FACULTY NAME : Dr .P.BOOMI

DEPARTMENT : BIOINFORMATICS

SEMESTER : IInd

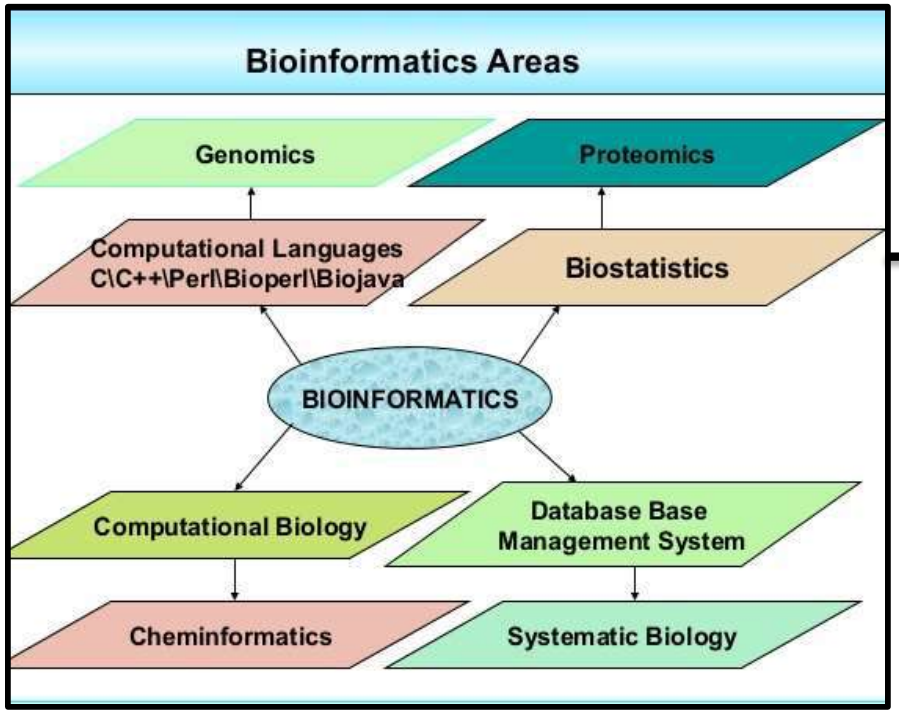
COURSE NAME : COMPUTATIONAL BIOLOGY

COURSE CODE : 502201

OUTLINE

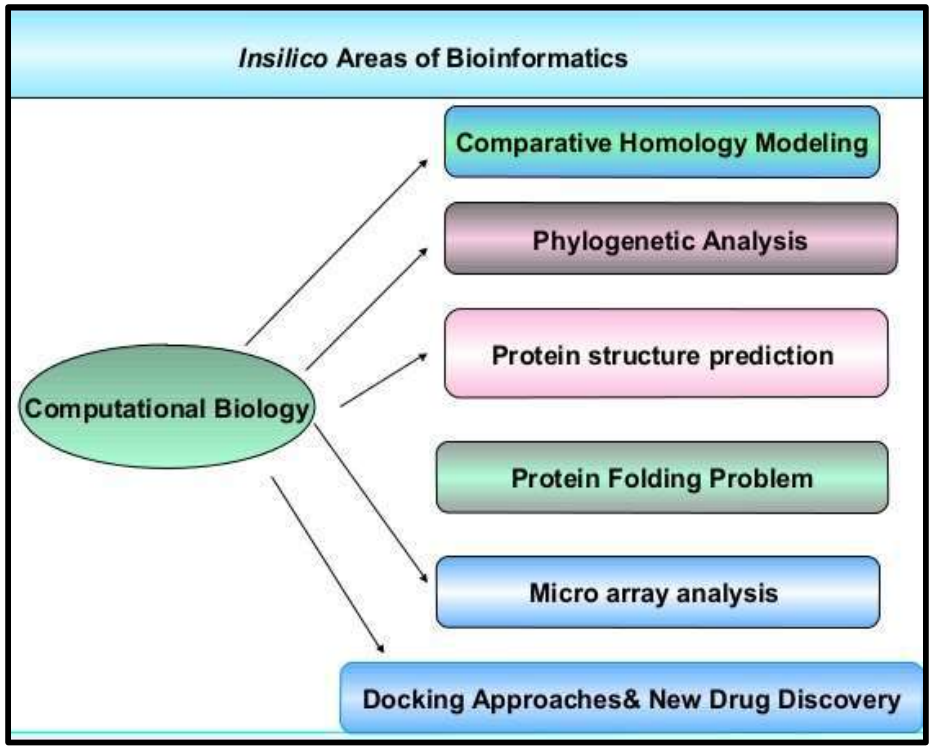
- Strings, Substrings, superstrings
- Suffix strings, prefix strings
- Operations on strings
- Concatenation, delete operator
- Graphs, directed, connected, cyclic, complete graphs
- Trees and terminology
- Algorithms, big O notation
- Classification of algorithms

