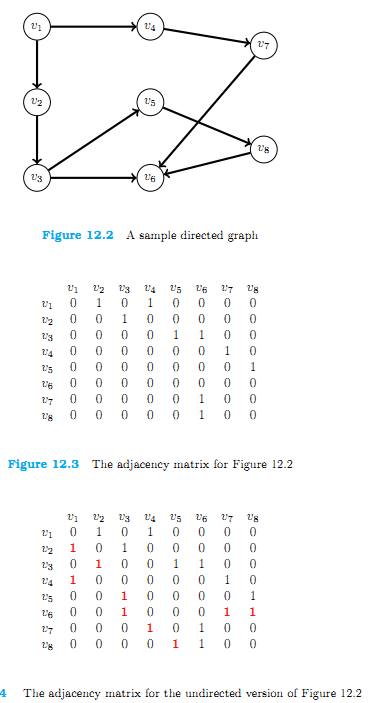
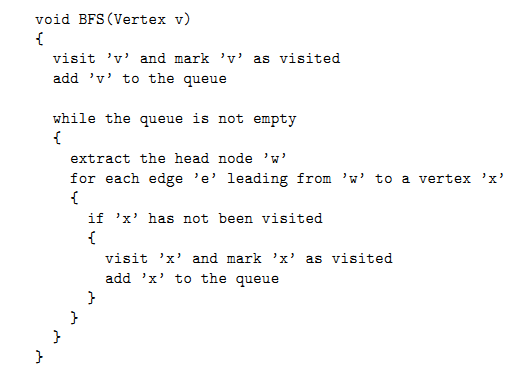
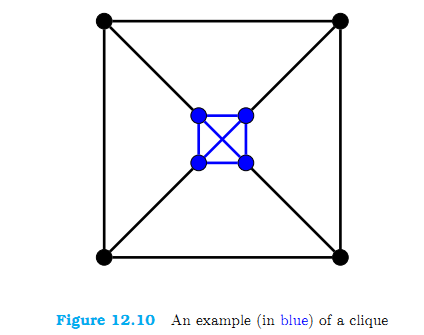
1. Chapter 11
   1. Associative matching
      1. Measure efficiency by number of probes in a search
      2. Searching though for loop causes two problems
         1. Requires write loop through array and prone to error
            1. Iterators allows walk through data directly instead of through subscripts
         2. Requires linear time – find key on average N/2 probes of N length
      3. Goal is to search in log number of probes
   2. Indexing for search
      1. Index- sorted array of keys for file then use binary search in log time
      2. Comes at a price of NlogN time to initially sort data and logN to maintain sort
   3. Binary trees
      1. Often we need dynamic data structure
      2. Binary search tree nodes hold key values for these two properties
         1. For any given node, the key values for all nodes in the left of the node are less than or equal to the value of that key
         2. For any given node. The key values in the right sub tree of that node are greater than or equal to value of that key
      3. Can choose to disallow equality of keys or not depending on need
      4. We get the best performance if the tree is balanced or nearly balanced because we can store the max number of entries 2^n when measured against the depth of the tree
      5. If BST is all on right side it is essentially a linear list and worst case scenario
   4. In order traversal and binary search trees
   5. Sophisticated search trees
      1. Balanced binary tree is most efficient data structure for performing searches
      2. The key to this is the promise to eliminate half of the remaining nodes after each probe
      3. Tree balancing – rotating sub trees so as to turn a three node linear sequence into a parent node with two children – to balance and unbalanced tree
   6. Hashing
      1. Mathematical mapping from the value of a key to s subscript in an array with the exception that the hash values will provide a reasonably random scattering of the data entries across the array in which the records will be stored
      2. One method to generate pseudorandom numbers is mixed linear congruential generator
         1. Computes from an integer n a mapped pseudorandom number f(n) as f(n) = (A\*n + B) where p is a prime A is chosen multiplier and B is a well-chosen addend
      3. Hah collision – occurs when for example different input SSN’s would map under the hash function to the same output value s.
