Page:
Date:

# Assignment No. 2

1a) Differentiate betn Greedy approach and Dynamic programming.

→

| Greedy Approach | Dynamic programming |
|---|---|
| (1) Generate a single decision sequence | many decision sequence may be generated. |
| (2) Less reliable | highly reliable |
| (3) Top-down approach | Bottom-up approach. |
| (4) Efficiency is more | Efficiency is less |
| (5) Overlapping subproblem cannot be handled | Choose optimal solution to subproblem |
| (6) Contain a particular set of feasible set of soln | There is no special set of feasible set of solution |
| (7) No memorization is required | Memorization is required |
| (8) Faster than dynamic | Slower than Greed approach |
| (9) Fast result | Slow result. |
| (10) Each step is locally optimal | past solution are used to create new one. |
| (11) Fractional knapsack is the example of greedy Approach | 0/1 knapsack is example of dynamic programming. |

16) Determine LCS of X = POLYNOMIAL and Y = EXPONENTIAL.

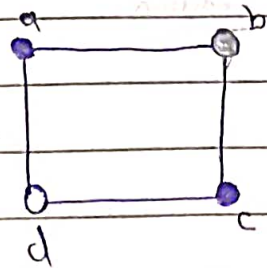
| | Xi | P | O | L | Y | N | O | M | I | A | L |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Yi | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 |
| X | 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 | ↑ 0 |
| P | 0 | ↖ (1) | ← 1 | ← 1 | ← 1 | ← 1 | ← 1 | ← 1 | ← 1 | ← 1 | ← 1 |
| O | 0 | ↑ 1 | ↖ (2) | ← 2 | ← 2 | ← 2 | ↖ 2 | ← 2 | ← 2 | ← 2 | ← 2 |
| N | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 2 | ↖ (3) | ← 3 | ← 3 | ← 3 | ← 3 | ← 3 |
| E | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 |
| N | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 2 | ↖ 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 |
| T | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 3 |
| I | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 | ↑ 3 | ↖ (4) | ← 4 | ← 4 |
| A | 0 | ↑ 1 | ↑ 2 | ↑ 2 | ↑ 2 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 4 | ↖ (5) | ← 5 |
| L | 0 | ↑ 1 | ↑ 2 | ↖ 3 | ← 3 | ↑ 3 | ↑ 3 | ↑ 3 | ↑ 4 | ↑ 5 | ↖ (6) |

$\therefore LCS(X,Y) = (P, O, N, I, A, L)$

3a) Explain Graph coloring method with example. Give algorithm for it.

→ Graph coloring is procedure of assignment of colors to each vertex of a graph G such that no adjacent vertices get same color. The objective is to minimize the no. of colors while coloring a graph. The smallest no. of colors required to color a graph G is called its chromatic number of that graph. Graph coloring problem is NP complete problem.

eg.



The figure, at first vertex a is colored blue. As the adjacent vertices of vertex a are again adjacent, vertex b and vertex d are colored with different color, black & white respectively. Then vertex c is colored as blue no adjacent vertex of c is colored blue. Hence the chromatic numbers of graph is 3.

