CS502 Fundamentals of Algorithms
Final TERM Subjective

Differentiate between back edge and forward edge Suppose you could prove that an NP-complete problem can not be solved in polynomial time. What would be the consequence?
Answer:

Back Edge
A **back edge** connects a vertex to an ancestor in a DFS-tree. Note that a self-loop is a back edge.
from descendent to ancestor
(u, v) where v is an ancestor of u in the tree.
DFS tree may only have a single back edge
A back edge is an arc whose head dominates its tail (tail -> head)
a back edge must be a part of at least one loop

Forward Edge

from ancestor to descendent
(u, v) where v is a proper descendent of u in the tree.
A **forward edge** is a non-tree edge that connects a vertex to a descendent in a DFS-tree.
Edge x-y is less than the capacity there is a forward edge x-y with a capacity equal to the capacity and the flow

According to the question this means that the problem can be solved in Polynomial time using known NP problem can be solved using the given problem with modified input (an NP problem can be reduced to the given problem) then the problem is NP complete.
The main thing to take away from an NP-complete problem is that it cannot be solved in polynomial time in any known way. NP-Hard/NP-Complete is a way of showing that certain classes of problems are not solvable in realistic time.
Ref: Handouts Page No.128

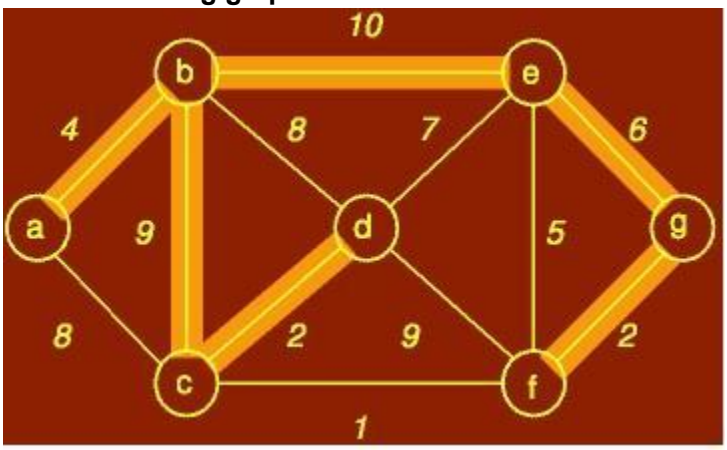
Recursive explanation of dynamic programming.
Formulate the problem into smaller sub problems, find optimal solution to these sub problems in a bottom up fashion then write an algorithm to find the solution of whole problem starting with base case and works it way up to final solution.

Answer:
Dynamic programming is essentially recursion without repetition. Developing a dynamic programming algorithm generally involves two separate steps:

Formulate problem recursively. Write down a formula for the whole problem as a simple combination of answers to smaller sub problems.

Build solution to recurrence from bottom up. Write an algorithm that starts with base cases and works its way up to the final solution.
Ref: Handouts Page No.75

What is the cost of the following graph?



Cost =33

Answer:
A common problem is communications networks and circuit design is that of connecting together a set of nodes by a network of total minimum length. The length is the sum of lengths of connecting wires.

Consider, for example, laying cable in a city for cable t.v.

The computational problem is called the *minimum spanning tree* (MST) problem. Formally, we are given
a connected, undirected graph G = (V, E) Each edge (u, v) has numeric weight of cost. We define the cost of a spanning tree T to be the sum of the costs of edges in the spanning tree

$$w(T) = \sum_{(u,v)2T} w(u, v)$$

A minimum spanning tree is a tree of minimum weight The first is a spanning tree but is not a MST;
Ref: Handouts Page No.142

Let the adjacency list representation of an undirected graph is given below. Explain what general property of the list indicates that the graph has an isolated vertex.

