EE2008 Data Structure & Algorithm - Tutorial

• Tutor : Prof. Maode Ma
• Email: emdma@ntu.edu.sg
• Office: Rm. S2.2-B2-27
• Tel No. 67904385
• Website: http://www.ntu.edu.sg/home/emdma
Topic 1: Mathematics

1. Concepts
2. Questions
Concept: Notations

Basic Notation:
$N$: set of natural numbers \{1,2,\ldots\}
$R$: set of real numbers; $R \geq 0$: nonnegative real numbers

Numbers Definitions and Notations:
If $X$ is a finite set, $|X|$ denotes the number of elements in $X$.
floor: $\lfloor x \rfloor = \text{largest integer less than or equal to } x$.
$\lfloor 4.9 \rfloor = 4$
ceiling: $\lceil x \rceil = \text{smallest integer greater than or equal to } x$.
$\lceil 7.1 \rceil = 8$
Concept: Sequences

A finite sequence $a$ is a function from the set $\{0, 1, \ldots, n\}$ to a set $X$.
The sequence is typically denoted as $a_0, a_1, \ldots$, the subscript $i$ in $a_i$ is the index of the sequence.
An infinite sequence $a$ is a function from the set $\{0, 1, \ldots\}$ to a set $X$. The sequence is denoted $a_0, a_1, \ldots$, or an expression

$a$ is an **increasing sequence** if $\forall i, a_i < a_{i+1}$

$a$ is a **decreasing sequence** if $\forall i, a_i > a_{i+1}$

$a$ is a **non-increasing sequence** if $\forall i, a_i \geq a_{i+1}$

$a$ is a **non-decreasing sequence** if $\forall i, a_i \leq a_{i+1}$
Concept: Binomial Coefficient

Factorials:
\[ n! = n \times (n-1) \times \ldots \times 3 \times 2 \times 1 \]

Binomial Coefficient \[ \binom{n}{k} \]

Provides the number of combinations of selecting \( k \) elements from a set with \( n \) elements.

For \( n \geq k \geq 0 \), the number of \( k \)-element subsets of an \( n \) element set is given by

\[
\binom{n}{k} = \frac{n!}{(n-k)! \, k!}
\]
Concept: Polynomial

A polynomial of degree $n$ is a function of the form

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + c_{n-2} x^{n-2} + \ldots + c_2 x^2 + c_1 x + c_0$$

with $c_n \neq 0$. The numbers $c_i$ are called coefficients.
Concept: Logarithms

Recall: if $b^x = n$, $\log_b n = x$

Laws of logarithms: Suppose that $b > 0$, and $b \neq 1$, then

- $b^{\log_b x} = x$
- $\log_b (xy) = \log_b x + \log_b y$
- $\log_b \left(\frac{x}{y}\right) = \log_b x - \log_b y$
- $\log_b x^y = y \log_b x$

if $a > 0$ and $a \neq 1$, $\log_a x = \frac{\log_b x}{\log_b a}$

if $b > 1$ and $x > y > 0$, $\log_b x > \log_b y$
**Concept: Summation**

\[ \sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n \]

\[ \sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \ldots + n^2 \]

\[ \sum_{i=1}^{n} a = a + a + a + \ldots + a = an \quad \text{\(n\) times} \]

\[ \sum_{i=p}^{q} a = a + a + a + \ldots + a = a(q-p+1) \quad \text{\(q-p+1\) times} \]
Concept: Induction

- Mathematical induction can be used to prove a sequence of statements indexed by the positive integers. There are two steps.
- Basic step: show that the equation is true for $n = n_0$;
- Inductive step: assume that equation is true for $n = k, k \geq n_0$, and prove that it is true for $n = k + 1$. 
Topic 2: Array

1. Concepts
2. Questions
Concept: Array

• An array is an object consisting of a numbered list of variables, which are always numbered from zero or 1 in increments of one.

• Multi-Dimensional Arrays: A two-dimensional array is an array of arrays. A three dimensional array is an array of arrays of arrays.

