MODULE 5

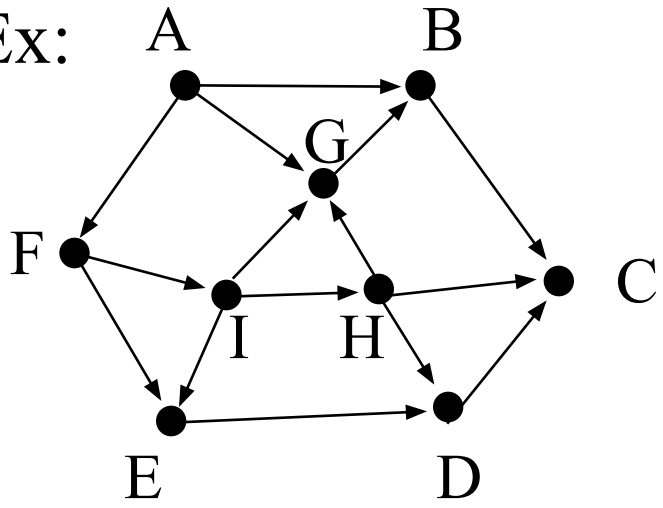
SOPHISTICATED ALGORITHMS

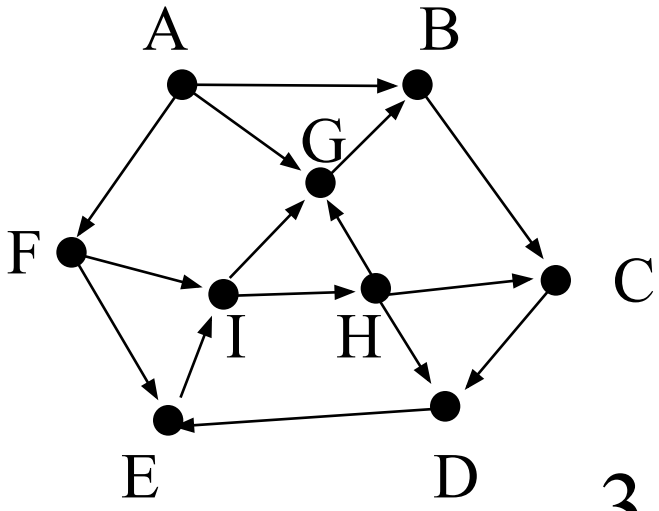
Topological Sorting

- Topological order
  - A list of vertices in a directed graph without cycles such that vertex x precedes vertex y if there is a directed edge from x to y in the graph
  - There may be several topological orders in a given graph
- Topological sorting
  - Arranging the vertices into a topological order

Topological Sort

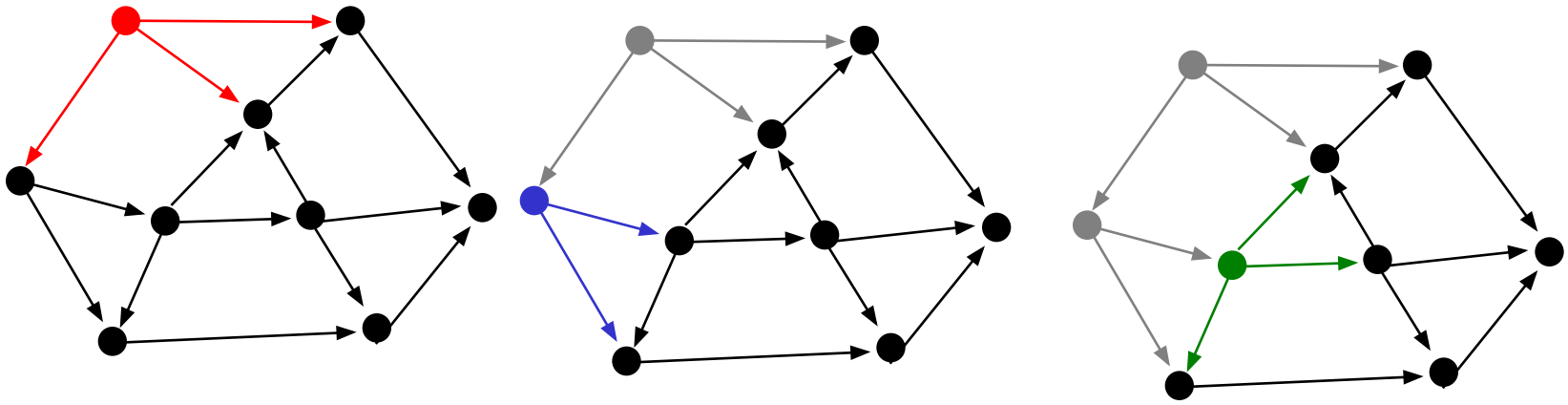
- Directed graph G.
- Rule: if there is an edge u v, then u must come before v.
- Ex:





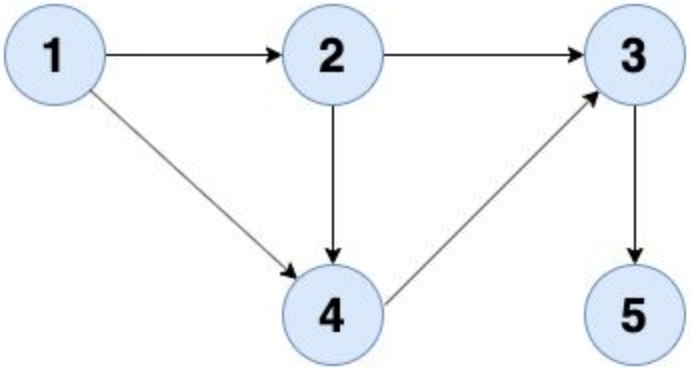
Intuition

- Cycles make topological sort impossible.
- Select any node with no in-edges
  - print it
  - delete it
  - and delete all the edges leaving it
- Repeat



Topological Sorting

For example-The topological sort for the below graph is 1, 2, 4, 3, 5



Important Points to remember

- There can be multiple topological ordering for a Directed Acyclic Graph.
- In order to have a topological sorting, the graph must not contain any cycles and it must be directed i.e. the graph must be a DAG. (**Think!**)

Topological Sorting

- Simple algorithms for finding a topological order
  - topSort1
    - Find a vertex that has no successor
    - Remove from the graph that vertex and all edges that lead to it, and add the vertex to the beginning of a list of vertices
    - Add each subsequent vertex that has no successor to the beginning of the list
    - When the graph is empty, the list of vertices will be in topological order

STRING MATCHING

The fundamental string searching (matching) problem is defined as follows: given two strings – a text and a pattern, determine whether the pattern appears in the text.

- KMP Algorithm
- Rabin Karp Algorithm

STRING MATCHING

Rabin Karp Algorithm(Page 993-Thomas H. Cormen)

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m−1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp + P[i]) mod q
 8      t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n − m                // matching
10      if p == t_s
11          if P[1..m] == T[s + 1..s + m]
12              print "Pattern occurs with shift" s
13      if s < n − m
14          t_{s+1} = (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

STRING MATCHING

KMP Algorithm(Page 1005-Thomas H. Cormen)

```
KMP-MATCHER(T, P)
 1  n = T.length
 2  m = P.length
 3  π = COMPUTE-PREFIX-FUNCTION(P)
 4  q = 0                                  // number of characters matched
 5  for i = 1 to n                         // scan the text from left to right
 6      while q > 0 and P[q + 1] ≠ T[i]
 7          q = π[q]                       // next character does not match
 8      if P[q + 1] == T[i]
 9          q = q + 1                      // next character matches
10      if q == m                          // is all of P matched?
11          print "Pattern occurs with shift" i − m
12          q = π[q]                       // look for the next match
```

STRING MATCHING

KMP Algorithm

```
COMPUTE-PREFIX-FUNCTION(P)
 1  m = P.length
 2  let π[1..m] be a new array
 3  π[1] = 0
 4  k = 0
 5  for q = 2 to m
 6      while k > 0 and P[k + 1] ≠ P[q]
 7          k = π[k]
 8      if P[k + 1] == P[q]
 9          k = k + 1
10      π[q] = k
11  return π
```

Complexity Classes

In computer science, there exist some problems whose solutions are not yet found, the problems are divided into classes known as Complexity Classes.

In complexity theory, a Complexity Class is a set of problems with related complexity.

These classes help scientists to groups problems based on how much time and space they require to solve problems and verify the solutions.

Types of Complexity Classes

P Class

NP Class

NP hard

NP complete

Complexity Classes

P Class

The P in the P class stands for Polynomial Time. It is the collection of decision problems(problems with a “yes” or “no” answer) that can be solved by a deterministic machine in polynomial time.

Features:

The solution to P problems is easy to find.

P is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice(a problem that can be solved in polynomial time). But the problems that can be solved in theory but not in practice are known as intractable(a problem that cannot be solved by a polynomial-time al- gorithm.)

This class contains many natural problems like:

Calculating the greatest common divisor.