Bioinformatics



Which describes using computational techniques to access, analyze, and interpret the biological information in any type of biological database

Computational Biology



Computational Biology encompasses all computational methods and theories applicable to molecular biology and areas of computer based techniques for solving biological problems

String

- Series of characters treated as single unit
- May include letters, digits, etc.
- Object of class String

- string is a finite sequence of symbols.

  For example,

  string (s, t, r, i, n, g)

  CS4384 (C, S, 4, 3, 8)

  101001 (1, 0)
- Symbols are given through alphabet.
- An alphabet is a finite set of symbols.

Length of a String

- The length of a string x is the number of symbols contained in the string x, denoted by |x|.
- For example, | string | = 6,
- |CS5400| = 6, | 101001 | = 6.
- The empty string is a string having no symbol, denoted by ε.

Equal: Two strings $x_1x_2\cdots x_n$ and $y_1y_2\cdots y_m$ are equal if and only if

(1) $n=m$ and

(2) $x_i=y_i$ for all i.

- For example, 01 ≠ 010 and 1010 ≠1110.

Substring

A substring of a string is another string that occurs "in". For example, "the best of" is a substring of "It was the best of times". This is not to be confused with subsequence, which is a generalization of substring. For example,

$$T = t_1 \ldots\ldots t_n$$

Example: The string ana is equal to substrings (and subsequences) of banana at two different offsets:

```
banana
  | | | | |
  ana| |
     | | |
     ana
```

Superstrings

Computational Challenge: assemble individual short fragments (reads) into a single genomic sequence ("superstring")

- Every path that covers every node is a superstring
- Zero weight edges result in alignments like:

```
GACA--------
----GCCC-----
--------TTAAAG
```

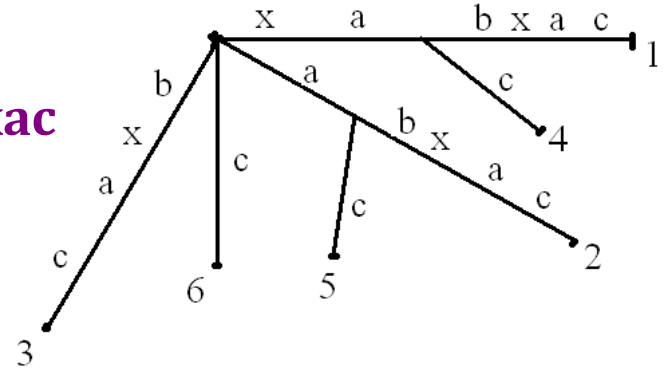
- Higher weights produce more overlap, and thus shorter strings
- The *shortest common superstring* is the highest weight path that covers every node

Suffix Trees

A suffix tree T for an m-character string S

- A rooted directed tree with exactly m leaves numbered from 1 to m.
- Each internal node, other than root, has at least two children and each edge is labeled with a non-empty substring of S.
- No two edges out of a node can have edge-labels beginning with the same character.
- For any leave i, the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i, that is, spells out S[i,…,m]

The suffix tree for string xabxac



Prefix

- Prefix (of a string)
  - Any number of leading symbols of that string
  - Example: abc
    - Prefixes: ε, a, ab, abc
- Proper Prefix (of a string)
  - A prefix of a string, but not the string itself
  - Example: abc
    - Proper prefixes: ε, a, ab

Prefix Property

- Context-Free Language (CFL) L is said to have the prefix property whenever w is in L and no proper prefix of w is in L
- Not considered a serve restriction
  - Why?
    - Because we can easily convert a DCFL to a DCFL with the prefix property by introducing an endmarker

