CS 213
Data Structures
Instructor: Milind Sohoni
Slot 1: Mon. 8:30, Tuesday 9:30, Thursday, 10:30.

4 Engines: NPTEL, Moodle, Teams, SAFE
TAs: Ankit Kumar Misra, Pooja Verma, Vishal Pramanik, Khushi Kapadia

Course Plan

We will be following a "flipped classroom" mode and use the lectures of Prof. Naveen Garg, from IIT Delhi as the base. The main link is:

https://nptel.ac.in/courses/106/102/106102064/

The topics list is as follows:
Modules / Lectures

Data Structures And Algorithms (link)

Our general strategy Computer Science and Engineering - Data Structures And Algorithms will be to introduce the material of the week in the Tuesday class, to discuss any difficulties in the Thursday class and to solve the accompanying questions in the Monday class of the subsequent week. Eventually, we will hold small quizzes in the Monday class.

Week 0 (26th July): Introduction.

Week 1 (27th July-2nd August)

- ***Introduction to Data Structures and Algorithms***
  Algorithms and data, size of data, running time, pseudo code, insertion sort and its running time, best case, worst case, average case, order notation.

- ***Stacks***

Abstract Data Types, Dynamic Set, The Stack and its rules, Array Implementation, An application - Span of daily stock prices, an array implementation O(n^2) and a stack implementation O(n).

- ***Queues and Linked Lists [0:40 minutes]***
  The Queue as an abstract data type, Its implementation as circular array, the one-sided linked list operation, doubly linked list.

Questions.

1. What is the time complexity of binary addition and multiplication? How much time does it take to do unary addition?
2. If f(n)<=F(n) and G(n)=>g(n) (in order sense) then show that h(n)=f(n) / G(n)<=F(n) / g(n).
3. There is a stack of dosas on a tava, of distinct radii. We want to serve the dosas of increasing radii. Only two operations are allowed: (i) serve the top dosa, (ii) insert a spatula (flat spoon) in the middle, say after the first k, hold up this partial stack and flip it upside-down and put it back. Design a data structure to represent the tava, input a given tava, and to produce an output in sorted order. What is the time complexity of your algorithm?
4. One problem is to check if a string of opening and closing brackets of various types is a valid (balanced) bracket sequence. For example {(){}[[]]} and ([[[]]]{}) are valid, that is balanced, while (){) and ())) are invalid (unbalanced). Write a stack based algorithm to check if the brackets are balanced or not.
5. *The mess table queue problem:* There is a common mess for K hostels. Each hostel has some $N_1,...,N_k$ students. These students line up to pick up their trays in the common mess. However, the queue is implemented as follows: If a student sees a person from his/her hostel, she/he joins the queue behind this person. This is the “enqueue” operation. The “dequeue” operation is as usual, at the front. Think about how you would implement such a queue. What would be the time complexity of enqueue and dequeue? Do you think the average waiting time in this queue would be higher or lower than a normal queue? Would there be any difference in any statistic? If so, what?

Week 2 (3rd August-9th August)

- ***Dictionaries***
  The dictionary as an ADT. Key-Element pairs. Ordering of Keys. The basic operations of Find, Insert and Delete. Array implementation and binary search. Its drawbacks. Sparse keys and the hash table. Collisions, efficiency and load factor alpha. Chaining.

- ***Hashing [0:40 minutes]***
  Hashing functions, Linear hash tables.

- ***Trees***
  Basic notation and examples. Binary trees, ordering of nodes. Recursive definition. Height, number of nodes and basic statistics. Implementing trees. Unbounded branching.

Questions

1. Given that k elements have to be stored using a hash function with target space n, what is an estimate of the probability of a collision?
2. Work out the solved examples for chaining as well as linear hashing for the modulo N hash function.
3. Suppose you want to store a large set of pairs of numbers (a_i ,b_i), for example, (name,address). You have operations which are addition, deletion and inspection of elements in this set. You also have queries whether a particular name or address is there in the set, and if so then count them and delete all such entries.How would you design your hash tables?
4. Suppose a binary tree has n nodes and has the property that either a node has 0 descendents or 2 descendents. What can be its maximum height?
5. The employees in a company are organized into teams. Each team has a hierarchical structure. Each employee may be a member of at most 3 teams. How would you represent such a structure? How will you answer simple questions such as “Does employee A have the same boss B in two different ways? Every team must meet together for one hour in a week. How many distinct hours are required for the whole company? Can you write a program to compute a simple approximation to this number?