• An array is expressed by $A[1..n,1..m,1..p]$, which indicates that the array $A$ is a three dimensional array with index $i$ ranging from 1 to $n$, $j$ from 1 to $m$, $k$ from 1 to $p$. 
Concept: Array

• An element of the array $A$ can be expressed by $a[i, j, k]$, which indicates that the element is indexed by $i$, $j$, and $k$. Once the three values are fixed, the element will be specified.
• The memory of a computer is essentially a one-dimensional array.
• The most natural way to implement a multi-dimensional array is to store its elements in a one-dimensional array.
Concept: Pseudocode

- Control flow
  if ... then ... [else ...]
  while ... do ...
  repeat ... until ...
  for ... do ...

- Algorithm declaration
  Algorithm name (arg, arg, ...)
  Input ...
  Output ...

- Calling another algo.
  Algo_called (arg, arg, ...)

- Return value
  return expression

- Expressions
  = Assignment
  == Equality testing
  $n^2$ Superscripts and other mathematical formatting allowed
Topic 3: Algorithm Analysis

1. Concepts
2. Questions
## Concept: Complexity Table

<table>
<thead>
<tr>
<th>Function</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>constant</td>
</tr>
<tr>
<td>( \log n )</td>
<td>logarithmic</td>
</tr>
<tr>
<td>( n )</td>
<td>linear</td>
</tr>
<tr>
<td>( n \log n )</td>
<td>( n \log n )</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>quadratic</td>
</tr>
<tr>
<td>( n^3 )</td>
<td>cubic</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>exponential</td>
</tr>
<tr>
<td>( n! )</td>
<td>factorial</td>
</tr>
</tbody>
</table>
Concept: Algorithm Analysis

- Analysis of an algorithm is a process to estimate the running time and the total or maximum memory space needed to execute the algorithm.
- The time needed to execute an algorithm is a function of the size of input.
- To analyze an algorithm with nested loops, it’s easier to start with the body of the inner-most loop.

- **Theorem:** For a nonnegative polynomial of degree $k$

  $$p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$$

  $$p(n) = O(n^k)$$
Concept: Asymptotic Notation

- Consider a non-negative function $f(n)$ for all integers $n > 0$. $f(n) = O(g(n))$, if there exists an integer $n_0$ and a constant $k > 0$ such that for all integers $n > n_0$, $f(n) \leq k(g(n))$. $g$ is an asymptotic upper bound for $f$.
- $f(n) = \Omega(g(n))$, If there exist constants $k_2$ and $N_2$ such that $f(n) \geq k_2g(n)$ for all $n \geq N_2$. $g$ is an asymptotic lower bound for $f$.
- $f(n) = \Theta(g(n))$, If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. $g$ is an asymptotic tight bound for $f$. 
Concept: Properties of Asymptotic Notation

- Transitivity: \( f(n) = \Theta(g(n)), \ g(n) = \Theta(h(n)) \) then \( f(n) = \Theta(h(n)) \);
- Reflexivity: \( f(n) = \Theta(f(n)) \);
- Symmetry: \( f(n) = \Theta(g(n)) \), if and only if \( g(n) = \Theta(f(n)) \);
- Transpose symmetry: \( f(n) = O(g(n)) \), if and only if \( g(n) = \Omega(f(n)) \);
- If \( f_1(n) = O(g_1(n)) \), \( f_2(n) = O(g_2(n)) \), then
  \[
  f_1(n) + f_2(n) = O(g_1(n) + g_2(n)).
  \]
- If \( f_1(n) = O(g_1(n)) \), \( f_2(n) = O(g_2(n)) \), then
  \[
  f_1(n) x f_2(n) = O(g_1(n) x g_2(n)).
  \]
- \( O(kf(n)) = O(f(n)) \);
- \( O(f(n) + g(n)) = \max(f(n), g(n)) \);
- \( O(f(n) x g(n)) = O(f(n)) x O(g(n)) \).
**Concept:** Useful Summations

\[
\sum_{i=1}^{n} (2i - 1) = n^2
\]

\[
\sum_{i=1}^{n} i(i + 1) = \frac{n(n + 1)(n + 2)}{3}
\]

\[
\sum_{i=1}^{n} i(i!) = (n + 1)! - 1
\]

\[
\sum_{i=1}^{n} i^2 = \frac{n(n + 1)(2n + 1)}{6}
\]

\[
\sum_{i=1}^{n} (-1)^{i+1} i^2 = \frac{(-1)^{n+1} n(n + 1)}{2}
\]

\[
\sum_{i=1}^{n} i^3 = \left[ \frac{n(n + 1)}{2} \right]^2
\]
Topic 4: Algorithm Analysis