         1. Easiest way to deal with it is take value to caused collision and store in next available storage value
      4. To get a good hash need load factor to not be large
         1. The quotient of the number of entries to be stored by the total number of possible storage locations
         2. If storing N hashed entries in an array that is only N long, need almost perfect hash function to avoid lots of collisions
   7. Random numbers and hash functions
      1. A sequence of numbers is said to be pseudorandom if it passes an accepted set of tests that would be passed by truly random numbers
         1. Statistics tests need to fail
   8. Cryptographic hash functions
      1. Password is encrypted and then when password is entered , that is encrypted and the two encryptions should match up – they never get decrypted
      2. Hash functions cannot be produced backwards for a truly good hash function
   9. Random numbers in java
      1. Math.random provides a wrapper for the random class
         1. Generated by linear congruential method
   10. Java collections framework
       1. Implement standard structures for use in storing and retrieving data
       2. Four interfaces that are useful
          1. The set list and queue interfaces that are part of the collection interface
          2. Map interface technically not a collection
             1. A collection represents a group of objects known as its elements.
       3. Set interface – defines a structure that is modeled after the mathematical concept of a set
          1. Two properties of a set
             1. First duplicate entries are not allowed
             2. The elements in a set are unordered so the following are all the same set: [a,b,c] [b,c,a] c,a,b]….
       4. List interface
          1. Array List has all properties of one dimensional array but also has dynamic addition and removal of entries and doesn’t need the number of entries specified beforehand.
       5. Second List structure is the Linked List interface that implements a linked list.
       6. Map interface
          1. A map structure that resembles a mathematical function
          2. Maps key values to data values
   11. The java Hash Set
       1. Built-in implementation of a hash table
       2. Permits one to implement a set with a hash function for constant tie access to the set
       3. Allows for storing the existence of a key in the set but not of the key itself for retrieval of a record. –Only to know if an item is added, removed, or if list is empty – need a Tree Map to store actual values.
       4. Iterator returns a list of keys in the hashset but not in any particular order – easy to compute but hard to invert – going backwards from the hashed value to the original data element is hard
   12. Spell checker program
       1. Uses a hash table to store all words to check against when spell checking
       2. Can use two arrays of hash sets to double to probability of not colliding when creating hash functions
   13. The java Tree Map
       1. Underlying structure is a red-black tree
       2. A tree map contains two variables that are declared:
          1. Tree Map<variable1, variable2> myTreeMap;
       3. Put method stores a value with a key assigned to it
       4. Can use an iterator of the key set to iterate over the Tree Map
       5. Containskey and containsValue checks if the key or value are contained in the treemap
       6. Higherkey and higherEntry method takes a key as parameter and return either the key or the key value mapping for the least key that strictly greater than the parameter or else null
   14. Digression on documentation
       1. There are no methods to traverse down child or parents only to access keys and values and you must create your own methods and documentation
   15. The java treeset, hashmap, and linkedlist
       1. Hashset – allows one to store a record of the existence of a given key with single probe access but does not allow one to retrieve the keys in sorted order
       2. Hashmap – a map in that it stores a record not just the existence of a key for that record with single probe access but does not allow one to retrieve the records in sorted order by key
       3. Treemap – a map that stores a record retrievable by key in sorted order by keys. Access is guaranteed to be in log time in a balanced tree
       4. Tree set – a set that stores the existence of keys but the keys cannot be retrieved in sorted order. Access is in log time
       5. Linked list – is a list structure similar to the array list but one which implements the insertion, deletion, and traversal mechanisms necessary for a doubly linked list.
2. Chapter 12 – graphs
   1. Intro
      1. A graph as a pair G=(V,E) compromising a set V=<vi> o nodes or vertex and a set E=<vi, vj> of edges or arcs with each edge being a pair of nodes
      2. If an edge ek=<vi,vj> exists, we say that node vi is connected to node vj and that edge ek is incident to node vi and node vj
      3. Graphs can be directed or undirected
         1. A graph that has even one edge (A,B) that must be interpreted as from node A to node B then the graph is directed.
      4. There can exist graphs as well as multigraphs in which more than one edge joins two nodes
      5. Most problems with graph have to do with
         1. finding a path from one node to another with the ability to find the shortest path,
         2. the ability to cycle from a node through a graph and back to itself,
         3. and the ability to find the connected components of a graph
      6. if the edges have edge weights assigned to them, example a flow capacity, then the problems posed often become problems not of the existence of a path between nodes but of determining the path with the maximum capacity.