Prefix and Suffix

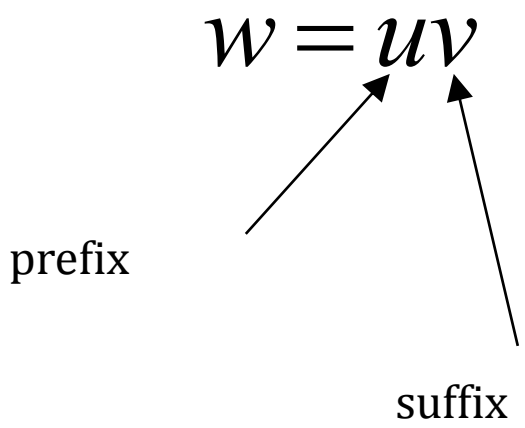
$abbab$

- Prefixes Suffixes

Prefixes	Suffixes
λ	$abbab$
a	$bbab$
ab	bab
abb	ab
$abba$	b
$abbab$	λ

Suffix trees can be used to solve in linear time

- exact matching problem.
- many string problems more complicated than exact matching.
- "We know of no other single data structure that allows efficient solutions to such a wide range of complex string problems"



String operations

In computer science, in the area of formal language theory, frequent use is made of a variety of string functions; however, the notation used is different from that used for computer programming, and some commonly used functions in the theoretical realm are rarely used when programming. This article defines some of these basic terms.

Concatenation

- Definition: $L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$
- Example: $\{a, ab, ba\}\{b, aa\}$

$$= \{ab, aaa, abb, abaa, bab, baaa\}$$

Concatenation

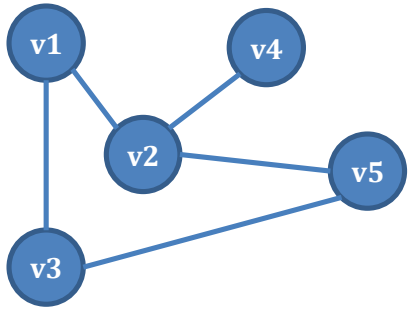
- The concatenation of two strings x and y is a string xy, i.e., x is followed by y.
- For example, CS5400 is a concatenation of CS and 5400.
- In particular, we denote

  $xx = x^2$, $xxx = x^3$, $xxxx = x^4$, ..., and define

  $x^0 = \varepsilon$
- For example, 101010 = $(10)^3$, $(10)^0 = \varepsilon$

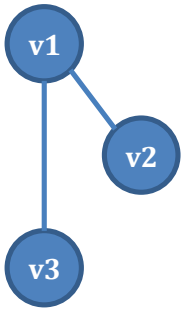
Graphs

A graph is a way of specifying relationships among a collection of items. A graph consists of a set of objects, called nodes, with certain pairs of these objects connected by links called edges

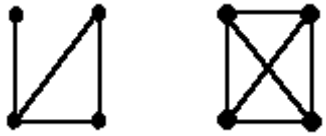


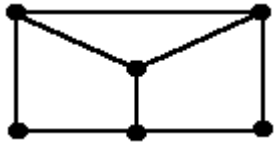
- Graph $G=(V,E)$ is a set of vertices V and edges E
  - $V = \{v1, v2, v3, v4, v5\}$
  - $E = \{(v1, v2), (v1, v3), (v2, v4), (v2, v5), (v3, v5)\}$