a □ b □ c □ e

b □ a □ d

c □ a □ d □ e □ f

d □ b □ c □ f

e □ a □ c □ f

f □ c □ d □ e

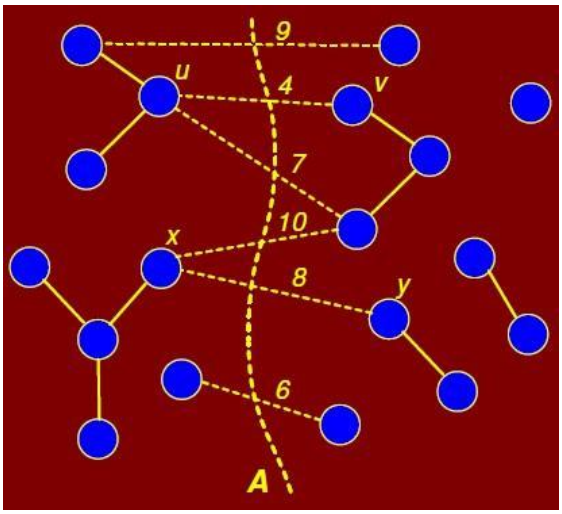
g

Answer:-

The main theorem which drives both algorithms is the following:

MST Lemma: Let G = (V, E) be a connected, undirected graph with real-valued weights on the edges.

Let A be a viable subset of E (i.e., a subset of some MST). Let (S, V − S) be any cut that respects A and let (u, v) be a light edge crossing the cut. Then the edge (u, v) is safe for A.



MST Proof: It would simplify the proof if we assume that all edge weights are distinct. Let T be any MST for G. If T contains (u, v) then we are done. This is shown in Figure 8.47 where the lightest edge (u, v) with cost 4 has been chosen.

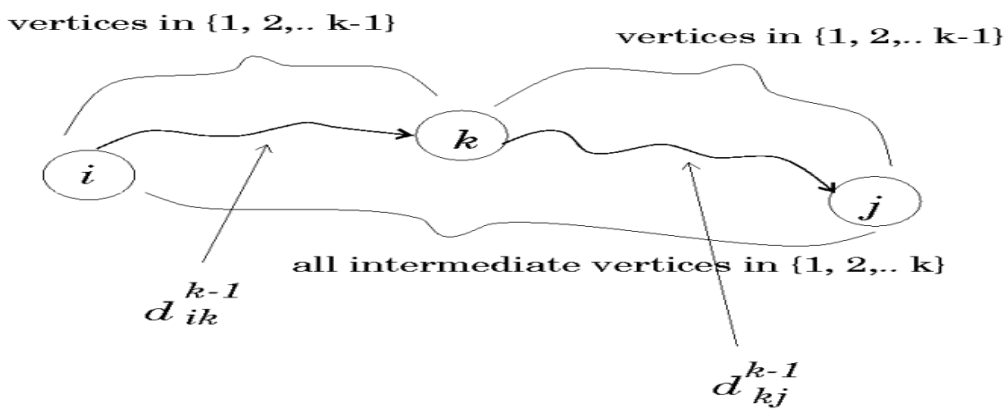
Ref: Handouts Page No.144

Floyd Warshal algorithm with three recursive steps

1. **Wij = 0 if I = j**
2. **Wij = w(i,j) if i != j and (i,j) belongs to E**
3. **Wij = infinity if i != j and (i,j) not belongs to E**

Answer

The Floyd-Warshall algorithm: Step (i)



The Floyd-Warshall algorithm: Step (ii)
Recursively define the value of an optimal solution.
Boundary conditions: for k = 0, a path from vertex i to j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all, **hence d (0)= wij**

Recursive formulation:

$$d\frac{(k)}{ij} = \begin{cases} wij \\ \min(d\frac{(k-1)}{ij}, d\frac{(k-1)}{ik} + d\frac{(k-1)}{kj}) ifk \geq 1 \end{cases}$$

$$\frac{D^{(n)} = (d^{(n)}ij\ )}{d^{(n)}ij\ \ = \sigma(i,j).}$$ **is the solution for this APSP problem:**

The Floyd-Warshall algorithm: Step (iii)
Compute the shortest-path weights bottom up
FLOYD-WARSHALL(W, n)

$\{$

$\mathbf{D}^{(0)} = W;$

for k := 1 to n

for i := 1 to n

for j := 1 to n

$\mathbf{D}_{ij}^{(k)} := \min(d_{ij}^{(k-1)}, d_{ik}^{(k+1)}, +d_{kj}^{(k-1)},)$

$return\mathbf{D}^{(n)};$

$\}$

$complexity\Theta(n^3).$

Ref: Handouts Page No.161

Give a detailed example for 2d maxima problem

Answer:-

The problem with the brute-force algorithm is that it uses no intelligence in *pruning* out decisions.

