VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA.

Name K.S.I.SIVANI Roll No. -052 Page No.

PRELAB QUESTIONS-1)

1) Compare explicit and implicit constraints of a backtracking solution:

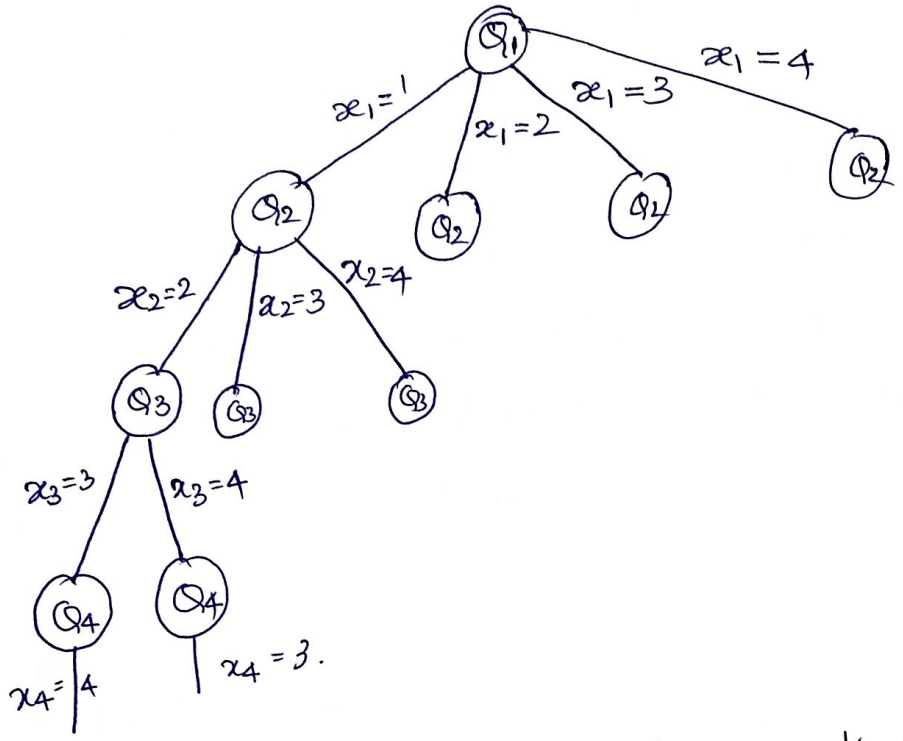
→ Explicit constraints are conditions or rules explicitly defined and specified in the problem statement.

* Characteristics of explicit constraints:
  Clearly stated, Precise and specific, deterministic, used for pruning

→ Implicit constraints are additional constraints or restrictions that are not explicitly stated in the problem statement but can be deduced or inferred based on the problem context.

* Characteristics of implicit constraints:
  indirectly derived, general and abstract, probabilistic.

2) Draw the state space tree to show a soln for 4-Queens problem:



3) Write the steps to solve Knapsack problem by using backtracking:

→ Define the problem

→ Implement the backtracking
   a) Recursive backtrack algorithm
   b) check for the base cases.
   c) Choose an item from the remaining items and add it to Knapsack.

→ Call the Backtrack algorithm

→ Retrieve and O/P the best solution.

4) Compare the Brute Force Approach with backtracking:

→ Brute Force Approach:

1) Exhaustive Search
2) No intelligent pruning
3) High computational cost
4) Simple Implementation
5) Guarantees Optimal solution

→ Backtracking:

1) Intelligent Search
2) Pruning & optimization
3) Faster computation
4) Recursive Implementation
5) May not guarantee optimality.

* PRELAB PROGRAMS-10:

1) Implement Hamiltonian cycle by using Backtracking:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 20
void printSolution (int path[], int n)
{ printf("Hamiltonian cycle:");
  for(int i=0; i<n; i++)
    { printf("%d ", path[i]); }
    printf("%d\n", path[0]); }
```

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052 Page No.

```
bool isSafe(int v, int graph[MAX_VERTICES][MAX_VERTICES],
            int path[], int pos, int n)
{ if(graph[path[pos-1]][v] == 0)
        return false;
  for(int i = 0; i<pos; i++)
  { if(path[i] == v) return false; }
  return true; }
bool hamiltonianCycleUtil(int graph[MAX_VERTICES]
                                   [MAX_VERTICES],
                          int path[], int pos, int n)
{ if(pos == n)
  { if(graph[path[pos-1]][path[0]] == 1)
        return true;
    else
        return false; }
  for(int v = 1; v<n; v++)
  { if(isSafe(v, graph, path, pos, n))
    { path[pos] = v;
      if(hamiltonianCycleUtil(graph, path, pos+1, n))
          return true;
```