Week 3 (10th August-16th August)

- ***Tree Walks / Traversals***
  Preorder and postorder traversals, evaluation of expressions, In-order traversal, Euler tour and expression with parentheses, building a tree from two traversals, limitations. Special case.

- ***Ordered Dictionaries***

The simple dictionary - min, max,pred, succ, search. List and array implementation. Insert and delete. Binary search trees. Implementing search, min and succ. The simple insert at the leaf level. Time complexity. Order of insertion and heights..

- ***Deletion[0:25]***
  The deletion - 3 cases, i.e., 0 leaves, 1 leaf, 2 leaves. Maintaining the inorder invariant. Inorder traversals. Sorting using a BST. Best and worst case.

  [0:25 onwards]: Argument on why, on the average, insertion of a random permutation takes O(nlogn) time.

Questions. Tutorial 3.

1. Given the elements [1,2,...,7] and the complete binary tree T with 7 nodes, label the nodes so that the preorder, inorder and postorder traversals produce the sequence 1,2,...,7 in that order.
2. Consider a binary tree with labels such that the postorder traversal of the tree lists the elements in increasing order. Let us call such a tree a post-order search tree. Describe how you will do search, min, max, insert and delete on this tree. Please write pseudo-code.
3. Construct the BST tree T whose post-order traversal is 1 3 5 4 2 7 8 6. For this tree delete the element 4 in two ways - by using its predecessor and its successor. Display these trees.
4. Given a BST T and an element a, the task is to delete all elements b<a from T. Write pseudocode to do this. How much time does your algorithm take?

Week 4 (17th August-23rd August)

- **Quick Sort**
  The algorithm and its average case analysis. The average permutation argument re-visited and the *O(nlogn)* time complexity.

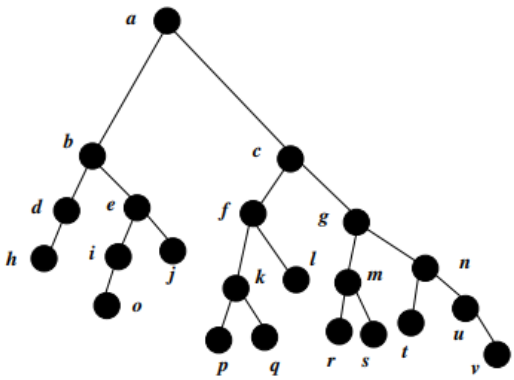
- **AVL Trees - Basics**
  The basic definition - height balance. The number of nodes in an AVL tree of height *h.* Two methods of estimating the number of nodes. Insertion and the loss of balance.

- **AVL Trees**

Insertion with the LL rotation. The LR case and double rotation. Deletion - reduction to the case of leaf deletion. Why does it percolate up? Running time. See https://www.cse.iitb.ac.in/cs213d/bst.pdf for a summary.

Questions. Tutorial 4.

1. In the ordinary BST, let us have an additional variable to be stored at the node, which is the height of the subtree rooted there. Thus every node x has the following attributes: left, right, value, height. Write pseudo-code for insert and delete in BST with the updation of height. Be careful with delete.
2. This is to ensure that you understand the analysis of the "random permutation". In Quicksort, we had said that if [a(1),a(2),...,a(n)] is a random permutation, then the probability that a(1)=i is precisely 1/n. Now, for a permutation as above, let b=a(a(1)). In other words, if a=[2,3,1,4,5] then b=3, but if a=[5,4,3,2,1] then b=1 and so on. If I were to select a random permutation, then what is the probability p(i) that b=i. Warning: p(i) is not the same for every i.
3. Consider the structure below. Is the tree structure AVL? If we wish to store the set 1, . . . , 22, label each node with the correct number.