1. Concepts
2. Questions
Concept: Recurrence

- A recurrence relation for the sequence $a_0, a_1, ...$ is an equation that relates $a_n$ to certain of its predecessors $a_0, a_1, ..., a_{n-1}$, with initial conditions to give values of a finite number of the terms in the sequence.
- A recurrence relation is used to describe the complexity of a recursive algorithm.
- To solve a recurrence for a sequence $\{C_n\}$ is to find a formula for $C_n$ expressed by initial conditions without $C_i$ involved.
- Iteration is a way to solve a recurrence relation.
Concept: Recurrence Solution

- One type of iteration is that the $nth$ term is given in terms of immediate preceding term $(n-1)th$ only.
- After each substitution, the subscript on the right side of the equation is reduced by 1.
- Another type of iteration is that $nth$ term is given in terms of preceding term $(n/b)th$, etc.
- After each substitution, the index on the right side is divided by $b^i$ with $i$ increasing 1.
- Continue substitution until the index can (or after further conversion) reach to the value to meet the initial conditions.
**Concept: Recurrence Example**

- The Fibonacci sequence, $f_0, f_1, \ldots$, is defined by the recurrence relation $f_n = f_{n-1} + f_{n-2}$, $n \geq 2$, with initial conditions: $f_0 = 0$, $f_1 = 1$.

- **Main Recurrence Theorem:**
  Let $a$, $b$, and $k$ be integers, $a \geq 1$, $b \geq 2$, and $k \geq 0$.
  If: 
  $$T(n) \leq aT(n/b) + f(n), f(n) = O(n^k),$$
  Then
  $$T(n) = O(n^k), \quad \text{if } a < b^k$$
  $$O(n^k \log_b n), \quad \text{if } a = b^k$$
  $$O(n^{\log_b a}), \quad \text{if } a > b^k$$
Topic 5: Stack, Queue & List

1. Concepts
2. Questions
Concept: Stack & Queue

- When an item is pushed into a stack, it is placed at the top of the stack;
- When an item popped, it is always the top item which is removed;
- The principle a stack follows is last-in, first-out or LIFO.
- In a queue, elements are added at one end and are removed from the other end.
- The principle a queue follows is first-in, first out or FIFO.
Concept: Stack Operations

Main functions:
- `stack_init( )`: Make the stack empty.
- `empty( )`: Return `true` if the stack is empty; otherwise, return `false`.
- `push(val)`: Add the item `val` to the top of the stack.
- `pop( )`: Remove the top (most recently added) item from the stack and no value is returned.
- `top( )`: Return the item most recently added to the stack, but do not remove it.
Concept: Queue Operations

Main functions:

- `queue_init( )`: Make the queue empty.
- `empty( )`: Return `true` if the queue is empty; otherwise, return `false`.
- `enqueue(val)`: Add the item `val` to the rear of the queue.
- `dequeue( )`: Remove the item from the front (least recently added) of the queue. No value is returned.
- `front( )`: Return the item from the front of the queue, but do not remove it.
Concept: List

- A singly linked list is a concrete data structure consisting of a sequence of linked nodes.
- To implement a linked list, a node structure with a data field `data` and a field `next` that references the next node in the list will be used. The `next` field of the last node has a special value null.
- There are three basic operations on the single linked list: Insertion, Deletion, and Visiting.
Concept: Operations

- **Insertion**: Insert a node after the node referenced by `pos`:
  Generate a node `temp`, assign a value to `temp.data`,
  `temp.next` is set to `pos.next`, `pos.next` is set to `temp`.