For example, once we know that a point pi is dominated by another point pj, we do not need to use pi for eliminating other points. This follows from the fact that dominance relation is *transitive*. If pj dominates pi and pi dominates ph then pj also dominates ph; pi is not needed.

Ref: Handouts Page No.17

How the generic greedy algorithm operates in minimum spanning tree?

Answer:-

A generic greedy algorithm operates by repeatedly adding any safe edge to the current spanning tree. When is an edge safe? Consider the theoretical issues behind determining whether an edge is safe or not.

Let S be a subset of vertices S _ V. A cut (S, V − S) is just a partition of vertices into two disjoint subsets. An edge (u, v) crosses the cut if one endpoint is in S and the other is in V − S. Given a subset of edges A, a cut respects A if no edge in A crosses the cut. It is not hard to see why respecting cuts are important to this problem. If we have computed a partial MST and we wish to know which edges can be added that do not induce a cycle in the current MST, any edge that crosses a respecting cut is a possible candidate.

Ref: Handouts Page No.143

What are two cases for computing

Answer:-

There are two cases for computing Lij the match case if ai = bj , and the mismatch case if ai 6= bj . In the match case, we do the following:

$$L_j^{\,i} = Inserta_j \,\mathrm{int}\, oL_j^{\,i-1},$$

and in the mismatch case, we do the following:

$$L_j^{\,i} = merge(L_{j-1}^{\,i}, L_j^{\,i-1}).$$

Ref: Handouts Page No.143

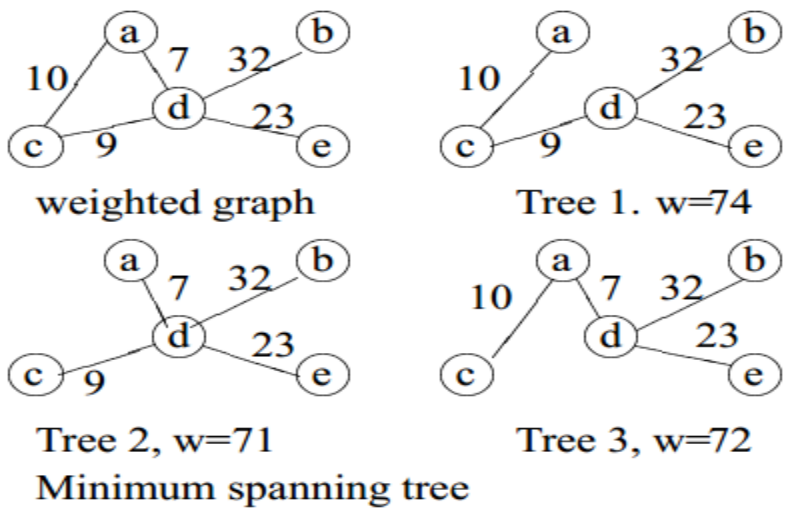
Describe Minimum Spanning Trees Problem with examples.

Problem

Given a connected weighted undirected graph, design an algorithm that outputs a minimum spanning tree (MST) of

Examples

The graph is a complete, undirected graph G = (V, E ,W), where V is the set of pins, E is the set of all possible interconnections between the pairs of pins and w(e) is the length of the wire needed to connect the pair of vertices



Ref: Handouts Page No.142

What is decision problem, also explain with example?

A decision problem is a question in some formal system A problem is called a decision problem if its output is a simple “yes” or “no” (or you may this of this as true/false, 0/1, accept/reject.) We will phrase may optimization problems as decision problems.