```
    path[pos] = -1; }
} return false; }

void HamiltonianCycle(int graph[MAX_VERTICES]
                              [MAX_VERTICES],
                      int n)
{ int path[MAX_VERTICES];
  for(int i=0; i<n; i++)
  { path[i] = -1; }

  path[0] = 0;
  if(hamiltonianCycleUtil((graph, path, 1, n))
  {printSolution(path, n); }

  else
   { printf("No Hamiltonian Cycle found\n"); }
 }
int main()
{ int n;
  printf("Enter the no. of vertices in graph:");
  scanf("%d", &n);

  printf("Enter adjacency matrix of the
          graph:\n");
```

```
int graph[MAX_VERTICES][MAX_VERTICES];
for(int i=0; i<n; i++)
{ for(int j=0; j<n; j++)
  {scanf("%d", &graph[i][j]);}
  }hamiltonianCycle(graph, n);
 return 0;}
```

2) Implement Sum of subsets problem using backtracking:

```
#include<stdio.h>
#include<stdbool.h>
void printSubset(int set[], int subset[], int n)
 { printf("Subset:");
  for(int i=0; i<n; i++)
  { if (subset[i])
     {printf("%d", set[i]);}
    printf("\n");}
void subsetSumUtil(int set[], int subset[],
                   int n, int sum, int currentSum,
                   int index)
{ if (currentSum == sum)
```

① Enter the no. of vertices in the graph: 4

Enter the adjacency matrix of the graph::

0 1 1 1

1 0 1 0

1 1 0 1

1 0 1 0.

Hamiltonian Cycle: 0 1 2 3 0.

```
int graph[MAX_VERTICES][MAX_VERTICES];
for(int i=0; i<n; i++)
{ for(int j=0; j<n; j++)
  { scanf("%d", &graph[i][j]); }
  }hamiltonianCycle(graph, n);
 return 0; }
```

2) Implement Sum of subsets problem using backtracking:

```
#include<stdio.h>
#include<stdbool.h>
void printSubset(int set[], int subset[], int n)
 { printf("Subset:");
  for(int i=0; i<n; i++)
  { if (subset[i])
     { printf("%d", set[i]); }
    printf("\n"); }
void subsetSumUtil(int set[], int subset[],
                   int n, int sum, int currentSum,
                   int index)
{ if (currentSum == sum)
```

```
    printSubset(set, subset, n);
    return;}
  if (index == n)
      return;
  if (currentSum + set[index] <= sum)
   { subset[index] = 1;
     subsetSumUtil (set, subset, n, sum, currentSum
                                    + set[index],
                                    index + 1);

     subset[index] = 0;}
  subsetSumUtil (set, subset, n, sum, currentSum,
                                    index + 1);

void subsetSum (int set[], int n, int sum)
{ int subset[n];
   subsetSumUtil (set, subset, n, sum, 0, 0);}

int main()
{ int n;
  printf("Enter the no. of elements in the set:");
  scanf("%d", &n);
  int set[n];
```

```
printf("Enter the elements of the set:\n");
for(int i=0; i<n; i++)
{scanf("%d", &set[i]);
}
int sum;
printf("Enter the target sum: ");
scanf("%d", &sum);
subsetSum(set, n, sum);
return 0;}
```

O/P:

```
Enter the no. of elements in the set: 4
Enter the elements of the set:
1 2 3 4
Enter the target sum: 5
Subset: 1 4
Subset: 2 3
Subset: 5.
```

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052 Page No.

PRELAB QUESTIONS-12

1) What is the difference between Branch & Bound and Backtracking?

→ Backtracking is an algorithm used to solve the decision problem whereas Branch and Bound technique is an algorithm used to solve the optimization problem

2) What is the significance of upper and lower bounds in exploring a node in BB?

→ Upper Bound is an estimate of the best possible solution value achievable within a particular subtree or node.

→ Upper bounds help to reduce the search space and improve the efficiency of algorithm.

→ Lower Bound is an estimate of minimum possible solution value that can be obtained in a subtree or node.

→ Lower bounds help in bounding the search space and improve the algorithm to focus on more promising branches.

VASAVI COLLEGE OF ENGINEERING

(AUTONOMOUS)
(Affiliated to Osmania University)
Hyderabad - 500 031.

DEPARTMENT OF : CSE

NAME OF THE LABORATORY : DAA

Name K.S.I.SIVANI Roll No. -052 Page No.

PRELAB PROGRAMS -12

1) Implement TSP by branch and bound:

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>
#define MAX_N 10
int n;
int graph[MAX_N][MAX_N];
int minCost= INT_MAX;
int bestPath[MAX_N];
void swap(int *a, int *b)
{ int temp=*a; *a=*b; *b=temp;}
int calculatePathCost(int path[])
{ int cost=0;
  for(int i=0;i<n-1;i++)
  { cost +=graph[path[i]][path[i+1]] ;}
   cost+=graph[path[n-1]][path[0]];
   return cost ;}
void TSPBBUtil(int path[], bool visited[], int level,
                int cost)
```

```
if(level==n)
{ int currentCost = cost + graph[path[level-1]][path[0]];
  if(currentCost < minCost)
  { minCost = currentCost;
    for(int i=0; i<n; i++)
    { bestPath[i] = path[i]; } }
  return; }
for(int i=0; i<n; i++)
{ if(!visited[i])
  { path[level] = i;
    visited[i] = true;
    int newCost = cost + graph[path[level-1]][i];
    int lowerBound = 0;
    for(int j=0; j<n; j++)
    { if(!visited[j])
      { int minEdgeCost = INT_MAX;
        for(int k=0; k<n; k++)
        { if(graph[j][k] < minEdgeCost && j!=k)
          { minEdgeCost = graph[j][k]; }}
```

```
    lowerBound += minEdgeCost; }}
    if (newCost + lowerBound < minCost)
    { TSPBBUtil(path, visited, level+1, newCost); }
      visited[i] = false; }}}
void TSPBB (int startingCity)
{ int path[MAX_N];
  bool visited[MAX_N];
 for(int i=0; i<n; i++){ visited[i] = false; }
 path[0] = startingCity;
 visited[startingCity] = true;
 TSPBBUtil(path, visited, 1, 0);
 printf("Optimal TSP path: ");
  for (int i=0; i<n; i++){printf("%d", bestPath[i]); }
  printf("%d", bestPath[0]); }
  printf("Optimal TSP cost: %d \n", minCost); }
int main()
{ printf("Enter the no. of cities: "); scanf("%d", &n);
  printf("Enter the adjacency matrix of distances
                  b/w cities: \n");
```

```
for(int i =0; i<n; i++)
{for(int j=0; j<n; j++){scanf("%d",&graph[i][j]);}}
int startingCity;
printf("Enter the starting city(0-%d): ", n-1);
scanf("%d", &startingCity);
TSPBB(startingCity);
return 0;}
```

2) Implement 0/1 Knapsack using branch and bound.

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_ITEMS 100
typedef struct{int weight; int value;} Item;
int maxProfit=0;
int bestItems[MAX_ITEMS]; int numItems;
int capacity;
void BBKnapsack(Item items[], int level, int currentWeig
                int currentValue, bool selected[])
{ if(currentWeight>capacity){return;}
  if(currentValue > maxProfit)
```

① Enter the no. of cities : 4

Enter the adjacency matrix of distance b/w cities:

0 10 15 20

10 0 35 25

15 35 0 30

20 25 30 0.

Enter the starting city (0-3) : 0

Optimal TSP Path : 0 1 3 2 0

Optimal TSP cost : 80

```
maxProfit = currentValue;
    for(int i=0; i<numItems; i++)
      { bestItems[i] = selected[i]; } }
  if( level == numItems){ return; }
double bound = currentValue;
int remainingWeight= capacity - currentWeight;
int i = level;
While(i<numItems && remainingWeight >0)
{ if (items[i].weight <= remaining Weight)
   { bound += items[i].value;
    remainingWeight -= items[i].weight ; }
   else{ bound+ =(double) items[i].value / items[i].weight
                                    *remainingWeight;
remainingWeight =0; }
 i++; }
if ( bound <= maxProfit) {return ; }
selected[level] = true;
BBKnapsack (items, level+1, currentWeight + items[level].
                                          weight,
          currentValue + items[level].value, selected);
```

```
    selected[level] = false;
BBKnapsack(items, level + 1, currentWeight, currentValue,
                selected); }

void Knapsack(Item items[], int num, int cap)
{ numItems = num;
  capacity = cap; bool selected[MAX_ITEMS] = {false};
  BBKnapsack(items, 0, 0, 0, selected);
  printf("Optimal items: ");
  for(int i=0; i<numItems; i++)
  { if(bestItems[i]) {printf("%d", i);}}
  printf("\n");
  printf("Optimal Profit: %d\n", maxProfit); }
int main()
{ int num, cap;
  printf("Enter the no. of items:");
  scanf("%d", &cap); Item items[MAX_ITEMS];
printf("Enter the weight and value of each
        item:\n");
```

```
for(int i=0; i<num; i++)
{scanf("%d %d", &items[i].weight,
                &items[i].value);
}
knapsack(items, num, cap);
return 0; }.
```

O/P:

```
Enter the no. of items: 4
Enter the capacity of the knapsack: 5
Enter the weight & value of each item:
2 3
1 2
3 4
2 2
Optimal Items: 0 2 3
Optimal Profit: 8.
```