Date - 25/01/23

Ex. No.: 1 Implementation of The Two Jug Problem

- **Aim:** A person has a m liter jug and a n liter jug where $0 < m < n$. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. The person has to use the jugs to measure d liters of water where $d < n$. Determine the min. no. of operations to be performed to obtain d liters of water in one jug.

- **Algorithm:**

This problem can modeled using Diophantine equation of the form $mx + ny = d$, which is solvable if and only if gcd(m, n) divides d. The solution of x, y is given using the Extended Euclid algorithm.

$$mx + ny = d$$

m, n = capacity of the jugs.
d = capacity liquid we have to get.
x, y = no. of times we have to fill and empty the jugs J1 and J2.

Now to find the min. no. of operations to be performed we have to decide which jug should be filled first, depending on which we'll have 2 solutions and min will be choosen among them.

**Solution 1** (Always pour from m liter jug to n liter jug)

1. Fill the m liter jug and empty it into n lite jug.

2. Whenever the m liter jug becomes empty fill it.
3. Whenever the n liter jug becomes full empty it.
4. Repeat steps 1, 2, 3 till either n liter jug or the m liter jug contain d litres of water.

Lets say Algo 1 achieves the task in C1 no. of operations.

2. Solution 2 (Always pour from n liter jug into m liter jug)

1. Fill the n liter jug and empty it into m liter jug.
2. Whenever the n liter jug becomes empty fill it.
3. Whever the m liter jug becomes full empty it.
4. Repeat steps 1, 2, and 3 till either n liter jug or the m liter jug contains d liters of water.

Let's say solution 2 achieves the task in C2 no. of operation.

∴ Now final solution will be min(C1, C2).

Exp-2

01/02/23

# Travelling Salesman Problem

- **Aim:** Given a set of cities and the distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

- **Algorithm:**

Let the given set of vertices be $\{1, 2, 3, 4, \ldots, n\}$. Let us consider 1 as starting and ending point of output. For every other vertex I, we find the min. cost path with 1 as the starting point, I as the ending point, and all vertices appearing exactly once. Let the cost of this path cost(i), and the cost of the corresponding cycle would cost(i) + dist(i,1), where dist(i,1) is the distance from I to 1. Finally we return the minimum of all [cost(i) + dist(i+1)] values.

Let us define a term C(S,i) be the cost of the min cost path visiting each vertex in set S exactly once, starting at 1 and ending at i. We start with all subsets of size 2 and calculate C(S,i) for all subsets where S is the subset, then we calculate C(S,i) for all subsets S of size 3 and so on.

If size of S is 2, then S must be (1,i).

$C(S,i) = dist(1,i)$

Else if size of S is greater than 2.

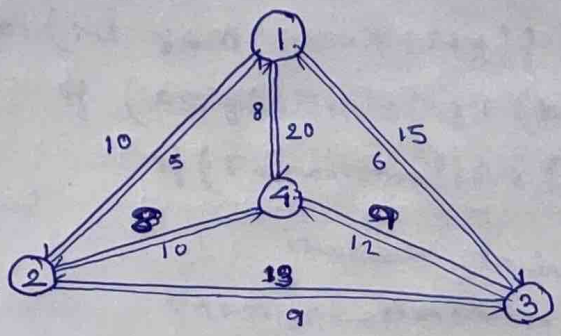
$C(S,i) = \min\{C(S-\{i\}, Y) + dis(Y,i)\}$ where

Y belongs to S, $Y != i$ and $Y != 1$.

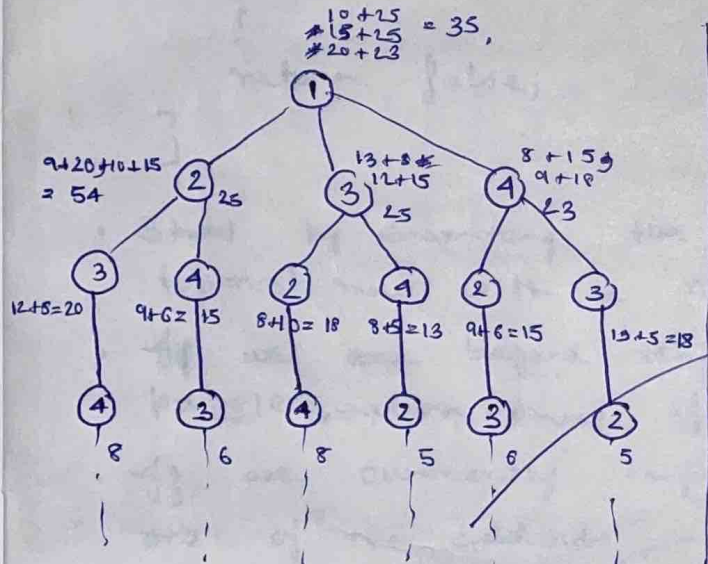
Dynamic programming solution using top-down recursive + memoized approach.