For example,

the MST decision problem would be: Given a weighted graph G and an integer k, does G have a spanning tree whose weight is at most k?

Ref: Handouts Page No.170

Do you think greedy algorithm gives an optimal solution to the activity scheduling problem?

The greedy algorithm gives an optimal solution to the activity scheduling problem.
Proof:
The proof is by induction on the number of activities. For the basis case, if there are no activities, then the greedy algorithm is trivially optimal. For the induction step, let us assume that the greedy algorithm is optimal on any set of activities of size strictly smaller than |S| and we prove the result for S. Let S0 be the set of activities that do not interfere with activity a1, That is

$$S' = \{ai \in S \mid si \geq f1\}$$

Any solution for S0 can be made into a solution for S by simply adding activity a1, and vice versa. Activity a1 is in the optimal schedule (by the above previous claim). It follows that to produce an optimal schedule for the overall problem, we should first schedule a1 and then append the optimal schedule for S0. But by induction (since |S0| < |S|), this exactly what the greedy algorithm does.
Ref: Handouts Page No.109

Define Forward edge

The most natural result of a depth first search of a graph (if it is considered as a function rather than procedure) is a spanning tree of the vertices reached during the search. Based on this spanning tree, the edges of the original graph can be divided into three classes: forward edges (or "discovery edges"), which point from a node of the tree to one of its descendants,
Ref: Handouts Page No.129 130

Is there any relationship between number of back edges and number of cycles in DFS?

Answer:
The time stamps given by DFS allow us to determine a number of things about a graph or digraph. we can determine whether the graph contains any cycles.

Lemma: Given a digraph G = (V, E), consider any DFS forest of G and consider any edge (u, v) 2 E. If this edge is a tree, forward or cross edge, then f[u] > f[v]. If this edge is a back edge, then f[u] _ f[v].

Proof: For the non-tree forward and back edges the proof follows directly from the parenthesis lemma. For example, for a forward edge (u, v), v is a descendent of u and so

v's start-finish interval is contained within u's implying that v has an earlier finish time. For a cross edge (u, v) we know that the two time intervals are disjoint. When we were processing u, v was not white (otherwise (u, v) would be a tree edge), implying that v was started before u. Because the intervals are disjoint, v must have also finished before u.

Ref: Handouts Page No.130

What is the common problem in communications networks and circuit designing?

Answer:

A common problem is communications networks and circuit design is that of connecting together a set of nodes by a network of total minimum length. The length is the sum of lengths of connecting wires.

Consider, for example, laying cable in a city for cable t.v.

Ref: Handouts Page No.142

Explain the following two basic cases according to Floyd-Warshall Algorithm,

1. Don t go through vertex k at all.
2. Do go through vertex k.

Answer:-

Don't go through k at all

Then the shortest path from i to j uses only intermediate vertices {1, 2, . . . , k − 1}. Hence the length of

the shortest is $d_{ij}(k-1)$

Do go through k

First observe that a shortest path does not go through the same vertex twice, so we can assume that we

pass through k exactly once. That is, we go from i to k and then from k to j. In order for the overall path to be as short as possible, we should take the shortest path from i to k and the shortest path from k to j.

Since each of these paths uses intermediate vertices {1, 2, . . . , k − 1}, the length of the path is

$$d\frac{(k-1)}{ik} + d\frac{(k-1)}{kj}$$

Ref: Handouts Page No.162

Show the result of time stamped DFS algorithm on the following graph. Take node A as a starting node.