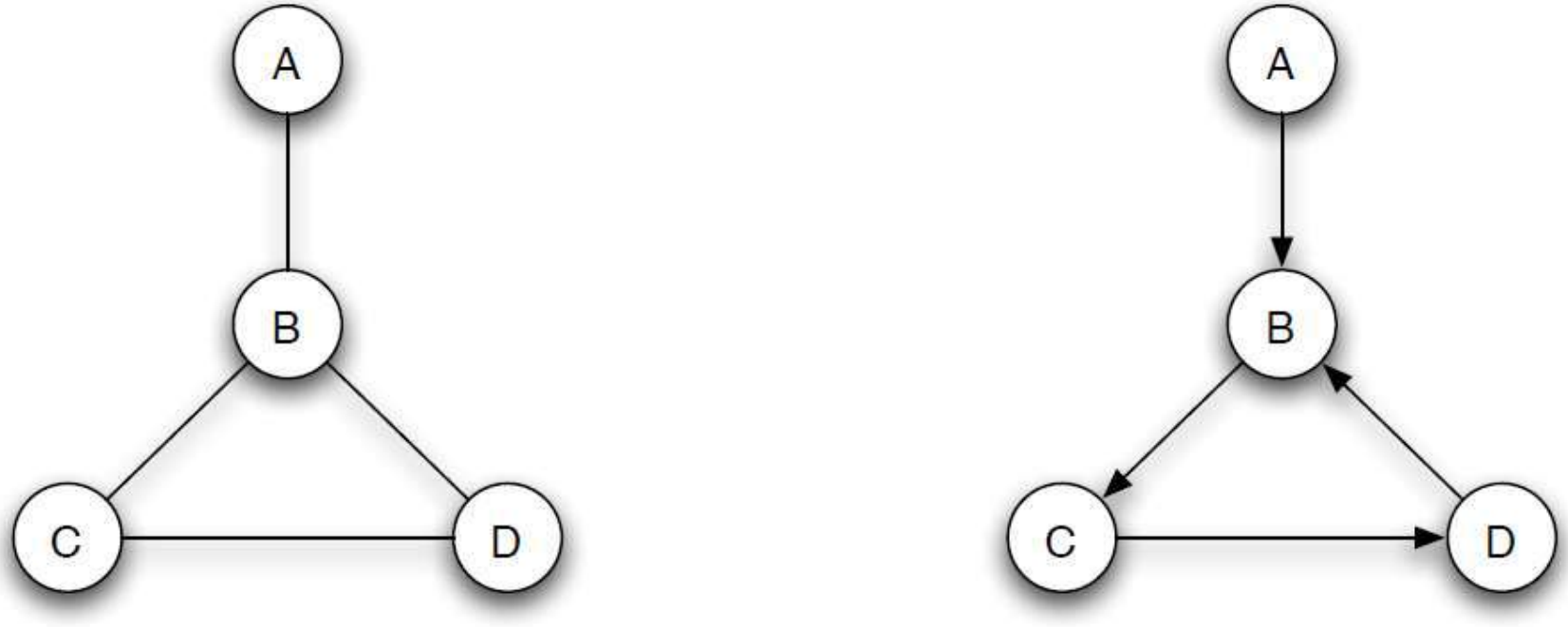
- A subgraph G' of G is induced by some $V' \subset V$ and $E' \subset E$
  - For example, $V' = \{v1, v2, v3\}$ and $E' = \{(v1, v2), (v1, v3)\}$



- Graph properties:
  - Directed vs. undirected
  - Weighted vs. unweighted
  - Cyclic vs. acyclic
  - Connectivity (node degree, paths)





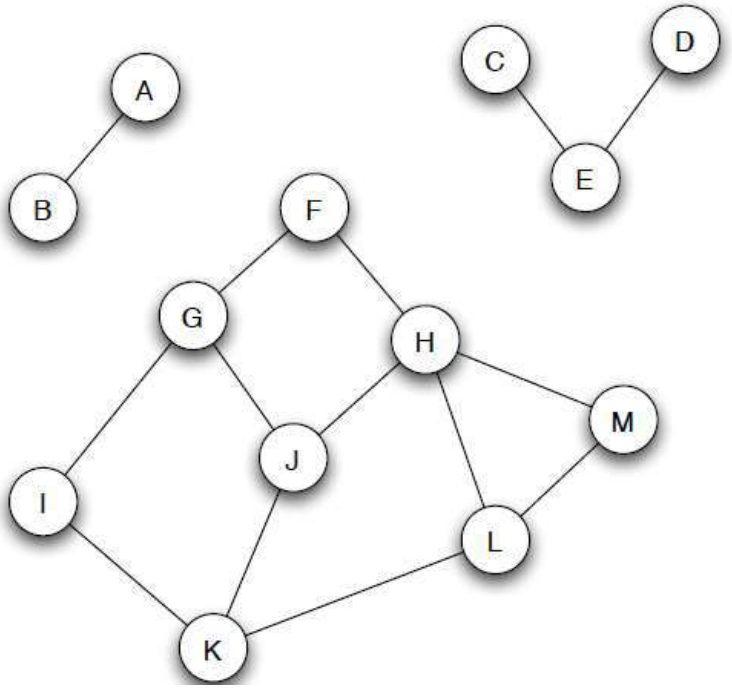


(a) *A graph on 4 nodes.* (b) *A directed graph on 4 nodes.*

Figure 2.1: Two graphs: (a) an undirected graph, and (b) a directed graph.

Directed graph to consist of a set of nodes, as before, together with a set of *directed edges; each directed edge is a link from one node* to another, with the direction being important.