- **Deletion**: Delete the node referenced by `pos.next`:
  `pos.next` is set to `pos.next.next`.

- **Visiting**: Read the value in the `data` field of each node:
  The first node is referenced by `start`, read the `start.data`,
  `start` is set to `start.next`, Until `start = null`. 
Topic 6: Tree

1. Concepts
2. Questions
Concept: Tree

• A tree is an abstract model of a hierarchical structure, which consists of nodes with a parent-child relation
• Root: node without parent
• Internal node: node with at least one child
• External node (leaf): node without children
• Ancestors of a node: parent, grandparent, etc.
• Depth of a node: number of ancestors
• Height of a tree: maximum depth of any node
• Descendant of a node: child, grandchild, etc.
• Subtree: tree consisting of a node and its descendants
Concept: Binary Tree

- A binary tree is a structure defined on a finite set of nodes that either contains no nodes, or is composed of three disjoint sets of nodes: a root node, its left subtree and its right subtree, which are binary trees;
- To traverse a binary tree is to visit each node in some order;
  - Preorder: root—left—right;
  - Inorder: left—root—right;
  - Postorder: left—right—root.
Topic 7: Hash Table & Heap

1. Concepts
2. Questions
Concept: Hash Table

- A hash table for a given key type consists of a Hash function $h$ and an Array (called table) of size $N$;
- The hash function $h$ maps a key $x$ of a given type to an integer in a fixed interval $[0, N-1]$, which is called the hash value of key $x$ and is the index of the Array;
- A collision may happen in hashing if more than two keys have been hashed to the same index;
- The way to resolve is chaining, which puts all the elements with same index in a linked list. The cell $j$ contains a pointer to the head of the list of all stored elements hashed to $j$. 
Concept: Priority Queue

- A priority queue is an abstract data type allowing to insert an item with a specified priority and to delete the item having the highest priority;
- The elements in a priority queue are prioritized such that the element having the highest priority is always deleted first.
- A binary search tree has the following features: 1) A node’s left child must have a key less than its parent; 2) A node’s right child must have a key greater than its parent.
Concept: Heap

- A heap structure is a binary tree in which all levels have as many nodes as possible;
- A binary maxheap is a heap in which values are assigned to the nodes so that the value of each node is greater than or equal to the values of its children;
- An array is used to store a maxheap by putting the root in the first cell. The nodes are stored level by level, from left to right, following the root;
- The operations of the algorithm, which maintains a heap with deletion of the root from maxheap are: Replace the root by the node at the bottom level, farthest right; Swap it with its largest child until no children larger than it.
Concept: Siftdown Algorithm

```
siftdown(v, i, n) {
    temp = v[i]
    // 2 * i ≤ n tests for a left child
    while (2 * i ≤ n) {
        child = 2 * i
        // if there is a right child and it is
        // bigger than the left child, move child
        if (child < n && v[child + 1] > v[child])
            child = child + 1
        // move child up?
        if (v[child] > temp)
            v[i] = v[child]
        else
            break // exit while loop
        i = child
    }
    v[i] = temp // insert original v[i] in correct spot
}
```
Topic 8: Heap

1. Concepts
2. Questions
Concept: Heap

- A heap structure is a binary tree in which all levels, except possibly the last (bottom) level, have as many nodes as possible. On the last level, all of the leaves are at the left.
- A binary min-heap is a heap structure in which values are assigned to the nodes so that the value of each node is less than or equal to the values of its children (if any).
- A binary max-heap is a heap structure in which values are assigned to the nodes so that the value of each node is greater than or equal to the values of its children (if any). In a max-heap, the maximum value is at the root.
- A max-heap is “weakly sorted” in the sense that the values along a path from the root to a terminal node are in non-increasing order.
Concept: Heap

- An array is used to store a max-heap by putting the root in the first cell. The nodes are stored level by level, from left to right, following the root.
- The process to re-organize the data into a max-heap or min-heap is heapify.
- The algorithm to make a max-heap/min-heap is `heapify()`, which rearranges the data in the array $A$, indexed from 1 to $n$, so that it represents a heap.
- There are 2 operations on max-heap, deleting a node from and inserting a node into a max-heap.
Concept: Partition