      7. Local-global problems
         1. Finding a path from one node to another is locally to only the info about the current node and its connections to its neighbors
            1. Only looking locally while ignoring the overall shortest distance from weight capacities can cause longer searches
   2. Adjacency matrices and linked lists
      1. Two basic data structures for storing the information about a graph
         1. Store info about every possible edge even if the edge doesn’t exist
         2. Store info only about the edges that do exist but is inefficient if most edges do not exist – impractical for huge graph such as internet graph
      2. The first option takes up more storage for each edge but if the number of actual edges is small compared to the number of possible edges then it can be more efficient
   3. Adjacent matrices
      1. The simplest data structure for storing the representation of a graph
      2. A graph with N vertices V=<vi : I = 1,…,N> we want a matrix of N rows and N columns.
         1. In location row I column j – put a 0 if not edge from vertex vi to vj and put a 1 if the edge does exist
      3. 
      4. Advantages
         1. Code required to do X for all edges leading out from vertex V is easy – a loop over the row for X with a test for the 1 value and execution of the code for Y in all cases in which the test comes up true
         2. Matrix is a simple 2 dimensional array
         3. Retrieval of the one or zero that indicates the presence or absence of an edge is the simple indexing a[i][j] into the adjacent matrix
      5. Disadvantages
         1. Storage and processing requirements for a sparse graph
         2. If only 9 non zeros exist in a 8\*8 graph (64 entries) then there is a lot of wasted space – need to store N^2-N zeros and only N ones and as graph matrix gets larger, the density (fraction of non-zeros) decreases
         3. The large the graph the more likely to be sparse
         4. In the case of very sparsely connected graphs, although code for traversing a row to find all edges connected to given vertex is easy, it is also computationally wasteful
            1. Can mitigate some of the wasted time by having a separate location for each row and for each column that would hold the number of non-zero entries – number of edges for that particular row/column allowing to stop process when reaching all nonzero
      6. Summary
         1. Adjacency matrix is easy to code but it is wasteful in both time and space if the graph is very sparsely connected
   4. Sparse graphs and sparse matrices
      1. Definition will be recursive – a matrix or graph is considered sparse of the techniques for dealing with sparse matrices or graphs efficiently happen to be efficient on that particular matrix or graph
   5. Linked lists for sparse graphs
      1. Techniques used in sparse matrix is to deal with only those entries that have to be there – the edges that actually exist between nodes rather than with the edges that could exist
      2. Instead of storing adjacency matrix of both connected and not connected info on all nodes, we store only info that indicates where the edges really do exist.
      3. Can represent using array of linked lists
         1. Each entry in the array would be a linked list of the column (nodes) for which an edge existed.
         2. Each entry in the linked list would contain as the data payload the node number (column number) and a link to the next nonzero
         3. Simplest and most naïve way in which to store a sparse graph for later processing = inflexible and doesn’t have all info needed to process on graph
         4. What is stored are lists of edges leading out from a given node – if computation need is to traverse graph from a node to nodes to which it is connected, might be sufficient representation
            1. Very cumbersome representation if need to find all the edges leading into a particular node
         5. Can have each nonzero entry itself be a node in two doubly linked lists one going across each row and one going down each column
         6. If the fraction of nonzero is small enough, the extra storage cost of storing a linked list or several linked lists is less than the cost of storing a very large number of zero entries
         7. Sparsely connected graphs – cost of traversing the list of edges for a given node is proportional to the number of edges not proportional to the number of nodes in the graph
   6. Middle ground helped by java
      1. Most processing on a graph will be done on rows of matrix with some on columns
      2. Can use a java array list as middle ground
         1. Array for the rows to permit efficient row processing without some overhead of traversing linked lists of nodes
   7. Breadth first and depth first search
      1. More complicated in graphs than in trees because cycles can exist
      2. For DFS
         1. Start at vertex A and pick any path from A and process in the order in which we labeled the vertices so the first move will be to vertex B
         2. Mark the nodes we have visited until reaching a marked node then need to back up until reaching a node with an unmarked path to a node.
      3. BFS
         1. Sweep across the edges from a given node but instead of recursion stack to keep the place, use a queue to list the nodes that we have not yet processed
         2. Visits node A-Mark as visited-insert A into queue
         3. Take A from the head of the queue
         4. Process edges A to B, D, and G in that order visiting these nodes and marking the nodes as visited and inserting them into queue
         5. Take B from head of the queue
         6. Process from B to C and E visiting the nodes and marking them and inserting them into queue which is now Q= <D,G,C,E>
         7. BFS algorithm terminates when the queue is drained
      4. 
