements set exists? Ex: {1,2,3,…,n}
- There are exactly two subsets when n=1 namely {Φ} and {1}, and four subsets when n=2, and 8 subsets when n=3.
- There are 2^n subsets of n elements.

Constructing Typical State Space…

❖Constructing all permutations

- To design a suitable state space for representing permutations, we start by counting them, there are n distinct choices for the value of the 1st element of a permutation of {1,2,…n}
- Once we have fixed this value of n, there are n-1 candidates remaining for the second position, since we can have any value except n1, as repetition is not allowed.
- If we take this argument to its logical conclusion we get a total of n! distinct permutations.

Unit-5: Branch & Bound Algorithm

- In order to find ***an optimal solution over a finite set of alternatives*** an obvious approach is to ***enumerate all the alternatives and then select the best option.***
- This method will be ***more demanding when the problem statement is very small.***
- Branch & Bound algorithm ***reduces the number of alternatives that needs to be considered*** by partitioning the problem into a set of smaller sub-problems by using the information available at each step.
- Each step generates an information which eliminates steps for final results.

8-Puzzle problem

- In the 8-puzzle problem, we are given with 3*3 matrix in which 1,2,….8 numbers are dynamically arranged.
- A matrix will have one empty cell which will be used to move numbers from one place to another.
- All the numbers are scrambled and the goal is to reach the target position.
- We can move a number to the left side, right side, in upward direction or downward direction as pert the space available.
- The solution space tree will have nodes representing the state of 8 tiles on the 3*3 board.
- We shall use the best first search technique.
- The given starting scrambled state will be represented as the root node.
- We shall use “Manhattan Distance” as the cost of re-arranging.

Shortest Path problem

- In this method, we have to find the shortest path between the starting node and the sink node (goal).
- We will find it in the depth first search (DFS) manner.
- We are interested in finding the optimal path amongst all the paths.
- This method of finding the shortest path is to be used when the size of problem is small.
- While finding the path we cannot move in the backward direction.
- Always choose the shortest value from the leaf node for further exploration.
- The bounding process takes place as we reach the sink.
- Prune the path whose value is bigger than the bound.

0/1 Knapsack problem

- The objective of this method is to fill the knapsack such that the profit earned is maximum and we are filling the knapsack less than or equal to its capacity.
- We would include an object (full of its weight) or not include it at all.
- Knapsack problem (0/1) is actually a maximization problem which can be solved with Greedy method, Dynamic Programming and Backtracking also.
- In Branch and Bound, we would consider it as a minimization problem, we will mark profit as a negative and then convert it into positive later on.
- We will use Least Cost - Branch and Bound (LC-BB).
- It means Least cost branch will be explored first.

Unit-1: Basics of Design & Analysis of Algorithms

- Computer have lots of memory but no imagination.
- So solving a problem or getting work done by a computer can be quite a demanding task.
- Getting work done by computer require considerable amount of careful planning and attention to details.
- One of the difficulties in getting a problem solved by a computer is that, when we humans perform any task, there may be many steps which are doing un-consciously and most of the time we are not aware of them.
- When the same job is required to be done by a computer, all these obvious and un-obvious steps need to be included in the algorithm.

Solving a problem with a computer

- Following steps are required to solve a problem using computer:

1. Statement of a problem or problem definition
2. Development of a model
3. Design of the algorithm
4. Checking the correctness of the algorithm
5. Implementation in some programming language.

1. Statement of a problem or problem definition

- Initially we have to understand the requirements of a particular problem.
- If we are the originator of the problem than we have to ask ourselves several questions.
- If originator of the problem is someone else than we have to interview him/her expecting detailed answers of every questions.
- To get right answers, requires right questions.

a) Do we understand the vocabulary used in the statement problem?

b) What information is given?

c) What is to be found?

d) How to recognize a valid solution?

e) What important information is missing?

f) What assumptions have been made?

2. Development of a model

- We have to develop a mathematical model where calculations can be done.
- There are two more questions which must be answered:

1. “Which mathematical structure seems to be suited?
2. Are there any problems that have been solved, which resemble this one?

- There are several general problem solving strategy which are available:

1. Divide and Conquer
2. Dynamic Programming
3. Greedy Search
4. Back tracking
5. Branch and Bound

3. Design of the algorithm

- There are various issues which arises in the design of an algorithm.
- The design approach depends mainly on the model we are choosing.
- There can be more than one algorithm to solve the problem and the choice between them will be based on the effectiveness.
- The natural tendency to start coding a problem early should be avoided.
- Additional time that is spent in the initial design phase would help you develop a program quickly which is likely bug free and efficient.

4. Checking the correctness of the algorithm

- One of the difficult task in algorithm development is to prove its correctness.
- One of the possible way to prove its correctness is to input several possibilities and compare it against manually calculated or known result.
- This approach to measure the correctness is not sufficient for its reliability and efficiency.
- This tells us that an algorithms correctness does not necessarily imply anything about its efficiency.