For maintaining the subsets we can use the bitmasks to represent the remaining nodes in our subset. Since bits are faster to operate and there are only few nodes in graph, bit masks is better to use.

- Manual Output:




$$A = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$




$$g(1,\{2,3,4\}) = \min_{k \in \{2,3,4\}} \{c_{1k} + g(k, \{2,3,4\} - \{k\})\}$$

$$g(i,S) = \min_{k \in S} \{c_{ik} + g(k, S-\{k\})\}$$

$\therefore g(2,\phi) = 5$
$g(3,\phi) = 6$
$g(4,\phi) = 8$
$g(2,\{3\}) = 15$
$g(2,\{4\}) = 8$
$g(3,\{2\}) = 5$
$g(3,\{4\}) = 8$
$g(4,\{2\}) = 5$
$g(4,\{3\}) = 6$

$g(2,\{3,4\}) = 15$
$g(3,\{2,4\}) = 25$
$g(4,\{2,3\}) = 23$

$\therefore g(1,\{2,3,4\})$
$= 35$

- Result: Thus, the Travelling Salesman Problem was implemented.

Exp-3. Date - 02/02/23

# CSP: Cryptarithmetic Puzzle

- Aim: To assign each letter a digit from 0 to 9 so that the arithmetic works out correctly

- Algorithm:

```
bool EnhaustiveSolve(puzzleT puzzle, string lettersToAssign)
{
  if (lettersToAssign.emtpy())
      return PuzzleSolved(puzzle);
  for (int i=0; i<=9; i++){
    if (AssignLetterToDigit(lettersToAssign[0], i)){
      if (EnhaustiveSolve(puzzle, lettersToAssign.substr(1)))
          return true;
      UnassignLetterFromDigit(lettersToAssign[0], i);
    }
  }
  return false;
}
```

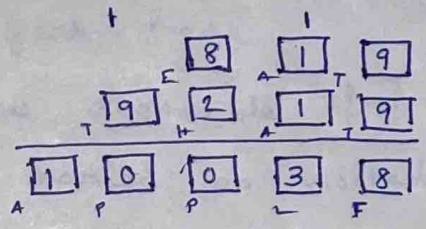
- Start by examining the rightmost digit of the topmost row, with a carry of 0.
- If we are beyond the leftmost digit of the puzzle, return true if no carry, false otherwise
- If we currently trying to assign a char in one of the addends
  If char already assigned, just recur on the row beneath this one, adding value into the sum.
  If not assigned the
- for (every possible choice among the digits not in use)
  make that choice and then on row beneath this one, if successful, return true if ! successful, unmake assignment and try another digit.

- return false if no assignment worked to trigger backtracking.
- Else if try to assign a char in the sum
- If char assigned & matches correct, recur on next column to the left with carry, if success return true,
- If char assigned & doesn't match, return false
- If char unassigned & correct digit already used, return false.
- If char unassigned & correct digit unused, assign it and recur on next column to left with carry, if success return true.
- return false to trigger backtracking.

- **Manual Output:**

```
   EAT
+ THAT
-------
 APPLE
```

E - 8
A - 1
T - 9
H - 2
P - 0
L - 3




T + T = E
A + A = L

- Result: Hence, the Cryptarithmatic problem was implemented.

16/2/23

Exp- 4

Date - 14-02-23

# 8 Puzzle Problem using DFS & BFS

- Aim: Given a 3×3 board with tiles and one empty space, the objective is to place numbers on tiles to match the final configuration using the empty space, using both DFS and BFS.
- Algorithm:

DFS :- // recursive

- mark node v as visited; //configurations generated // before, not done in "tree // search mode
- If v is "goal" return success;
- operate on v; // eg. evaluate if it is the goal mode
- for each node w accessible from node v do
  if w is not marked as visited then
  DFS(w);

  end for;
  End Algorithm.

BFS : S = start node

- Initialize a Q with s;
- v = Pop(Q); // BFS is Queue based algorithm
- If v is "goal" return success;
- mark node v as visited;
  // config. generated before,
- operate on v;

- for each node w accessible from node v d
  dif w is not marked as visited then
  Push w at the back of Q;

end for.

- Manual Output:

Initial

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 0 | 5 |
| 7 | 2 | 6 |

Final

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

DFS:

Initial

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 0 | 5 |
| 7 | 2 | 6 |

↓

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 2 | 5 |
| 7 | 0 | 6 |

↓

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 2 | 5 |
| 7 | 6 | 0 |

↓

| | | |
|---|---|---|
| 1 | 3 | 4 |
| 8 | 2 | 0 |
| 7 | 6 | 5 |

Final.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

↑

| | | |
|---|---|---|
| 1 | 2 | 5 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

↑

| | | |
|---|---|---|
| 1 | 0 | 3 |
| 8 | 2 | 4 |
| 7 | 6 | 5 |

↑

| | | |
|---|---|---|
| 1 | 3 | 0 |
| 8 | 2 | 4 |
| 7 | 6 | 5 |

6 moves.