- Partition divides an array into two parts and sizes of the two parts can range from nearly equal to highly unequal.
- The division depends on a particular element, the partition element selected.
- The partition algorithm partitions the array $A[i..j]$ by inserting $val = A[i]$ at the index $h$ where it would be if the array was sorted.
- When the algorithm concludes, the values at the indexes less than $h$ are less than $val$, and values at indexes greater than $h$ are greater than or equal to $val$.
- The algorithm returns the index $h$. 
**Concept: Quicksort**

- The quicksort follows the divide-and-conquer approach.
- **Divide:** Partition (rearrange) the array $A[p..r]$ into 2 possibly empty subarrays $A[p..h-1]$ and $A[h+1..r]$ such that each element in $A[p..h-1]$ is less than $A[h]$, which is in turn less than each element in $A[h+1..r]$. This partition procedure returns the index $h$.
- **Conquer:** Sort the two subarrays $A[p..h-1]$ and $A[h+1..r]$ by recursive calls to quicksort.
- **Combine:** Since the subarrays are sorted in place, no work is needed to combine them as the entire array $A[p..r]$ is sorted.
Topic 9: Binary Search

1. Concepts
2. Questions
Concept: Sequential Search

- The goal of the search is to find all records with keys matching a given search key.
- The simplest method for searching is to store the records in an array and look through the array elements one by one sequentially.
- The worst case of the sequential search is that the key appears in the last position of the array or it is not in the array.
- In the worst case, all elements in the array with \( n \) elements needs to be searched. It incurs complexity of \( O(n) \).
Concept: Binary Search

- Use to search for an item in a sorted array.
- Input: an array $L$ sorted in non-decreasing order.
- The binary search algorithm begins by computing the midpoint $k = (1+n)/2$;
- If $L[k] = \text{key}$, record is found; Otherwise, the array is divided into two parts of nearly equal size;
- If $L[k] < \text{key}$, continue search in the second part, Otherwise, continue search in the first part;
- Until key is found or $i > j$. 
Topic 10: Graph

1. Concepts
2. Questions
**Concept: Graph Presentation**

- **Adjacency Matrix**: an $n \times n$ matrix, where $n = \#$ of vertices, $A(i,j) = 1$ if $(i,j)$ is an edge. There are $n^2$ bits of space, which needs $O(n)$ time to find the degree of a vertex and/or vertices adjacent to a given vertex.

- An array is used to access the various linked lists, which are the adjacency lists. $Adj[i]$ is the reference to the first node in a linked list of nodes representing the vertices adjacent to the vertex $i$.

- Let $m = \#$ number of edges in the graph, $n = \#$ number of adjacency lists. Each edge $(i,j)$ is represented twice in the adjacency lists: $j$ appears once in $i$’s list and $i$ appears once in $j$’s list. Hence there are total of $2m$ nodes in the adjacency lists.
Concept: Degree Search Algorithm

Input Parameter: adj
Output Parameters: None

```java
degrees(adj) {
    for i = 1 to adj.last {
        count = 0
        ref = adj[i]
        while (ref != null) {
            count = count + 1
            ref = ref.next
        }
        println("vertex " + i + " has degree " + count)
    }
}
```
Topic 11: Graph Search

1. Concepts
2. Questions
Concept: Depth-First-Search

- The graph is represented using adjacency lists, adj[i]
- The idea of Depth-First-Search is to search as deep as possible
- The procedure is:
  - 1) visit vertex $v$, mark $v$ as visited.
  - 2) For each visited vertex, moves to next node in the adjacent list. For each unvisited vertex $u$ adjacent to $v$, execute Depth-First-Search on $u$.
- Overall complexity = $O(n+m)$, $m$ is the No. of nodes in the adjacent list.
Concept: **DFS**

```java
dfs(adj, s) {
    n = adj.last
    for (i = 1 to n) {
        visit[i] = false
    }
    dfs_recur(adj, visit, s)
}

dfs_recur(adj, visit, v) {
    visit[v] = true
    ref = adj[v]
    while (ref != null) {
        if (!visit[ref.data])
            dfs_recur(adj, visit, ref.data)
        ref = ref.next
    }
}
```
Concept: Breadth-First-Search