Answer:-

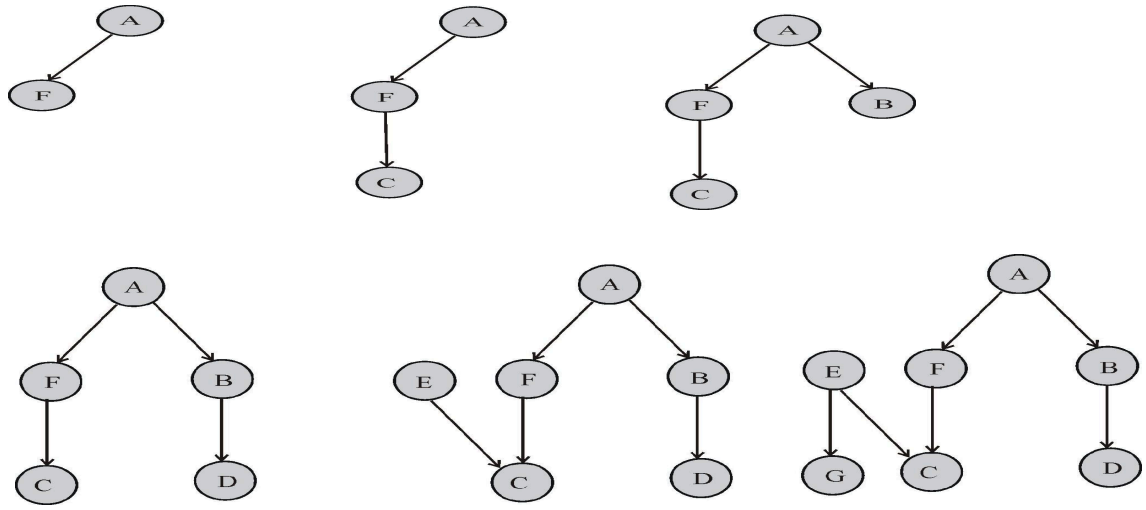
Depth-first search (DFS) :

It is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, one starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it hadn't finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a LIFO stack for exploration.

```
 DFS (graph)
{
  list L =empty
  tree T =empty
 choose a starting vertex x
 search(x)
  while(Lis not empty)
 {
    remove edge (v, w) from beginning of L
      if w not yetvisited
      {
        add (v, w) to T
        search(w)
      }
  }
}

search(vertex v)
{
   visit v
   for each edge (v, w)
   add edge (v, w) to the beginning of L
}
```



Ref:
http://www.chegg.com/homework-help/questions-and-answers/result-time-stamped-dfs-algorithm-followingqraph-node-starting-node-stronglyconnected-comp-q96642

Why we need reduction?
Answer:-
The class NP-complete (NPC) problems consists of a set of decision problems (a subset of class NP) that no one knows how to solve efficiently. But if there were a polynomial solution for even a single NP-complete problem, then ever problem in NPC will be solvable in polynomial time. For this, we need the concept of reductions.
Ref: Handouts Page No.173

Consider the digraph on eight nodes, labeled 1 through 8, with eleven directed edges 1 2, 1 4, 2 4, 3 2, 4 5, 5 3 ,5 6, 7 8, 7 1, 2 7,8 7
Answer:-

0-7 0-1 1-4 1-6 2-3 3-4 4-2 5-2 6-0 6-3 6-5 7-1 7-3

Draw the DFS tree for the standard adjacency-matrix representation. List the edges of each type in the space provided below.

0-7	tree
0-1	tree
1-4	tree
1-6	tree
2-3	tree
3-4	back
4-2	tree
5-2	cross
6-0	back
6-3	cross
6-5	tree
7-1	cross
7-3	cross

```
    0
   / \
  1   7
 / \
4   6
 |   |
2   5
 |
 3
```

Ref
http://www.cs.princeton.edu/courses/archive/fall04/cos226/exams/fin-s01-sol.html

Prove that the generic TRAVERSE (S) marks every vertex in any connected graph exactly once and the set of edges (v, parent (v)) with parent (v) [1]F form a spanning tree of the graph

Answer:-

Generic Traverse

- Suppose we want to visit every node in a connected graph (represented either explicitly or implicitly)
- The simplest way to do this is an algorithm called depth-first search
- We can write this algorithm recursively or iteratively - it's the same both ways, the iterative version just makes the stack explicit
- Both versions of the algorithm are initially passed a source vertex v

Lemma
Traverse(s) marks each vertex in a connected graph exactly once, and the set of edges (v, parent(v)), with parent(v) not nil, form a spanning tree of the graph.

