* Implementation levels of data structures are the
  + **Abstract level**
  + **Application level**
  + **Implementation level**
* A binary search tree whose left subtree and right subtree differ in height by at most 1 unit is called a
  + **AVL Tree**
* **Implementation level** is where the model becomes compatible executable code.
* True about abstract data types
  + **It exports a type**
  + **It exports a set of operations**
* **Operations, Storage Structures and Algorithms** are data structures.
* A **tree** is a non-linear data structure.
* Herder node is used as sentinel in
  + **Binary trees**
* What data structure is used in BFS of a graph to hold nodes
  + **Queue**
* A **graph** is a nonlinear type.
* To represent hierarchical relationship between elements, use a
  + **Tree**
* A directed graph is **strongly connected** if there is a path from each vertex to every other vertex in the digraph.
* In the **depth first** traversal we process all of the vertex’s descendants before we move to an adjacent vertex.
* True
  + Network is a graph that has weights or costs associated with it
  + An undirected graph which contains no cycles is called a forest.
  + A graph is complete if there is an edge between every pair of vertices
* A graph is **bipartite** if the vertices can be split into two sets V1 and V2 such there are no edges between two vertices of V1 or two vertices of V2.
* There is an extra element at the head of the list called a
  + **Sentinel**
* A graph is a collection of nodes, called **vertices** and lone segments called arcs or **edges** that connect pair nodes.
* A **network** and **weighted** **graph** is a graph that has weights of costs associated with its edges.
* The property of a binary tree is
  + **The right subtree can be empty**
* True
  + Nodes that are not root and not leaf are called as internal nodes
  + A node is a parent if it has successor nodes
  + A empty tree is also a binary tree
  + In strictly binary tree, the out-degree of every node is either 0 or 2.
* False
  + A node is a child node if out of degree is one
  + The degree of root node is always zero
* Any node is the path from the root to the node is called
  + **Ancestor node**
* A **directed** **tree** is an acyclic digraph, which has only one node with in-degree 0, and other nodes have in-degree 1.
* **Binary Tree** is a directed tree in which out-degree of each node is less than or equal to 2.

CHAPTER 9 – Skip Lists

Skip Lists

* Is a simple randomized data structure that can be used to implement the ordered map ADT or a dictionary efficiently
* Randomization in data structures and algorithm design usually produce simple and efficient results
* Fast search is made possible by maintaining a linked hierarchy of subsequences, with each successive subsequence skipping over fewer elements than the previous one

Graph Terminology

* Graph Theory – the study of graphs.
* Graph – A mathematical structure that represents relationships among entities in the real world
  + Consists of a non-empty set of vertices, nodes or points and a set of edges that connect the vertices
* Directed edge
  + Ordered pair of vertices
  + First vertex is the origin
  + Second vertex is the destination
* Directed graph – all edges are directed (arrows)
* Undirected edge
  + Unordered pair of vertices
  + Move both directions between vertices
* Undirected graph – all the edges are undirected (solid lines)
* Weighted graph – each edge has an associated numerical value called the weight of the edge
  + Weights could be costs, distances
* End vertices (endpoints) – Two vertices joined by an edge
  + If an edge is directed, the first endpoint is called its origin and the other is the destination of the edge
* Adjacent vertices – if there is an edge that ends at the other vertex (also called neighbors)
* Adjacent edges – if they are connected to the same vertex (also called neighbors)
* Incident – an edge that joins two vertices
* Degree – is the number of incident edges
* Loop – edge that connected a vertex to itself
* Outgoing edges – the directed edges whose origin is that vertex A->B
* Incoming edges – the directed edges whose destination is that vertex B<-A
* In-degree – the number of incoming edges
* Out-degree – the number of outgoing edges
* Collection – group of edges
* Parallel edges
  + two undirected edges that have the same vertices
  + two directed edges that have the same origin and destination
* Self-loop
  + Edges that connects a vertex to itself
* Simple graph – if it does not have parallel edges or self-loops
* Path
  + Sequence of alternating vertices and edges
  + Begins with a vertex
  + Ends with a vertex
  + Each edge is preceded and followed by its endpoints
  + Simple path is all its vertices and edges are distinct
* Directed path
  + All edges are directed
  + Traversed along their direction
  + Each edge is preceded and followed by its endpoints
* Cycle
  + Circular sequence of alternating vertices and edges
  + Each edge is preceded and followed by its endpoints
  + Simple cycle if all its vertices and edges are distinct
* Directed cycle
  + All edges are directed
  + Traversed along their direction