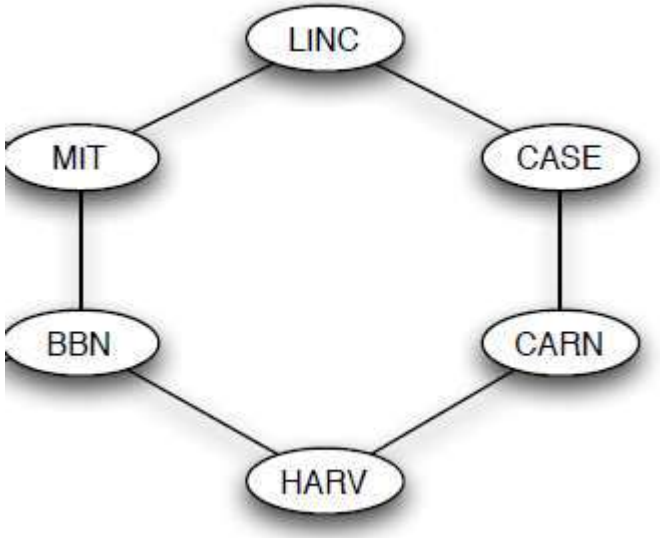
Connected Graph



Given a graph, it is natural to ask whether every node can reach every other node by a path. With this in mind, we say that a graph is *connected if for every pair of* nodes, there is a path between them.

For example, the 13-node Arpanet graph is connected; and more generally, one expects most communication and transportation networks to be connected or at least aspire to be connected — since their goal is to move traffic from one node to another.

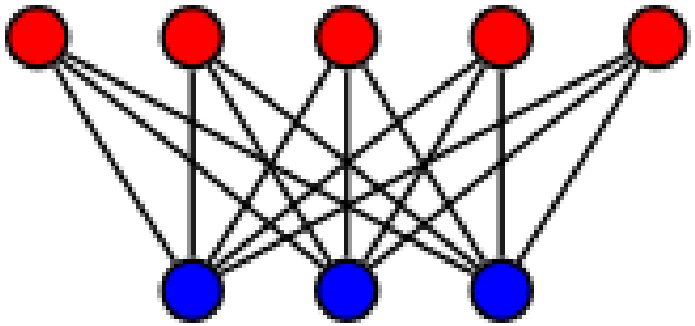
Cycles



A particularly important kind of non-simple path is a *cycle, which informally is a* "ring" structure such as the sequence of nodes like linc, case, carn, harv, bbn, mit and linc.

More precisely, a cycle is a path with at least three edges, in which the first and last nodes are the same, but otherwise all nodes are distinct