5. Implementation in some programming language

- Once the algorithm is designed and checked for correctness, it is the time to code it in some programming language.
- This could be a straight forward or a difficult task depending upon how clearly the algorithm is written and the programming strategy is adopted.
- So one major difficulty is that we must design the data structures to represent important information held within the algorithm.
- To do so we must answer the following questions:

1. What are the variables?
2. What are their types?
3. How many arrays are required and what are the sizes?
4. Is it worth while to use linked list?
5. Which subroutines are needed?
6. Which programming language will support this data structure sufficiently?

What is Algorithm?

- The word **'algorithm'** means "set of rules to be followed in any type of problem-solving operations".
- It means algorithm is a set of rules/instructions that step-by-step define how a work is to be executed so as to get the expected results.

Characteristics of an Algorithm

1). **Clear and Unambiguous**: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.

2). **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs.

3). **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well.

What is Algorithm?...

4). Finite-ness: The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.

5). Feasible: The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources. It must not contain some future technology, or anything.

6). Language Independent: The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

Top down design

- Programmers have tried to design and code algorithms directly in programming language, rather than carefully decomposing the problem into simpler problems and checking the relationship.
- The method which is applying the concept of decomposing is called the “Top down technique”.
- Experience has shown that top-down design technique results in fewer mistakes than line by line development.
- The top down method decomposes the overall problem into precisely specified sub-problems.
- Than it proves that if each sub-problem is solved correctly, and the solutions are fitted together in a specified way, than the original problem will be solved correctly.

Structured programming

- A structured programming is useful in development of the algorithm and in the implementation of the program.
- While writing a program, the major concerns of a programmer are:

1. Easy debugging
2. Easy modification, extension and maintenance
3. Easy understanding by other programmers in the team
4. Reasonable assurance that the program logic is correct

- Structured programming is a perfect way of developing and implementing algorithms.
- It means that, while debugging a program, a programmer would like to know:

  "How did I came here"?

  - That is the path that the program control has traversed.
  - In a structured programming, all the control constructs have a single entry point & single exit point.

Factors affecting an efficiency of an algorithm

- The design and implementation of algorithm have an influence on their efficiency.
- An algorithm when implemented must use some resources to complete its task.
- Resources like: CPU time, and Internal memory (RAM), must be used efficiently so that resources could be used somewhere else.
- In the earlier time, cost of the computer resources was the driving force behind the desire to design an efficient algorithms.
- In the present, the cost of these resources are reduces and it is continuously decreasing but, because of more complex requirements, efficient algorithm is inevitable.

1. Removing redundant computation outside the loops:

- Most of the in-efficiency that lies into the design of an algorithm are due to redundant computations or un-necessary storage.
- The effect of redundant computation is serious when it is embedded within a loop, which must be executed many times.
- The most common mistake when using loop is to repeatedly calculate the part of an expression.

Factors affecting an efficiency of an algorithm…

- Example:

```
x=0
for i=1 to n do
begin
        x=x+0.01;
        y=(a*a*a+c) * x*x+b*b*x;
end
```

Factors affecting an efficiency of an algorithm…

2. Referencing of an array elements:
- Generally, arrays are processed by iterative constructs.
- If proper care is not taken while programming, redundant computation can creep into an array processing.

3. Inefficiency due to late termination:
- Another possibility of in-efficiency that may occur into the implementation of an algorithm is when considerably more tests are carried out.
- Example: Bubble sort

4. Early detection of desired output conditions
- Any sorting algorithm requires to compare all the elements with each other.
- What if, if the elements which are input for sorting are already in the sorted structure?
- By putting a check at the very early stage of an algorithm we could reduce the un-required processing of the algorithm.

Design using recursion

- Some computer programming languages allow a module or function to call itself.
- This technique is known as recursion. In recursion, a function α either calls itself directly or calls a function β that in turn calls the original function α.
- The function α is called recursive function.
- One may argue why to use recursion, as the same task can be done with iteration.
- The first reason is, recursion makes a program more readable and because of latest enhanced CPU systems, recursion is more efficient than iterations.
- **Time Complexity:**
- Amount (extra) of time required to execute/compute the certain function/task is known as time complexity.
- In case of iterations, we take number of iterations to count the time complexity.
- Likewise, in case of recursion, assuming everything is constant, we try to figure out the number of times a recursive call is being made.
- **Space Complexity:**
- Space complexity is counted as what amount of (extra) space is required for a module to execute.
- The space required in recursion is high, hence it is considered that space complexity of recursive function is high.

Regular Expressions (RE)

- Regular expression is a method of expressing long and possible infinite sequences of some symbols.
- Generally, it is in the form of the formula shown in the form of the formula which shows ordering of various symbols.
- Suppose, we want to denote two strings, ‘ab’ and ‘ac’, it will be written as:
  - RE r = a(b|c)
- Here, the ‘|’ (pipe) symbol denotes alternate possibilities and string concatenation operation which is denoted by simply writing symbols one after another.
- Use of RE here is an indication of a link between the theory of algorithms and the theory of formal languages.
- One is the specification of how to do computation and the other is a record of a dynamic behaviour when the computation takes place.