4. Now add 23 to the set and then delete 1. Also do the same in the reverse order. Are the answers the same? When will the answers be the same?
5. Why is it that in insert, a single or double rotation is sufficient to balance the tree, but not in deletion?

Week 5 (26th August-31st August)

- **2-4 Trees**

A different way of keeping O(logn) find, insert, delete in search trees. Keeping flexible the number of children. Easier Insertion and trickier deletion.
https://www.youtube.com/watch?v=JZhdUb5F7oY

- **Disk Based Data Structures and B Trees**
  The practical aspects of storing in secondary memory. B Trees as generalization of 2-4 trees. https://www.youtube.com/watch?v=VbVroFR4mq4
  The amazing video. 5.28 B tree deletion in data structures by Ms. Jenny!

Questions. Tutorial 5.

1. Note that in both 2-4 and B-trees, the heights of all the children of any node are the same. With this requirement, would 2-6 trees or 5-7 trees make sense. What about general A-B trees?
2. Show two examples of 2-4 trees where an insertion increases the levels by 1 and a deletion drops the level by 1. Can these be the same trees?
3. Why is insertion in 2-4 and B-trees simpler than deletion?
4. In real-life B-trees, the number of children could be 100-1000. In this case, what is the fraction of values which are stored at the leaf level of the total number of values?
5. Design a structure where the values are stored only at the leaf level. Only end-markers are stored in the intermediate levels. Would insertion and deletion be easier in such "simple-B-trees"?

Week 6 (2nd September - 6th September)

- **Case Study: Searching for Patterns**
  The exact substring problem. The longest possible prefix which is a suffix at position i. The suffix short-cut and the KMP algorithm. Successful and unsuccessful comparisons. The running time O(m+n) assuming the h matrix.
  Lecture - 17 Case Study: Searching for Patterns

- **Tries**
  The Trie data structure. Motivation: Processing strings, computation of h matrix. Trie for a subset of words. Basic operations. Optimization. Size of the trie. Tries and the internet. Suffix trie and its construction. Lecture - 18 Tries

Questions. Tutorial 6.

1. This is concerning Prof. Garg's argument that the total number of unsuccessful comparisons = shifts in KMP algorithm is no more than |T|. Is this strictly true?
2. Please review the KMP algorithm to see how it detects two overlapping occurrences of the pattern.
3. For a pattern P[1...n], h(i) is defined to be the smallest k>0 such that P[1]=P[k+1],..., P[i-k]=P[i], but P[i-k+1]!=P[i+1]. If there is no such match, define h(i) to be i. Fill in the table below.

	1	2	3	4	5	6	7	8	9
P	b	a	b	b	a	a	b	b	a
h(i)									

4. Suppose that there is a letter z in P of length n such that it occurs in only one place, say k, which is given in advance. Can you optimize on the computation of h?
5. Compute the Suffix Trie for *abracadabra$*. Compress degree 1 nodes. Use substrings as edge labels. Put a square around nodes where a word ends. Use it to locate the occurrences of *abr*.
6. Review the argument that for a given text T, consisting of k words, the ordinary Trie occupies space which is a constant multiple of |T|. How is it that the suffix tree for a text T is of size O(|T|^2)?

Week 7-8 (20th September, 28th September)

- **Data Compression - Huffman Codes**
  Coding texts for compression. Weighted length as an optimization problem. The Huffman coding tree. Lecture - 19 Data Compression

- **Priority Queues**
  Examples of the need for priority queues. The basic delmin and insert operations. The array implementations, the binary heap. The heap condition and insert.
  Lecture - 20 Priority Queues

- **Binary Heaps**
  Delmin, heapify and its analysis. The heap sort and its analysis.
  Lecture - 21 Binary Heaps