Complete graph

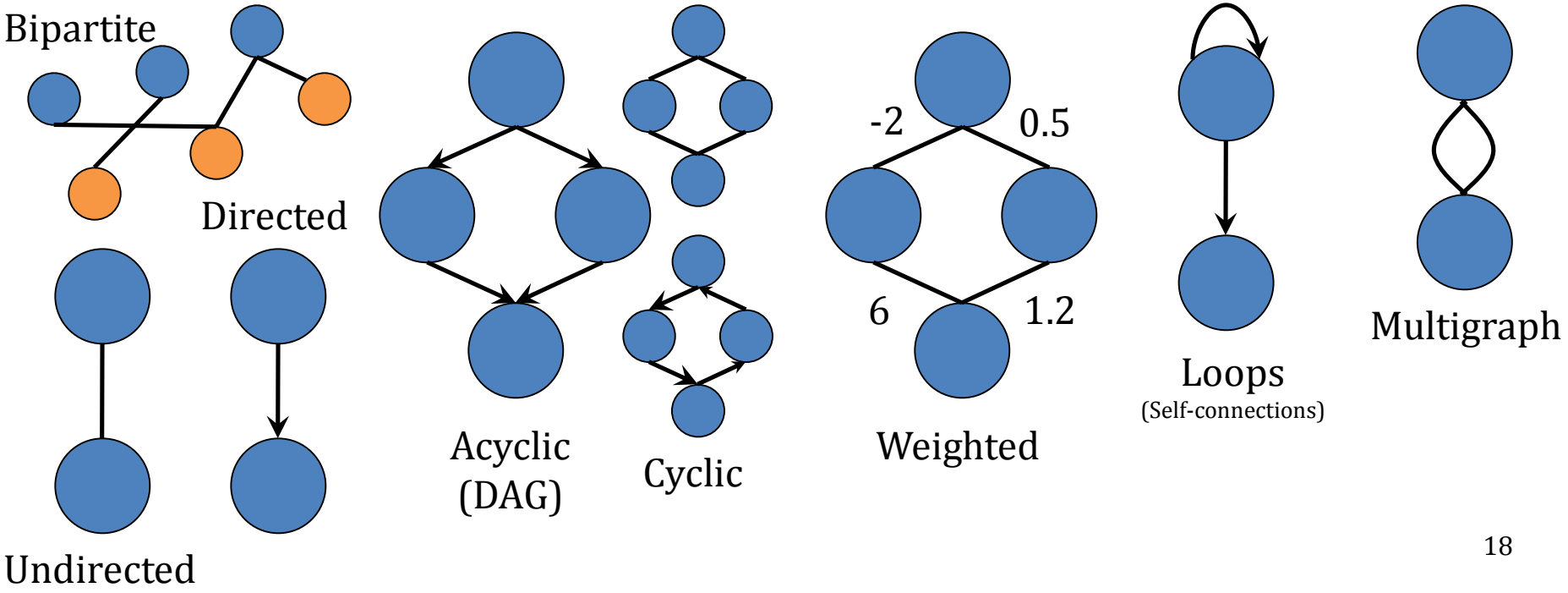


a complete graph is a special kind of bipartite graph where every vertex of the first set is connected to every vertex of the second set.

A **complete graph** is a graph whose vertices can be partitioned into two subsets V_1 and V_2 such that no edge has both endpoints in the same subset, and every possible edge that could connect vertices in different subsets is part of the graph. That is, it is a bipartite graph (V_1, V_2, E) such that for every two vertices $v_1 \in V_1$ and $v_2 \in V_2$, v_1v_2 is an edge in E. A complete bipartite graph with partitions of size $|V_1|=m$ and $|V_2|=n$, is denoted $K_{m,n}$; every two graphs with the same notation are isomorphic.

Networks and Graphs: Terminology

- Formally, a network is a graph is…
  - **G** = (**V**, **E**), an ordered tuple of two sets
  - **V** = $\{v_1, \ldots, v_n\}$, a set of unique nodes, and
  - E = $\{(v_i, v_j), \ldots\}$, a set of (un)ordered node tuples



Algorithms

- What is an algorithm?

- An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

- This is a rather vague definition. You will get to know a more precise and mathematically useful definition when you attend CS420.

- But this one is good enough for now…

Algorithms

- Properties of algorithms:

- **Input** from a specified set,
- **Output** from a specified set (solution),
- **Definiteness** of every step in the computation,
- **Correctness** of output for every possible input,
- **Finiteness** of the number of calculation steps,
- **Effectiveness** of each calculation step and
- **Generality** for a class of problems.

The Growth of Functions

- The growth of functions is usually described using the **big-O notation**.

- **Definition:** Let f and g be functions from the integers or the real numbers to the real numbers.
- We say that f(x) is O(g(x)) if there are constants C and k such that

- $|f(x)| \leq C|g(x)|$

- whenever $x > k$.

Big O notation

- The most common method and notation for discussing the execution time of algorithms is "Big O”.
- For the alphabetized dictionary the algorithm requires O(log N) steps.
- For the unsorted list the algorithm requires O(N) steps.
- Big O is the *asymptotic execution time* of the algorithm.

Formal Definition of Big O

- T(N) is O(F(N)) if there are positive constants c and N_0 such that $T(N) \leq cF(N)$ when $N \geq N_0$
  - There is a point N_0 such that for all values of N that are past this point, T(N) is bounded by some multiple of F(N).
  - Thus if T(N) of the algorithm is $O(N^2)$ then, ignoring constants, at some point we can *bound* the running time by a quadratic function of the input size.
  - Given a *linear* algorithm, it is *technically correct* to say the running time is $O(N^2)$. O(N) is a more precise answer as to the Big O bound of a linear algorithm.