      5. Need to minimize the number of times we examine a node – pass through – and then visit if necessary.
      6. Theorem: the BFS and DFS algorithms perform node examinations O(abs(E)) times where abs(E) is the cardinality of the set E of edges in the graph
   8. Graph problems
   9. NP completeness
      1. some problems on graphs are simple and some similar ones are very difficult
      2. the Class P
         1. a class is said to be in the P class if an algorithm exists that runs in polynomial time p(N) in the input size of the problem – which solves the problem for any inputs
         2. ex-claim that addition of two b-bit integers can be done in fewer than p(b) = 2b Boolean operations on bits
            1. if line up bits for standard addition algorithm we have b pairs of bits – for each pair we have one addition of two bits plus possibly one more addition of two bits in order to add in the carry from the previous bit addition
            2. this is at most 2b bit operations
      3. the Class NP
         1. the set of problems for which we can check a putative solution in polynomial time
         2. ex- Boolean satisfiability – SAT –
            1. given a Boolean expression consisting of variables And, Or, Not and parenthesis, there is assignment of true and false to variables that will make the entire eval to true
         3. (a OR b OR c) AND (b OR c)
            1. Satisfies if a=false, b=true, and c=false
         4. With n variables in a SAT expression, there are 2^n possible assignments of true and false that could be the solution
      4. NP completeness
         1. P contains all problems for which we found polynomial time algorithms for their solution
         2. NP contains problems for which we can check a putative solution in polynomial time
         3. NP complete if it is in the class NP and if any other problem in the class NP can be reduced to it in polynomial time
            1. Can transform instance of SAT into instance of Hamiltonian cycle problem in a polynomial number of cycles
         4. The list is essentially the list of the toughest open problems in all algorithm and comp science.
         5. Biggest problem being question of whether the class P is the same as the class NP
   10. Eulerian cycles
       1. Path/cycle is a path/cycle that visits each edge exactly once
       2. Theorem: a graph G= (V,E) has an eulerian cycle if and only if the graph is connected and every node has even degree
   11. Hamiltonian cycles
       1. Visits each node exactly once – must return to the originating node as it traverses last edge – NP complete
   12. Shortest path
       1. Path – a sequence of edges that connect nodes
       2. If the edge weights were distance in cities then the shortest path between two cities on the map is the problem
       3. Best known and shortest algorithm running O(abs(V^2)) time
   13. Traveling salesman
       1. NP-hard
       2. Graph is a set of cities as nodes and the distances between every pair of cities as edges weights with all edges considered possible
       3. Problem is to find the path from a given node that traverses every node in the graph with the minimum total distance traveled
          1. Simplest and purest form is a complete graph
   14. Cliques
       1. Clique in a graph G=(V,E) is a sub graph G’=(V’,E’) with V’ U V and with E’ U E such that every for every pair of vertices in V’, there exists an edge E’ connecting those vertices,
       2. 
       3. Maximal clique problem is the problem of finding a clique of maximal size in a given graph – ex – finds the largest subset of people on face book all of whom have friended each other in pairs.
          1. Ex2 – find the largest subset of computers in a network all of which are connected in pairs a high speed connection
   15. Connected components
       1. Ex the connected component of you on face book is the transitive closure of your friends and their friends and the friends and etc
       2. Finding the connected components of a graph is not hard-DFS or BFS
          1. BFS – greedy algorithm – start at any given node connect nodes adjacent to starting point and include these in component – perform the same on the new components
   16. Spanning Trees
       1. Number of problems involving graphs in which it is necessary to create a spanning tree for the graph
       2. Spanning tree – a sub graph of a graph that is a tree ( and no cycles) and that contains all the nodes in the original graph
          1. Collection of mobile communication devices might need to set up a conference call link along all the devices in the set
          2. If each device is connected to a single fixed tower – then a spanning tree would be needed among the fixed towers in order to link all the devices together
3. Chapter 14
   1. Utilities
      1. Standard FileUtils class with static methods for opening/closing scanners and printwriters and check for errors
   2. Labeling trace output
      1. classLabel variable in each class to track which class is executing what
   3. naming conventions
      1. naming exception with the word exception first so that all exception classes are together in a folder
      2. sort names of instance variables and methods to help with searching
      3. sort variables based to complexity of data type and then alphabetically
      4. Boolean-int-long-float-double-string-inherent java types- own java types
   4. Use of this in java
      1. Anything not prefixed with this in the name is local to method and is easier to distinguish from class variables
   5. Labeling closing braces
      1. Comment closing brackets with the leading line of code from the opening statement ex - // public void method();
   6. Keep it simple
   7. Double equals
      1. Constant on the left hand side of an equals method or ==
      2. Helps compiler find errors better than the other way
4. Chapter 14 – jargon terms