Algorithm :—→

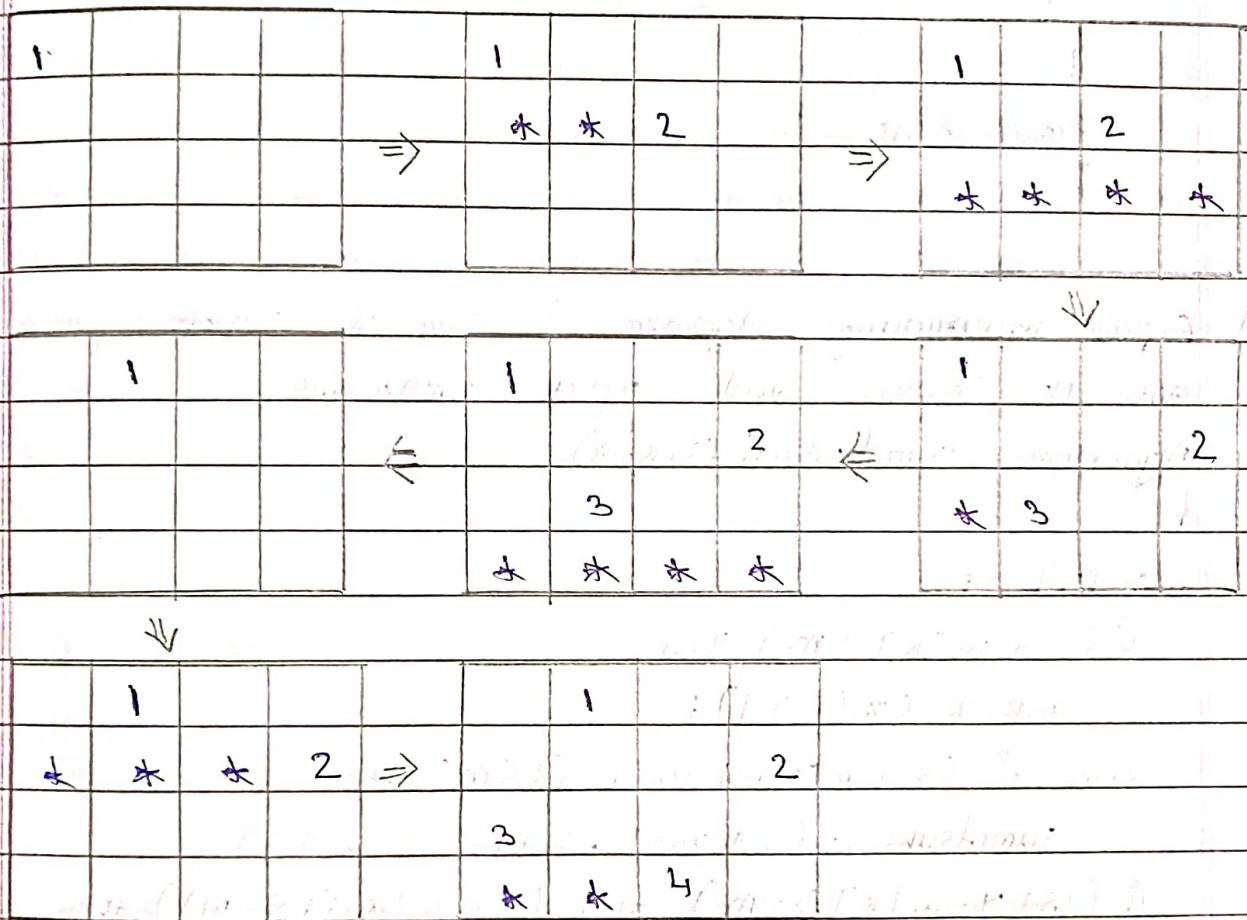
```
Algorithm mcoloring(k)
{
 repeat
   { next value (k)
     if (x[k]==0) then return
     if (k==n) then write (x:[1:n])
     else
       mcoloring(k+1)
   } until (false)
}

Algorithm next value (k)
{
 repeat
   {
     x[k] = (x[k]+1) mod(m+1)
     if (x[k]==0) then return
     for j=1 to n do
       { if ((G[k,j]≠0] and (x[k] == x[j]))
           then break
       }
     if (j == (n+1)) then return
   } until (false)
}
```

Page:
Date: / /

3b) Discuss 4- Queen's problem and give its algo. using Backtracking approach.

→ In 4 queen problem, we have 4 queens to be placed on 4 * 4 chessboard, satisfying the constraint that no two queens should be in the same row, same column or in same diagonal.

The solution space accordingly to external constraints consists of 4 to power 4, 4-tuples i.e. $S_i = \{1,2,3,4\}$ and $1 <= i <= 4$, whereas accordingly to internal constraints they consist of 4! solution i.e. permutation of 4.




Result : {2,4,1,3}

Algorithm:→

```
private static boolean issafeplace(int column, int qi, int [] board)
  for (int i=0; i<qi ; i++)
  {
    if (board [i] == column)
      { return false;
      }
    if (math.abs (board [i] - column) == math.abs (i - qi))
    {
      return false;
    }
  }
  return true;
  }
```

4c) Explain backtracking algorithm for sum of subsets problem state its implicit and explict constraints.

→ Algorithm SumofSubsets (s, k, r)

```
{
  x [k] = 1;
  if (s + w [k] = m) then
      WRITE (x [1:k]);
  else if (s + w [k] + w [k+1] <m) then
      sumofsubsets (s + w[k] , k+1 , r-w [k]);
  if ((s+ r - w [k] >= m) and (s + w [k+1] <= m)) then
   {
      x [k]=0;
      sumofsubsets (s, k+1, r-w[k]);
   }
}
```

Page:
Date:

Implicit constraint: →

It is a rule in which how each element in a tuple is related.

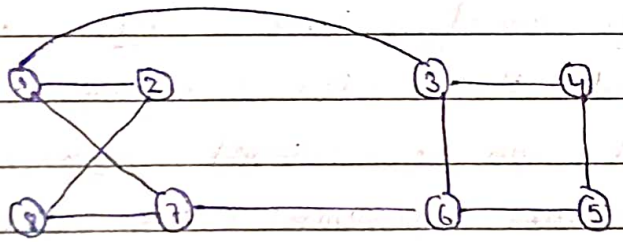
Explicit constraint: →

The rules that restrict each element to be chosen from given set.