Proof

- Call an edge (v, parent(v)) with parent(v) =6 nil a parent edge.
- Note that since every node is marked, every node has a parent edge except for s.
- It now remains to be shown that the parent edges form a spanning tree of the graph

Ref http://www.cs.unm.edu/~saia/362-s08/lec/lec18-2x2.pdf

What is a run time analysis and its two criteria
Answer:-
The main purpose of our mathematical analysis will be measuring the execution time. The running time of an implementation of the algorithm would depend upon the speed of the computer, programming language, optimization by the compiler etc. Two criteria for measuring running time are worst-case time and average-case time.
Worst-case time is the maximum running time over all (legal) inputs of size n. Let I denote an input instance, let |I| denote its length, and let T(I) denote the running time of the algorithm on input I. Then

$$T_{worst}(n) = \max_{|I|=n} T(I)$$

Average-case time is the average running time over all inputs of size n. Let p(I) denote the probability of seeing this input. The average-case time is the weighted sum of running times with weights being the probabilities:

$$T_{avg}(n) = \sum_{|I|=n} p(I)T(I)$$

We will almost always work with worst-case time. Average-case time is more difficult to compute; it is difficult to specify probability distribution on inputs. Worst-case time will specify an upper limit on the running time.
Ref: Handouts Page No.173

Dijkstra algorithm correctness criteria two conditions
Answer:
We will prove the correctness of Dijkstr's algorithm by Induction. We will use the definition that _(s, v)
denotes the minimal distance from s to v.
For the base case
1. S = {s}
2. d(s) = 0, which is _(s, s
Ref: Handouts Page No.158

Compare bellman ford algorithm with dijikstr's algorithm. Also give the time complexity of bellman ford algorithm (3 marks)
Answer:
Dijkstra's single-source shortest path algorithm works if all edges weights are non-negative and there are no negative cost cycles. Bellman-Ford allows negative weights edges and no negative cost cycles. The algorithm is slower than Dijkstra's, running in _(VE) time. Like Dijkstra's algorithm, Bellman-Ford is based on performing repeated relaxations. Bellman-Ford applies relaxation to every edge of the graph and repeats this V − 1 times.
Ref: Handouts Page No.159

psedo code of timestamp DFS
Answer:
DFS(G)
1 for (each u 2 V)
2 do color[u] white
3 pred[u] nil
4 time 0
5 for each u 2 V
6 do if (color[u] = white)
7 then DFSVISIT(u)
Ref: Handouts Page No.126

variants of shortest path solution
Answer
There are a few variants of the shortest path problem.
Single-source shortest-path problem: Find shortest paths from a given (single) source vertex s 2 V to every other vertex v 2 V in the graph G.

Single-destination shortest-paths problem: Find a shortest path to a given destination vertex t from each vertex v. We can reduce the this problem to a single-source problem by reversing the direction of each edge in the graph.

Single-pair shortest-path problem: Find a shortest path from u to v for given vertices u and v. If we solve the single-source problem with source vertex u, we solve this problem also. No algorithms for this problem are known to run asymptotically faster than the best single-source algorithms in the worst case.

All-pairs shortest-paths problem: Find a shortest path from u to v for every pair of vertices u and v. Although this problem can be solved by running a single-source algorithm once from each vertex, it can usually be solved faster.
Ref: Handouts Page No.153

Prove the following lemma,
Lemma: Given a digraph G = (V, E), consider any DFS forest of G and consider any edge (u, v) ∈ E. If this edge is a tree, forward or cross edge, then f[u] > f[v]. If this edge is a back edge, then f[u] ≤ f[v]

Answer:-
Proof: For the non-tree forward and back edges the proof follows directly from the parenthesis lemma. For example, for a forward edge (u, v), v is a descendent of u and so v's start-finish interval is contained within u's implying that v has an earlier finish time. For a cross edge (u, v) we know that the two time intervals are disjoint. When we were processing u, v was not white (otherwise (u, v) would be a tree edge), implying that v was started before u. Because the intervals are disjoint, v must have also finished before u.