Graph Traversals

* Systematic procedure for exploring a graph by examining all of its vertices and edges
  + Process of visiting each vertex in the graph exactly once
* Both BFS and DFS result in a spanning tree
* Subgraph
  + The vertices of A are a subset of the vertices of B
  + The edges of A are a subset of the edges of B
  + Spanning subgraph – of B is a subgraph that contains all the vertices of B
* Connected – if there is a path between every pair of vertices
* Connected component – of a graph is a maximal connected subgraph
* Tree (free) – is an undirected graph such that
  + Its connected
  + Has no cycles
* Forest – an undirected graph without cycles
  + Connected components of a forest are trees
* Spanning tree – a spanning subgraph of that tree
  + Not unique unless the graph is a tree
  + Have applications to the design of communication networks
* Spanning forest – a spanning subgraph that is a forest
* Discovery edges – edges used to discover new vertices
* Back edges – edges that lead to already discovered vertices
* Cross edges – the edges that lead to already discovered vertices

Directed graph

* Digraph – a graph whose edges are all directed
* Scheduling – edge (a, b) means task A must be completed before B can start
* Directed cycle – all edges are traverse according to their respective cycles
* Acyclic graph – graph has no directed cycles
* Traversals by traveling edges only along their direction
* Discover or tree edges – A DFS on a directed graph partitions the edges of a graph reachable from the starting vertex into discovery or tree edges
* Back edged – connected a vertex to an ancestor in the DFS tree
* Forward edges – connects a vertex to a descendant in the DFS tree
* Cross edges – connects a vertex to a vertex that is neither an ancestor or descendant
* Directed acyclic graphs
  + Digraphs without directed cycles
  + Prerequisites between courses for a degree program
  + Scheduling constraints between tasks on a project

Floyd Warshall

* Shortest path between all pairs of vertices, negative cycles not allowed, negative edges allowed

Weighted graphs

* There are two types of weighted graphs
  + The vertices are assigned a weight
  + The edges are assigned a weight
* Each edge has an associated numerical value, called the weight of the edge
  + May represent distance costs
* Priority adjacency lists
  + Weight graphs using priority adjacency lists can implemented priority queues
  + Edges are removed in increasing order

First Fit

* The first of these is called first fit. The basic idea with first fit allocation is that we begin searching the list and take the first block whose size is greater than or equal to the request size, as illustrated in Example 9.3. If we reach the end of the list without finding a suitable block, then the request fails. Because the list is often kept sorted in order of address, a first fit policy tends to cause allocations to be clustered toward the low memory addresses. The net effect is that the low memory area

Next Fit

* If we want to spread the allocations out more evenly across the memory space, we often use a policy called next fit. This scheme is very similar to the first fit approach, except for the place where the search starts. In next fit, we begin the search with the free block that was next on the list after the last allocation. During the search, we treat the list as a circular one. If we come back to the place where we started without finding a suitable block, then the search fails.

Best Fit

* In many ways, the most natural approach is to allocate the free block that is closest in size to the request. This technique is called best fit. In best fit, we search the list for the block that is smallest but greater than or equal to the request size. This is illustrated in Example 9.5. Like first fit, best fit tends to create significant external fragmentation, but keeps large blocks available for potential large allocation requests.

Worst Fit

* If best fit allocates the smallest block that satisfies the request, then worst fit allocates the largest block for every request. Although the name would suggest that we would never use the worst fit policy, it does have one advantage: If most of the requests are of similar size, a worst fit policy tends to minimize external fragmentation.

**Mark Phase**  
When an object is created, its mark bit is set to 0(false). In the Mark phase, we set the marked bit for all the reachable objects (or the objects which a user can refer to) to 1(true). Now to perform this operation we simply need to do a graph traversal, a [depth first search approach](http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/) would work for us. Here we can consider every object as a node and then all the nodes (objects) that are reachable from this node (object) are visited and it goes on till we have visited all the reachable nodes.

* Root is a variable that refer to an object and is directly accessible by local variable. We will assume that we have one root only.
* We can access the mark bit for an object by: markedBit(obj).

**Sweep Phase**  
As the name suggests it “sweeps” the unreachable objects i.e. it clears the heap memory for all the unreachable objects. All those objects whose marked value is set to false are cleared from the heap memory, for all other objects (reachable objects) the marked bit is set to false.  
Now the mark value for all the reachable objects is set to false, since we will run the algorithm (if required) and again we will go through the mark phase to mark all the reachable objects.