- The graph is represented using adjacency list, \( \text{adj}[i] \).
- The procedure of Breadth-First-Search is
  1) visit the vertex \( s \), mark \( s \) as visited
  2) add \( s \) to a queue \( q \).
  3) while \( q \) is not empty, i) return the front value of \( q \) and store it as \( v \), ii) visit each unvisited vertex \( u \) adjacent to \( v \), and add \( u \) to the queue \( q \), iii) remove the front element of \( q \)
Concept: BFS

bfs (\textit{adj}, \textit{s}) \{
\hspace{1em} n = \textit{adj}.last
\hspace{1em} for \hspace{1em} i = 1 \hspace{1em} \text{to} \hspace{1em} n \hspace{1em} \text{visit}[i] = \text{false}
\hspace{1em} visit[s] = \text{true} \hspace{1em} // \hspace{1em} \text{visit} \hspace{1em} \textit{s}
\hspace{1em} q.\text{enqueue}(s)
\hspace{1em} \text{while} \hspace{1em} (!q.\text{empty}()) \{
\hspace{2em} v = q.\text{front}()
\hspace{2em} ref = \textit{adj}[v]
\hspace{2em} \text{while} \hspace{1em} (\text{ref} != \text{null}) \{
\hspace{3em} \text{if} \hspace{1em} (!\text{visit[ref.data]}) \{
\hspace{4em} \text{visit[ref.data]} = \text{true} \hspace{1em} // \hspace{1em} \text{visit} \hspace{1em} \text{ref.data}
\hspace{4em} q.\text{enqueue}(\text{ref.data})
\hspace{3em} \}
\hspace{2em} ref = \text{ref}.\text{next}
\hspace{2em} \}
\hspace{1em} q.\text{enqueue}()
\hspace{1em} \}
\}
\}
Concept: MST

• A graph is called a tree if it is connected without any cycle.
• A spanning tree of a graph G is a subgraph of G that is a tree and includes all vertices of G; Every connected graph possesses (at least) one spanning tree.
• MST is a spanning tree of a weighted graph with minimum total edge weight
Topic 12: Greedy Algorithms

1. Concepts
2. Questions
**Concept: Dijkstra’s Algorithm**

- Single-source shortest path problem is to find the shortest paths from a given vertex $s$ to all the other vertices.
- Dijkstra’s Algorithm solves the single-source shortest path problem with the following two steps.
  1) Add a minimum edge starting from vertex $s$ to an empty SPT
  2) If the number of the edges of SPT is less than $n-1$, keep growing SPT by repeatedly adding edges which can extend the paths from $s$ in SPT as short as possible.
Concept: MST

- A graph is called a tree if it is connected without any cycle;
- A spanning tree of a graph G is a subgraph of G that is a tree and includes all vertices of G; Every connected graph possesses (at least) one spanning tree.
- MST is a spanning tree of a weighted graph with minimum total edge weight;
- MST can be build up by the Kruskal’s algorithm;
- Kruskal’s algorithm: 1) begins with all vertices and no edges in the MST; 2) repeatedly adds an edge with minimum weight without making a cycle to the MST; 3) stops when n-1 edges have been added.
Concept: Topological Sort

• A directed acyclic graph (DAG) is a digraph that has no directed cycles.

• A topological sorting of a DAG is an ordering of the vertices such that in that list, \( v_i \) precedes \( v_j \) whenever a path exists from \( v_i \) to \( v_j \).

• A topological sort of a DAG gives an order in which the vertices can be visited while still satisfying the constraints.
Concept: Topological Sort

- Given a number of vertices to visit, there are often a number of constraints on the visit of each vertex.
- Vertex A must be visited first before the visit of B can start.
- These vertices together with the constraints form a DAG.
- To generate a topological sort, it starts with a vertex with an in-degree of zero.
- At any point to choose next, it could form a different topological sort. They are not unique.