- **Merge Sort**.
  The Divide and Conquer Approach. Lecture - 22 Why Sorting

Tutorial 7

1. In an Huffman code instance, show that if there is a character with frequency greater than ⅖ then there is a codeword of length 1. Show that if all frequencies are less than ⅓ then there is no codeword of length 1.
2. Suppose that there is a source which has three characters {a,b,c}. The output of the source cycles in the order of a,b,c followed by a again, and so on. In other words, if the last output was a b, then the next output will either be a b or a c. Each letter is equally probable. Is the Huffman code the best possible encoding? Are there any other possibilities? What would be the pros and cons of this?
3. Consider the following table of letters and frequency. Design a Huffman code tree.

a	20	**d**	7	**g**	8	**j**	4
b	6	**e**	25	**h**	8	**k**	2
c	6	**f**	1	**i**	12	**l**	1

4. Can a Priority Queue be implemented as an AVL tree? What advantages does a Heap implementation have over an AVL tree implementation?
5. The Heap implementation needs us to find the “last’ element in the heap. Write a code snippet to maintain the last element. Suppose we maintain a pointer to the last element, write a code snippet to go to the previous one. This will be useful if there are a sequence of deletes.What is the worst and the average time complexity of locating the previous? What happens if we do not maintain the last element? How do we locate the last element?
6. Suppose we have a 2D array where we maintain the following conditions: for every (i,j), we have A(i,j)<=A(i+1,j) and A(i,j)<=A(i,j+1). Can this be used to implement a priority queue?

1	1	2	3	Q

2	2	4	Q	Q
3	4	6	Q	Q
5	5	8	Q	Q
7	8	Q	Q	Q

7. Here is an interesting problem from Prof. Abhiram Ranade. A piecewise linear function on [0,1] may be represented by a sequence of special x and y values such as the table below:

x	0	0.3	0.5	1
f(x)	1.3	1.2	4.1	0.3

Thus, a good representation is f.NumberOfSegments, f.x (array of x values), f.y (array of y values). Now given two functions f and g, let h be the minimum of f and g. Clearly, it is also a piecewise linear function. Compute the representation of h.

Week 9 (30th Sep. - 5th October)

- **Graphs**
  Graphs - the basic definitions. Directed and undirected graphs. Paths and cycles. Subgraphs. Connectedness, connected components. Trees and forests. Number of edges and connectivity. Spanning trees. Nice (Eulerian) cycles. Degree of a node. The graph ADT. Various methods. Lecture - 24 Graphs

- **Data Structures for Graphs**
  The Edge-Vertex adjacency list. The adjacency matrix and the traditional adjacency list. Various extensions of the adjacency list. The Breadth First Search. Examples and the queue implementation. The O(E+V) time analysis. Predecessors and the BFS tree. Tree and non-tree edges. Lecture - 25 Data Structures for Graphs lec25.ppt.pdf

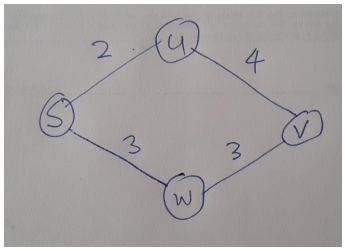
- **Two Applications of Breadth First Search**

Connected components and shortest paths. Bipartite graphs, odd cycles and detection using BFS. Diameter of a graph. A crude algorithm. The adjacency matrix.
Lecture - 26 Two Applications of Breadth First Search