Big O Examples

- $3n^3 = O(n^3)$
- $3n^3 + 8 = O(n^3)$
- $8n^2 + 10n * \log(n) + 100n + 10^{20} = O(n^2)$
- $3\log(n) + 2n^{1/2} = O(n^{1/2})$
- $2^{100} = O(1)$
- $T_{linearSearch}(n) = O(n)$
- $T_{binarySearch}(n) = O(\log(n))$

Algorithm classification

- Algorithms that use a similar problem-solving approach can be grouped together
- This classification scheme is neither exhaustive nor disjoint
- The purpose is not to be able to classify an algorithm as one type or another, but to highlight the various ways in which a problem can be attacked

P

$P = \{L \subseteq \{0, 1\}^* : \exists$ an algorithm A that decides L in p-time$\}$

$PATH \in P$

Algorithm A **verifies** language L if

$L = \{x \in \{0, 1\}^* : \exists\, y \in \{0, 1\}^*$ s.t. $A(x, y) = 1\}$

Can verify PATH given input $\langle G, u, v, k \rangle$ and path from u to v

PATH $\in$ P, so verifying and deciding take p-time

For some languages, however, verifying much easier than deciding

SUBSET-SUM: Given finite set S of integers, is there a subset whose sum is exactly t?

NP

- **NP is not the same as non-polynomial complexity/running time. NP does not stand for not polynomial.**
- **NP = Non-Deterministic polynomial time**
- NP means verifiable in polynomial time
- Verifiable?
  - If we are somehow given a ‘certificate’ of a solution we can verify the legitimacy in polynomial time

NP - hard

- What are the hardest problems in NP?

$$L_1 \leq_p L_2$$

- That notation means that L_1 is reducible in polynomial time to L_2 .
- The less than symbol basically means that the time taken to solve L_1 is no worse that a polynomial factor away from the time taken to solve L_2.

A problem (a language) is said to NP-hard if every problem in NP can be poly time reduced to it.

$$L' \leq_p L \text{ for every } L' \in NP$$

NP Complete problems/languages

- Need to be in NP
- Need to be in NP-Hard

If both are satisfied then it is an NP complete problem

Reducibility is a transitive relation.

If we know a single problem in NP-Complete that helps when we are asked to prove some other problem is NP-Complete

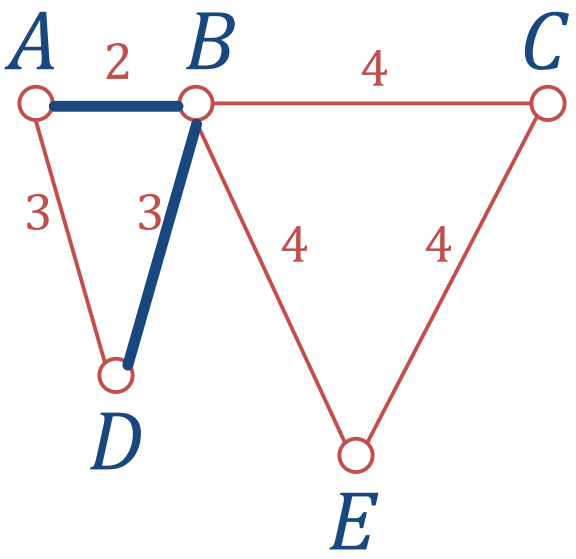
Assume problem P is NP Complete
All NP problems are reducible to this problem
Now given a different problem P'
If we show P reducible to P'
Then by transitivity all NP problems are reducible to P'

What is in NP-Complete

- For this course, we will axiomatically state that the following problems are NP-Complete
- SAT – Given any boolean formula, is there some assignment of values to the variables so that the formula has a true value
- 3-CNF SAT
- Actually any boolean formula can be reduced to 3-CNF form

Traveling salesman

- A salesman must visit every city (starting from city ***A***), and wants to cover the least possible distance
  - He can revisit a city (and reuse a road) if necessary
- He does this by using a greedy algorithm: He goes to the next nearest city from wherever he is



- From ***A*** he goes to ***B***
- From ***B*** he goes to ***D***
- This is *not* going to result in a shortest path!
- The best result he can get now will be ***ABDBCE***, at a cost of 16
- An actual least-cost path from ***A*** is ***ADBCE***, at a cost of 14