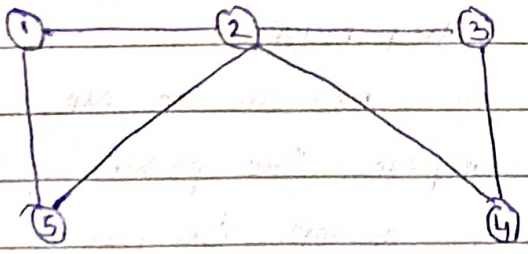
4b) Discuss Hamiltonian cycle. Also write an algorithm for finding Hamiltonian cycle of a graph.

→ Let G = (V, E) be a connected graph with n vertices. A Hamiltonian cycle is a round trip path along n edges of G that visits every vertex once and returns to initial or starting position. In other words, if a Hamiltonian cycle begins at some vertex $v_i \in G$ and vertices of G are visited in order $v_1, v_2, v_3 \dots v_{n+1}$. then edges $(v_i, v_{i+1})$ are in E, $1 <= i < n$, and $v_i$ are different except for $v_1$ and $v_{n+1}$ which are equal. Hamiltonian cycle was suggested by Sir William Hamilton.

G1:




G2:



The fig. shows a graph in which contains Hamiltonian cycle 1, 2, 8, 7, 6, 5, 4, 3, 1. The graph in 2 does not contain any Hamiltonian cycle. There is no easy way to find whether a given graph contains a Hamiltonian cycle. We have backtracking algo. that finds all the Hamiltonian cycle in a graph. The graph may be directed or undirected. only distinct cycle are output.

Algorithm:

```
Algorithm Hamiltonian (k)
{ repeat
  { Next value (k);
    if (x[k] = 0) then return;
    if (k = n) then write (x[1:n]);
    else Hamiltonian (k+1);
  } until (false);
}
```

5a) Comment on P = NP.

→ Every decision problem that is solvable by a deterministic polynomial time algo. is also solvable by a polynomial time non-deterministic algorithm. All problem in P can be solved with polynomial time algo., whereas all problems are intractable. It is not known whether P = NP. However, many problems are known in NP with property that if they belong to P, then it can be proved that P = NP.

If P ≠ NP there are problem in NP that are neither in P nor in NP-complete. The problem belongs to class P if it's easy to find a soln for problem.

The problem belongs to NP, if its easy to check a soln that may have been very tedious to find.

5b) Explain polynomial Reduction.

→ Let A and B are decision problem

A → • we like to solve in polynomial time
• not having a polynomial time algo.
• Let instance of A - α.

B → • having polynomial time algo. let instance of B - β.

- Converting α to β in polynomial time using reduction algo.
- Assume α → β conversion is happening in time of executing problem A so that A is solved in polynomial time.

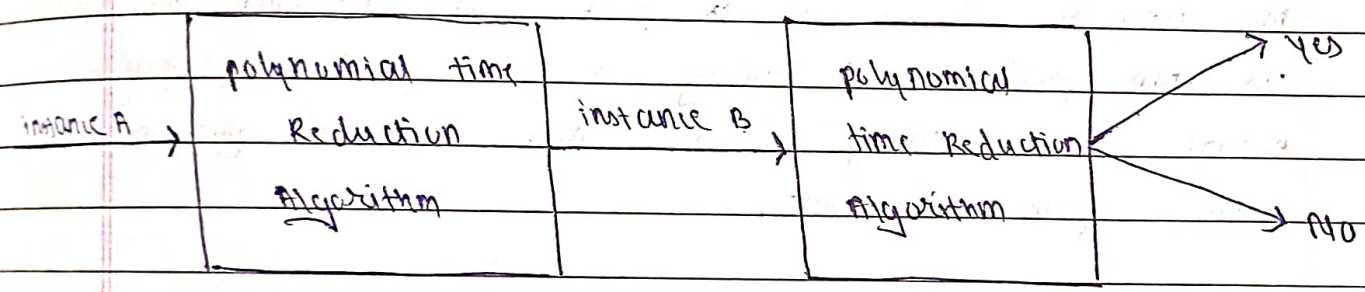



Fig: polynomial time algo. to decide A.

- Suppose that we have a procedure that transform α to β with following characteristic.

① The transformation takes in polynomial time.

② The answers are same i.e. the answer for α is 'yes' if the answer for β is also 'yes'.