Tutorial 8

1. The graph is an extremely useful modelling tool. Here is how a Covid tracing tool might work. Let V be the set of all persons. We say (p,q) is an edge (i) in E1 if their names appear on the same webpage, and (ii) in E2 if they have been together in a common location for more than 20 minutes. What significance does the connected components in these graphs and what does the BFS do? Does the second graph have epidemiological significance? If so, what? If not, how would you improve the graph structure to get a sharper epidemiological meaning?
2. Let us take a plane paper and draw circles and infinite lines to divide the plane into various pieces. There is an edge (p,q) between two pieces if they share a common boundary of intersection (which is more than a point). Is this graph bipartite? Under what conditions is it bipartite?
3. There are three containers A, B and C, with capacities 5,3 and 2 liters respectively. We begin with A having 5 liters of milk and B and C being empty. There are no other measuring instruments. A buyer wants 4 liters of milk. Can you dispense this? Model this as a graph problem with the vertex set V as the set of configurations c=(c1,c2,c3) and an edge from c to d if d is reachable from c. Begin with (5,0,0). Is this graph directed or undirected? Is it adequate to model the question: How to dispense 4 liters?
4. Suppose that there are M workers in a call center for a travel service which gives travel directions within a city. It provides services for N cities - C1,...,CN. Not all workers are familiar with all cities. The number of requests from each city per hour are R1,...,RN. A worker can handle K calls per hour. How would you model this problem? Assume that R1,...,RN and K are small numbers.
5. There is a set of bureaucrats B={b1,...bm}. Subgroups of them keep meeting and making decisions of n attributes, e.g., Parking is to be allowed (Y/N), Garba can take place (Y/N) etc, . Let us call these as boolean variable P1,P2,...,Pn. Each meeting M has a time-stamp and a decision to change these boolean values. The new assignment is carried forward with the bureaucrats who participated. Model this as a graph. What questions can be answered using this model? For example, given a meeting in which two bureaucrats bi and bj have opposite decisions on an attribute, can they decide who of the two has the latest information?

6. Study the BFS code from Prof. Naveen's slides. Argue that at any time during the running of the algorithm the d-values of vertices in the queue are non-decreasing and can only take 1 or 2 consecutive values.
7. List the properties of the white, grey and black vertices. In line 10, while processing u, can we encounter a vertex v which is gray? Or black? In such cases what are the values possible for d[v]? At any time, what are the properties of the sets of white, black and grey vertices?
8. There are many variations of BFS to solve various needs. For example, suppose that every edge e=(u,v) also has a weight w(e) (say the width of the road from u to v). For a path p= (v1,v2,...,vk), let the weight w(p) be the minimum of the weights of the edges in the path. We would like to find a shortest path from a vertex s to all vertices v. If there are multiple such paths, we would like to find a path whose weight is maximum. For example, in the graph below, we would prefer path s->w->v. Can we adapt BFS to detect this path?




```
BFS(G,s)
01 for each vertex u ∈ V[G]-{s}
02     color[u] ← white
03     d[u] ← ∞
04     π[u] ← NIL
05 color[s] ← gray
06 d[s] ← 0
07 π[u] ← NIL
08 Q ← {s}
09 while Q ≠ ∅ do
```

```
10    u ← head[Q]
11    for each v ∈ Adj[u] do
12       if color[v] = white then
13          color[v] ← gray
14          d[v] ← d[u] + 1
15          π[v] ← u
16          Enqueue(Q,v)
17    Dequeue(Q)
18    color[u] ← black
```

Week 10

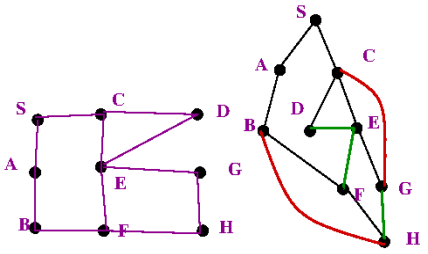
- Depth First Search.
  The recursive program. The global and the local variables. DFS numbering. DFS tree. Tree and non-tree edges. Back edges and order of visiting. Arrival and departure times.

  global counter
  function DFS(*G*, *v*) **is**
    label *v* as discovered
    ar[u]=counter; counter=counter+1;
    for all directed edges from *v* to *w that are* **in** *G*.adjacentEdges(*v*) **do**
      if vertex *w* is not labeled as discovered **then**
        //inlabel[(v,w)]=counter;
        recursively call DFS(*G*, *w*) // outlabel[(v,w)]=counter
      Endif
    Endfor
    dep[u]=counter; counter=counter+1;
  Endfunction

- Applications of DFS.
  Adjacency list implementation and a non-recursive implementation. Timing analysis. 2-edge-connectedness. Use of dfs to check this. Checking if a tree-edge is essential. Deepest back edge. Planar graphs. 2-vertex connected.