Hamilton Circuit

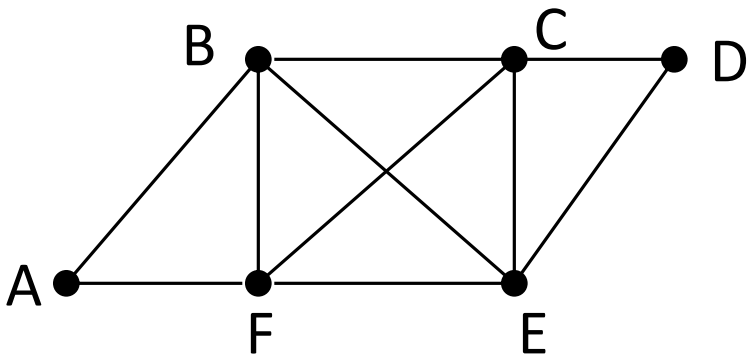
Hamiltonian Path is a path in a directed or undirected graph that visits each vertex exactly once. The problem to check whether a graph (directed or undirected) contains a Hamiltonian Path is NP-complete, so is the problem of finding all the Hamiltonian Paths in a graph. Following images explains the idea behind Hamiltonian Path more clearly. A **Hamilton circuit** in a graph is a circuit that visits each vertex exactly once (returning to the starting vertex to complete the circuit).

Example: Identifying Hamilton Circuits

a) $A \to B \to E \to D \to C \to F \to A$

b) $A \to B \to C \to D \to E \to F \to C \to E \to B \to F \to A$

c) $B \to C \to D \to E \to F \to B$



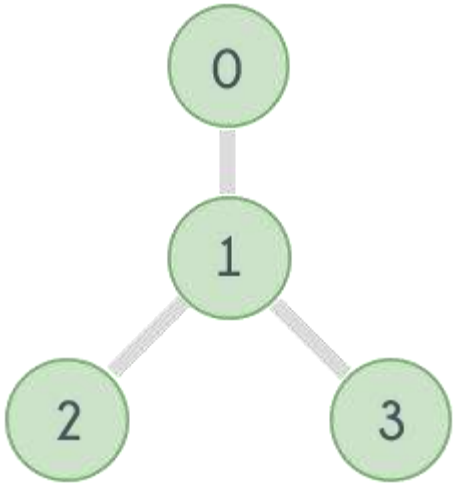


Fig. 1

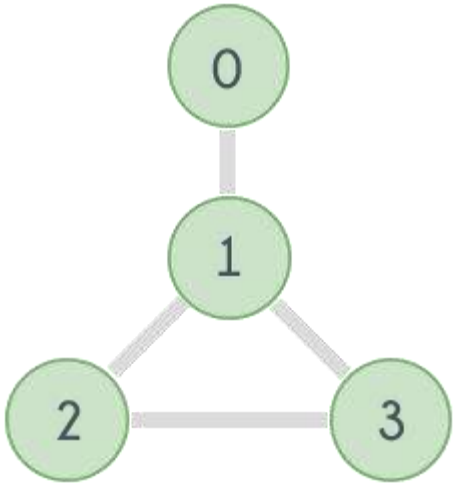


Fig. 2

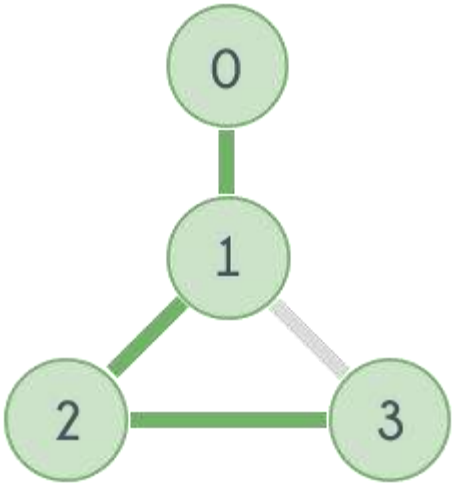


Fig. 3

Graph shown in Fig.1 does not contain any Hamiltonian Path. Graph shown in Fig. 2 contains two Hamiltonian Paths which are highlighted in Fig. 3 and Fig. 4

Following are some ways of checking whether a graph contains a Hamiltonian Path or not. A Hamiltonian Path in a graph having N vertices is nothing but a permutation of the vertices of the graph $[v_1, v_2, v_3, ...v_{N-1}, v_N]$, such that there is an edge between v_i and v_{i+1} where $1 \leq i \leq N-1$. So it can be checked for all permutations of the vertices whether any of them represents a Hamiltonian Path or not. For example, for the graph given in Fig. 2 there are 4 vertices, which means total 24 possible permutations, out of which only following represents a Hamiltonian Path.

0-1-2-3, 3-2-1-0, 0-1-3-2, 2-3-1-0