Such a procedure is called polynomial time Reduction Algorithm.

5c) Give definations of NP hard and NP complete class of problems.

→ * NP-hard problem :→

Any given problem x acts as NP hard only if there exists a problem Y that is NP-complete. Here problem Y become reducible to problem X in a problem polynomial time. The hardness of an NP-complete hard is comp equivalent to that of NP-complete problem. But the NP-hard problems don't need to be in NP class.

* NP-complete problem :→

Any given problem x acts as NP-complete when there exists an NP problem Y. So that problem Y gets reducible to problem x in polynomial time. This means that a given problem can only become NP complete if it is a part of NP-hard as well as NP problems. A Turing machine of non-deterministic nature can easily solve this type of problem in a given polynomical time.

6a) Explain Non-deterministic algo. Give non-deterministic for searching and sorting problem.

→ A non-deterministic algo. can provide different outputs for same input on different executions. Unlike a deterministic algo. can provide different output for same input on different runs, a non-deterministic algo. travels in various routes to arrives at different outcomes.

Non-deterministic algo. are useful for finding approx. soln, when an exact soln is difficult or expensive to derive using a deterministic algorithm

Page:
Date:

Algorithm:

```
Algorithm nd-search (a,n,x)
{ i=1 to n do
  { j = select (a,n)
    if (a[j] = x]
      verification process success ();
  } failure ();
  }

Algorithm nd-sort (a,b,n)
{
  for i=1 to n do {
  j = select (a,n) b[i] = a[i]
  }
  for j=1 to n do { if (b[i] > b[i+1])
      failure ();
      } success (); }
```

(iii) Explain the concept of polynomial Reduction and how it can be used for showing NP completeness of problem.

→ From the definition of NP complete, it appears impossible to prove that a problem L in NP complete. By defination, it requires us to that show every problem in NP is polynomial time reducible to L. Fortunately there is an alternate way to prove it. The idea is to take a known NP-complete problem and reduce it to L

Assume $\alpha \longrightarrow \beta$ conversion is happening in time of executing problem A so that A is solved in polynomial time. Suppose that we have a procedure that transform $\alpha$ to $\beta$ with following characteristic.

(i) The transformation takes in polynomial time.

(ii) The answers are same. ie. the answer for α is 'yes' if the answer for is also 'yes'.

Such a procedure is called polynomial time Reduction Algorithm.

2) Find OBST with its cost and show all necessary matrices for given data.

| i | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $p_i$ | 0.3 | 0.2 | 0.1 | 0.1 |
| $q_i$ | 0.1 | 0.1 | 0.05 | 0.05 |

→ Step 1: compute matrix

The recursive function to compute matrix w to construct OBST by using dynamic programming is follows as:

$$w(i,j) = \begin{cases} q_{i-1}, & \text{if } j = i-1 \\ w[i,j-1] + p_j + q_j & \text{if } i < j \end{cases}$$

computation at diagonal -1

$w[i,j] = q_{i-1}$ if $j = i-1$

$w[1,0] = q_0 = 0.1$

$w[2,1] = q_1 = 0.1$

$w[3,2] = q_2 = 0.05$

$w[4,3] = q_3 = 0.05$

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| 1 | 0.1 | 0.4 | 0.55 | 0.7 | ↓ i |
| 2 | | 0.1 | 0.25 | 0.4 | |
| 3 | | | 0.05 | 0.2 | |
| 4 | | | | 0.05 | |

j →

computation at diagonal -2

$w[i,j] = w[i,j-1] + p_j + q_j$ if $i < j$

$w[1,1] = w[1,0] + p_j + q_j$

$w[1,1] = 0.1 + 0.2 + 0.1 = 0.4$

$w[2,2] = w[2,1] + 0.1 + 0.05 = 0.1 + 0.1 + 0.05 = 0.25$

$w[3,3] = w[3,2] + 0.1 + 0.05 = 0.05 + 0.1 + 0.05 = 0.2$

Page:
Date: / /

Computation at diagonal 3:

$w[i,j] = w[i,j-1] + p_j + q_j$ if $i \le j$

$w[1,2] = w[1,1] + 0.1 + 0.05 = 0.4 + 0.1 + 0.05 = 0.55$

$w[2,3] = w[2,2] + 0.1 + 0.05 = 0.25 + 0.1 + 0.05 = 0.4$

Computation at diagonal - 4:

$w[i,j] = w[i,j-1] + p_j + q_j$

$w[1,3] = w[1,2] + 0.1 + 0.05 = 0.55 + 0.1 + 0.05 = 0.7$

Step 2 :→ Compute matrix - e and matrix root

The recursive function to compute matrice to construct OBST by using dynamic programming is as follows:

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min \{ e[i,r-1] + e[r+1,j] + w[i,j] & \text{if } i \le j \end{cases}$$

r value = $i \le r \le j$

Computation at diagonal - 1:

$e[i,j] = q_{i-1}$ if $j = i-1$

$e[1,0] = q_0 = 0.1$

$e[2,1] = q_1 = 0.1$

$e[3,2] = q_2 = 0.05$

$e[4,3] = q_3 = 0.05$

$e[i,j]$

| 0 | 1 | 2 | 3 | |
|---|---|---|---|---|
| 0.1 | 0.6 | 1.05 | 1.6 | 1 |
| | 0.1 | 0.4 | 0.8 | 2 |
| | | 0.05 | 0.3 | 3 |
| | | | 0.05 | 4 |

Computation at diagonal - 2:

$e[i,j] = \min \{ e[i,r-1] + e[r+1,j] + w[i,j] \}$ if $i \le j$ $\quad i \le r \le j$

$e[1,1] = e[1,0] + e[2,1] + w[1,1] = 0.1 + 0.1 + 0.4 = 0.6$ for $r=1$

$e[2,2] = e[2,1] + e[3,2] + w[2,2] = 0.1 + 0.05 + 0.25 = 0.4$ for $r=2$

$e[3,3] = e[3,2] + e[4,3] + w[3,3] = 0.05 + 0.05 + 0.2 = 0.3$ for $r=3$

Page:
Date:

Computation at diagonal - 3:

$e[i,j] = \min\{e[i,r-1] + e[r+1,j] + w[i,j]\}$ if $i \le j$

$i < r \le j$

Two values of r are possible : $r=1$ & $r=2$

$e[1,2] = e[1,1] + e[3,2] + w[1,2] = 0.6 + 0.05 + 0.05 = 1.25$ for $r=2$

$e[1,2] = e[1,0] + e[2,2] + w[1,2] = 0.1 + 0.4 + 0.55 = 1.05$ for $r=1$

$e[1,2] = \min(1.05, 1.2) = 1.05$ for $r=1$

Two values of r are possible : $r=2$ and $r=3$

$e[2,3] = e[2,1] + e[3,3] + w[2,3] = 0.1 + 0.3 + 0.4 = 0.8$ for $r=2$

$e[2,3] = e[2,2] + e[4,3] + w[2,3] = 0.4 + 0.05 + 0.4 = 0.85$ for $r=3$

$e[2,3] = \min(0.8, 0.85) = 0.8$ for $r=2$

Computation at diagonal - 4:

$e[i,j] = \min\{e[i,r-1] + e[r+1,j] + w[i,j]\}$ for $i \le j$

$i < r \le j$

There are three values of r are possible

$r = 1, 2, 3.$

If $r=1$:

$e[1,3] = e[1,0] + e[2,3] + w[1,3]$

$= 0.1 + 0.8 + 0.7 = 1.6$

If $r=2$:

$e[1,3] = e[1,1] + e[3,3] + w[1,3]$

$= 0.6 + 0.3 + 0.7 = 1.6$

If $r=3$:

$e[1,3] = e[1,2] + e[4,3] + w[1,3]$

$= 1.05 + 0.05 + 0.7$

$= 1.8$

$e[1,3] = \min(1.6, 1.6, 1.8) = 1.6$ for $r=1$ & $r=2$

| j → | 0 | 1 | 2 | 3 | i ↓ |
|---|---|---|---|---|---|
| | 0.1 | 0.4 | 0.55 | 0.7 | 1 |
| | | 0.1 | 0.25 | 0.4 | 2 |
| | | | 0.05 | 0.2 | 3 |
| | | | | 0.05 | 4 |

w(i,j)

| j → | 0 | 1 | 2 | 3 | i ↓ |
|---|---|---|---|---|---|
| | 0.1 | 0.6 | 1.05 | 1.6 | 1 |
| | | 0.1 | 0.4 | 0.8 | 2 |
| | | | 0.05 | 0.3 | 3 |
| | | | | 0.05 | 4 |

e(i,j)

| j → | 1 | 2 | 3 | i ↓ |
|---|---|---|---|---|
| | 1 | 1 | 1,2 | 1 |
| | | 2 | 2 | 2 |
| | | | 3 | 3 |

root (r,j)

Step 3: Construct OBST from root matrix

Now we have total three keys as $(a_1, a_2, a_3)$ we wants to find the root among $(a_1, a_2, a_3)$

Therefore, root $(a_1, a_2, a_3)$ = root $(1,3) = 1,2$

Thus, OBST configuration will be