Exp-5

Date-21/02/23

# Best First Search & A* Search

- Aim: To study Best First Search and A* Search

- Algorithm:

Best First Search -

```
Best-First-Search (Graph g, Node start)
  1. Create an empty PriorityQueue.
     PriorityQueue pq;
  2. Insert "start" in pq.
     pq.insert(start)
  3. Until PriorityQueue is empty
       u = PriorityQueue.DeleteMin
       If u is the goal
          Exit
       Else
          For each neighbor v of u
             If v "Unvisited"
                Mark v "Visited"
                pq.insert(v)
          Mark u "Examined"
  End procedure.
```

A* Search -

1. Initialize the open list
2. Initialize the closed list
   put the starting node on the open list (you can leave its f at zero).

3. while the open list is not empty
   a. find the node with least f on the open list, call it "q".
   b. pop q off the open list.
   c. generate q's 8 successors and set their parents to q.
   d. for each successor
      i. if successor is the goal, stop search
      ii. else, compute both g and h for successor.

          successor.g = q.g + distance between successor and q.

          Successor.h = distance from goal to successor

          Successor.f = successor.g + successor.h
      iii. if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor.
      iv. if a node with the same position as successor is in the CLOSED List which has a lower f than successor, skip this successor otherwise, add the node to the open list

   end (for loop)

   e. push q on the closed list

end (while loop)

Exp-6 Date - 10/02/23

# Implementation of Fuzzy Logic

- Aim: To implement Monty Hall problem using Fuzzy Logic.
- Algorithm:

A = dist(). dist dictionary A, B, x, y

→ Initiate the fuzzy set c to be empty

→ For each rule RI

ⓐ Determine the degree of membership of each input value in corresponding fuzzy set A.

ⓑ Determine minimum degree of membership of all input variables.

ⓒ Evaluate consequence of rules using minimum degree or membership as degree at activation.

→ Normalize Resulting Fuzzy

→ Return fuzz setc as a result.

- Result: Thus Monty Hall problem using fuzzy logic was implemented.

Verified
Sj
10/3/23

Exp-7

Date - 23/03/23

# Implementation of Unification and Resolution

- Aim: To implement unification and resolution for a given set of statements.

## Exp-7(a) [Unification]

Algorithm -

1. Initialize the substitution set to empty.
2. Recursively unify atomic sentences.
   - check for identical expression matches.
   - If one expression is available $V_i$, and the other is a team to which it does not contain variable $V_i$, then:
   - Substitute $t_i/V_i$ in the existing substitutions.
   - If both the expressions are functions, then function name must be similar, and the no. of arguments must be the same in both the expressions. For each pair of the following atomic sentences find the most general unifier (If exists).

## Exp-7(b) [Resolution]

Algorithm -

Resolution

3+3+4 = 10

## Exp-7(b)

- Aim: To implement resolution Algorithm.
- Algorithm:

Resolution is used if various statements are given, and we need to prove a conclusion of statement. Resolution is a single inference code which can effectively operate

→ Conversion of fact into first-order logic
→ Convert FOL statements into CNF.
→ Negate the statement which needs to prove (proof by contradiction)
→ Draw resolution graph (Unification).

- Result: Successfully implemented both unification and resolution.

Exp-8

Date - 30/3/23

# Learning Algorithm of a Application

- Aim : To implement learning algorithm for an application.
- Learning Models Used -
  1. Logistic Regression
  2. Support Vector Machine (SVM)
  3. Naive Bayes,

1 Logistic Regression - is a statistical method that is used for building ML models where the dependent variable is dichotomous ie binary. It uses the sigmoid function,

$$f(x) = \frac{1}{1+e^{-x}}$$

as activation function, so it gives the probabilistic values which lie between 0 and 1,

Code :

```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegressio
lr = LogisticRegression()
lr.fit(X_train, Y_train)
Y_pred = lr.pred(x_test).
score_lr = round(accuracy_score(Y_pred, Y_test)
                 * 100, 2).
score_lr.
```

o/p

Verified

2. Support Vector Machine (SVM): Is a linear model for classification and regression problems. The algorithm creates a line or a hyperplane that seperates the data into classes. It is a type of Supervised Learning.

Code -

```
from sklearn import svm

sv = svm.SVC(kernel='linear')
sv.fit(X_train, Y_train)
Y_pred = sv.predict(X_test)
score = round(accuracy_score(Y_pred, Y_test)
                * 100, 2)
```

3 Naive Bayes: algorithm is a supervised algorithm, which is based on Bayes Theorem and is used for classification. It is used in text classification that includes high-dimensional training dataset.

Code -

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, Y_train)
Y_pred = nb.predict(X_test)
score = round(accuracy_score(Y_pred, Y_test)
                *100, 2).
```

- Result: Thus, the above three algorithms have been implemented.