RAM(Random Access memory)and its Applications?
Answer:
A RAM is an idealized machine with an infinitely large random-access memory. Instructions are executed one-by-one (there is no parallelism). Each instruction involves performing some basic operation on two values in the machines memory. Basic operations include things like assigning a value to a variable, computing any basic arithmetic operation (+, - , × , integer division) on integer values of any size, performing any comparison (e.g. x _
Ref: Handouts Page No.10

Describe Dijkstra's algorithm working?
Answer
Dijkstra's algorithm works on a weighted directed graph G = (V, E) in which all edge weights are non-negative, i.e., w(u, v) _ 0 for each edge (u, v) 2 E.

Ref: Handouts Page No.154

Prim algorithm graph?
Answer
Prim's algorithm builds the MST by adding leaves one at a time to the current tree. We start with a root vertex r; it can be any vertex. At any time, the subset of edges A forms a single tree. We look to add a single vertex as a leaf to the tree.
Ref: Handouts Page No.149

write suedo code of relaxing a vertex 5
Answer
RELAX((u, v))
1 if (d[u] + w(u, v) < d[v])
2 then d[v] d[u] + w(u, v)
3 pred[v] = u
Ref: Handouts Page No.155
Define NP completeness
Answer:
The set of NP-complete problems is all problems in the complexity class NP for which it is known that if anyone is solvable in polynomial time, then they all are. Conversely, if anyone is not solvable in polynomial time, then none are.
Definition: A decision problem L is NP-Hard if
L0 _P L for all L0 2 NP.
Definition: L is NP-complete if
1. L 2 NP and
2. L0 _P L for some known NP-complete problem L0.
Ref: Handouts Page No.176

Define DAG
A directed graph that is acyclic is called a directed acyclic graph (DAG).
Ref: Handouts Page No.116

How Kruskal's algorithm works ?
Answer:
Kruskal's algorithm works by adding edges in increasing order of weight (lightest edge first). If the next edge does not induce a cycle among the current set of edges, then it is added to A. If it does, we skip it and consider the next in order. As the algorithm runs, the edges in A induce a forest on the vertices. The trees of this forest are eventually merged until a single tree forms containing all vertices.
Ref: Handouts Page No.147

What are two steps generally involved while developing a dynamic programming algorithm?

Answer:
Dynamic programming is essentially recursion without repetition. Developing a dynamic programming algorithm generally involves two separate steps:
Formulate problem recursively. Write down a formula for the whole problem as a simple combination of answers to smaller sub problems.
Build solution to recurrence from bottom up. Write an algorithm that starts with base cases and works its way up to the final solution.
Ref: Handouts Page No.75

The following adjacency matrix represents a graph that consists of four vertices labeled 0, 1, 2 and 3. The entries in the matrix indicate edge weights.

	0	1	2	3
0	0	1	0	3
1	2	0	4	0
2	0	1	0	1
3	2	0	0	0

Answer
NO. Because number of rows and number of columns are always same.

What is the application of edit distance technique?
Answer
Edit Distance: Applications
There are numerous applications of the Edit Distance algorithm. Here are some examples:

Spelling Correction
If a text contains a word that is not in the dictionary, a 'close' word, i.e. one with a small edit distance, may be suggested as a correction. Most word processing applications, such as Microsoft Word, have spelling checking and correction facility. When Word, for example, finds an incorrectly spelled word, it makes suggestions of possible replacements.
Plagiarism Detection
If someone copies, say, a C program and makes a few changes here and there, for example, change variable names, add a comment of two, the edit distance between the source and copy may be small. The edit distance provides an indication of similarity that might be too close in some situations.
Computational Molecular Biology

Speech Recognition

Algorithms similar to those for the edit-distance problem are used in some speech recognition systems. Find a close match between a new utterance and one in a library of classified utterances.

Ref: Handouts Page No.76