Finding a maximum matching.

Complexity Classes

NP Class

The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

Features:

The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.

Problems of NP can be verified by a Turing machine in polynomial time.

This class contains many problems that one would like to be able to solve effectively:

Boolean Satisfiability Problem (SAT).

Hamiltonian Path Problem.

Graph coloring.

Complexity Classes

NP-hard class

An NP-hard problem is at least as hard as the hardest problem in NP and it is the class of the problems such that every problem in NP reduces to NP-hard.

Features:

All NP-hard problems are not in NP.

It takes a long time to check them. This means if a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.

A problem A is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Some of the examples of problems in Np-hard are:

Halting problem.

Qualified Boolean formulas.

No Hamiltonian cycle.

Complexity Classes

NP-complete class

A problem is NP-complete if it is both NP and NP-hard. NP-complete problems are the hard problems in NP.

Features:

NP-complete problems are special as any problem in NP class can be transformed or reduced into NP-complete problems in polynomial time.

If one could solve an NP-complete problem in polynomial time, then one could also solve any NP problem polynomial time.

Some example problems include:

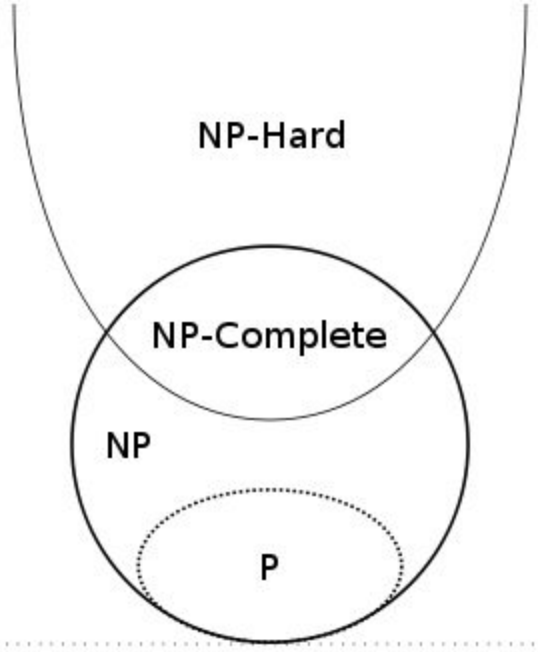
0/1 Knapsack.

Hamiltonian Cycle.

Satisfiability.

Vertex cover.

Complexity Classes



Determinism and Non-determinism

Sr. No.	Key	Deterministic Algorithm	Non-deterministic Algorithm
1	Definition	The algorithms in which the result of every algorithm is uniquely defined are known as the Deterministic Algorithm. In other words, we can say that the deterministic algorithm is the algorithm that performs fixed number of steps and always get finished with an accept or reject state with the same result.	On other hand, the algorithms in which the result of every algorithm is not uniquely defined and result could be random are known as the Non-Deterministic Algorithm.
2	Execution	In Deterministic Algorithms execution, the target machine executes the same instruction and results same outcome which is not dependent on the way or process in which instruction get executed.	On other hand in case of Non-Deterministic Algorithms, the machine executing each operation is allowed to choose any one of these outcomes subjects to a determination condition to be defined later.
3	Type	On the basis of execution and outcome in case of Deterministic algorithm, they are also classified as reliable algorithms as for a particular input instructions the machine will give always the same output.	On other hand Non deterministic algorithm are classified as non-reliable algorithms for a particular input the machine will give different output on different executions.
4	Execution Time	As outcome is known and is consistent on different executions so Deterministic algorithm takes polynomial time for their execution.	On other hand as outcome is not known and is non-consistent on different executions so Non-Deterministic algorithm could not get executed in polynomial time.
5	Execution path	In deterministic algorithm the path of execution for algorithm is same in every execution.	On other hand in case of Non-Deterministic algorithm the path of execution is not same for algorithm in every execution and could take any random path for its execution.

Approximation algorithms

Planar graph colouring.

In graph theory, a planar graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other.

Problem: Coloring of the vertices of a graph such that no two adjacent vertices have the same color

Goal: minimize the number of color used

ALGORITHM

Refer note book

Approximation algorithms

Vertex cover(Page no: 1109 Thomas H. Cormen)

```
APPROX-VERTEX-COVER(G)
1  C = ∅
2  E' = G.E
3  while E' ≠ ∅
4      let (u, v) be an arbitrary edge of E'
5      C = C ∪ {u, v}
6      remove from E' every edge incident on either u or v
7  return C
```