Tutorial 9

1. Please study the undirected graph below and the tree edges (in black), non-tree edges in green (and some forbidden edges in red). We have seen that this gives a tree, in fact, a shortest-path tree for the vertex s. In other words, it is a tree T for which the path from s to another vertex u is the shortest path in the original graph. Thus bfs produces a shortest path tree. Note that there may be many shortest path trees and the particular tree produced by bfs is one of them, which depends on the vertex ordering, i.e., how the vertices are ordered among themselves. Now, given a shortest path tree T, is there always an ordering of the vertices which will produce this tree T as outputs?



2. For the same graph below, execute dfs and produce a trace of the algorithm. List the vertices in order of push and pop events and mark the arrival and departure numbers. Modify the DFS to label tree edges twice with the counter, once while going forward, i.e., push (i.e., the recursive call) and the second time when popping, i.e., when exiting.

3. Suppose that there is an undirected graph G(V,E) where the edges are colored either red or blue. Given two vertices u and v. It is desired to (i) find the shortest path irrespective of colour, (ii) find the shortest path, and of these paths, the one with the fewest red edges, (iii) a path with the fewest red edges. Draw an example where the above three paths are distinct. Clearly, to solve (i), BFS is the answer. How will you design algorithms for (ii) and (iii)?

4. Suppose that we have a graph and we have run dfs starting at a vertex s and obtained a dfs tree. Next, suppose that we add an edge (u,v). Under what conditions will the dfs tree change? Give examples.

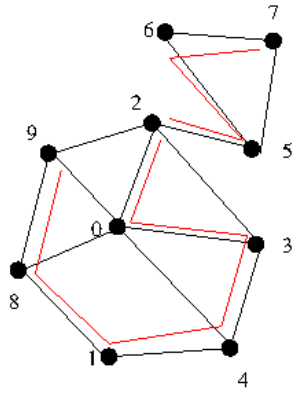
5. We have a new search algorithm which uses a set S for which we have two functions (i) add(x,S) which adds x to S, and (ii) y=select(S) which returns an element of S following a certain rule.

```
Function mysearch
Global visited;
For all u visited[u]=false;
for all edges e, found[e]=false;
S=empty;
add(s,S); nos=1; record[nos]=s;
While nonempty(S)
   y=select(S)
   nos=nos+1; record[nos]=y;
   For all v adjacent to y
     If visited[v]==false
        visited[v]=true;
        found[(u,v)]=true;
        add(v,S);
     Endif;
   Endfor;
Endwhile
Endfunction;
```

(i) Compare the bfs and dfs algorithms with the above code. Take special care in understanding visited.
(ii) Let us look at the sequence record[1],record[2],...,record[n]. Show that there is a path from record[i] to record[i+1] using only edges which have been found at that point.
(iii) Compare BFS and DFS in terms of the above path lengths.

6. This is the theoretical basis of edge-2-connectedness. Let G(V,E) be a graph. We define a relation on edges as follows: two edges e and f are related (denoted by e~f iff there is a cycle containing both. Show that this is an equivalence relation. The equivalence class [e] of an edge e is called its 2-connected component.

7. Given a dfs tree for an undirected graph and a vertex v, we define C(v) as the edge (x,y) where x is a descendent of y while x is a parent of v, with x!=v and y!=v. Modify dfs to list C(v) and to compute |C(v)| . In the example below C(6)=1, C(5)=0 and C(1) =3. How much time does your algorithm take? Modify the code counterdfs.c

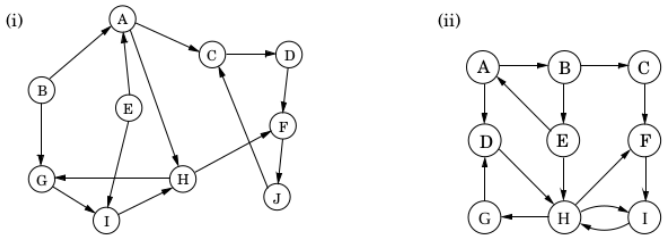


dfs=[2 0 3 4 1 8 9 5 6 7]

Week 11.

- **DFS in Directed Graphs**
  The non-tree edges and the 4 types. Cycles and back-edges. An acyclic graph and its topological sort. Strongly connected graphs and components. The O(mn) time algorithm to check strong-connectedness. Lecture - 29 DFS in Directed Graphs

- **Applications of DFS in Directed Graphs**
  Checking strong connectedness by graph reversal. The connected component algorithm. Ordering by arrival times.
  Lecture - 30 Applications of DFS in Directed Graphs



Week 12.

- Single Source Shortest Paths
  The s-t shortest path problem. Enumerating paths and its complexity. Property of shortest paths. The incremental algorithm. The specification of the Dijkstra's algorithm.
  Lecture - 34 Single Source Shortest Paths

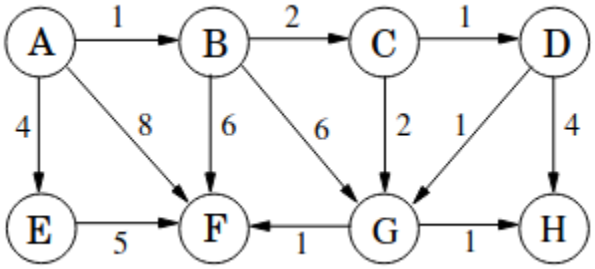
Figure 4.8 Dijkstra's shortest-path algorithm.

```
procedure dijkstra(G, l, s)
Input:    Graph G = (V, E), directed or undirected;
          positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue(V)   (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
      if dist(v) > dist(u) + l(u, v):
         dist(v) = dist(u) + l(u, v)
         prev(v) = u
         decreasekey(H, v)
```

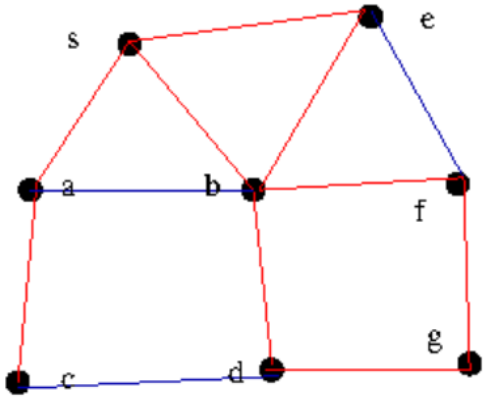
- Correctness of Dijkstra's Algorithm
  The inductive argument. Running time using priority queues. The problem with negative edges. A counter example.
  Lecture - 35 Correctness of Dijkstras Algorithm

Tutorial 11.

1. Modify Dijkstra's algorithm to compute the number of shortest paths from s to every vertex t.
2. Compute the shortest path for all vertices starting from A. Do this in tabular form.



3. Show an example of a graph with negative edge weights and show how Dijkstra's algorithm may fail. Suppose that the minimum negative edge weight is -d. Suppose that we create a new graph G' with weights w', where G' has the same edges and vertices as G, but w'(e)=w(e)+d. In other words, we have added d to every edge weight so that all edges in the new graph have edge weights non-negative. Let us run Dijkstra on this graph. Will it return the shortest paths for G?
4. Look at the following graph from Tutorial 9 with red edges and blue edges. Our task was to find the path from s to every vertex t, with the fewest red edges. Run any modified bfs of your choice and Dijkstra and compare the sequence of vertices visited by BFS and by Dijkstra.



5. You are given a time table for a city. The city consists of n stops V={v1,v2,...,vn}. It runs m services s1,s2,...,sm. Each service is a sequence of vertices and timings. For example, the schedule for service K7 is given below. Now, you are at stop A at 8:00am and you would like to reach stop B at the earliest possible time. Assume that buses may be delayed by at most 45 seconds. Model the above problem as a shortest path problem. The answer should be a travel plan.

Service : K7			
H15	Convocation Hall	Market Gate	H15
7:15am	7:20am	7:30	7:40