Harkirat Kaur 1905334
Btech D3 IT A2 1921036

Date
Page

Assignment 02

Q1 Illustrate the use of backtracking technique in solving knapsack problems.

Ans: When n elements are given which have w_i as weights associated with them & p_i as profits associated with them.

It is required to fill the knapsack of capacity 'm' such that:-

$$\sum_{1 \le i \le n} w_i x_i \le m \quad \text{and} \quad \sum_{1 \le i \le n} p_i x_i \text{ is maximized}$$

& x_i = is either 0 / 1.

The backtracking technique to solve this problem involves a recursive function:-

```
void BKnap (int k, float cp, float cw).
```

Recursive Backtracking Algorithm for knapsack problem,

cp → current profit total
cw → current weight total

Invoked As:-

```
BKnap (1, 0, 0)
```

```
void BKnap (int k, float cp, float cw)
{
```

```
// Generate left child
if (cw + w[k] <= m)                    // if feasible?
{
    y[k] = 1;                          // include element
    if (k<n)
    {
        BKnap(k+1, cp + p[k], cw + w[k]);
    }
    if ((cp + p[k] > fp) && (k==n))
    {
        fp = cp + p[k];
        fw = cw + w[k];
        for (int j=1; j<=k; j++)
        {
            x[j] = y[j];
        }
    }
}
// Generate right child
if (Bound(cp, cw, k) >= fp)
                                       // if not feasible
{                                      // compute upper bound using
    y[k]=0                             greedy method.
                                       // donot include element.
    if (k<n) {
        BKnap(k+1, cp, cw);
    }
    if ((cp > fp) && (k==n)) {
        fp = cp;
        fw = cw;
```

```
for (int j=1; j<=k; j++) {
        x[j] = y[j]
    }
}
}
```

Steps to solve Knapsack problem using backtracking :-

- Arrange items in decreasing order of P_i/w_i

$$\left(\text{i.e. } \frac{P_i}{w_i} \geq \frac{P_{i+1}}{w_{i+1}}\right)$$

- Find Upper Bound Value :-
The UB value is a value which can be generated by applying greedy algorithm & relaxing the requirement as $0 \leq x_i \leq 1$.
i.e. allowing fractions.

- Generate state space tree.

- Follow Depth-first-Generation while expanding state space tree.

Example of a 0/1 Knapsack problem being solved by using backtracking algorithm design technique :-

Item	x_1	x_2	x_3	x_4
P	45	30	45	10
w	3	5	9	5
P/w	15	6	5	2

Capacity of Knapsack - $m = 16$

Step 1: $P_1/w_1 \geq P_2/w_2 \geq P_3/w_3 \geq P_4/w_4$

for all items :- x_1, x_2, x_3, x_4

Step 2: Upper Bound at root node :-

UB	x_1	x_2	x_3
P	45	30	$45 * 8/9 = 40$
W	3	5	8 out of 9 i.e. 8/9
		wt =	$3+5+\frac{8}{9}\times 9 = 16 = m$

UB = 45 + 30 + 40 = 115

So, kill the node

So, kill the node

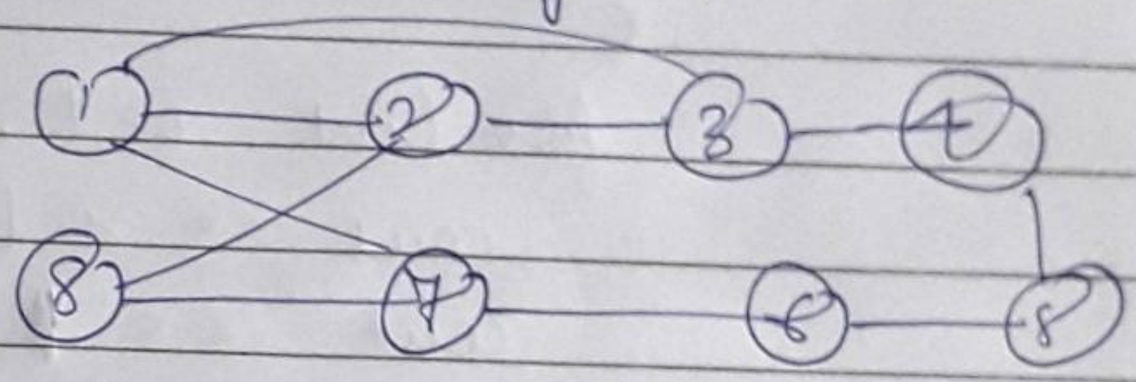
Answer Tuple is $(1\ 0\ 1\ 0)$ for $x_1\ x_2\ x_3\ x_4$

Ques2. Define a Hamiltonian cycle. Explain how backtracking can be used to find the Hamiltonian cycles in a graph

Ans2. 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 & returns to its starting position.

For Example :-

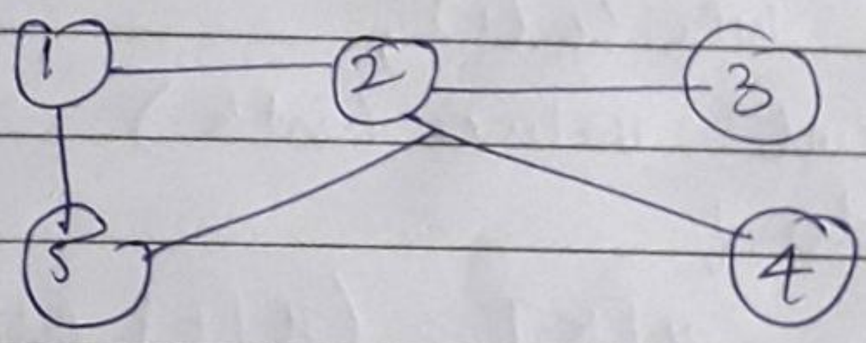
Graph G1 :-



This graph contains following Hamiltonian cycle :-

1, 2, 8, 7, 6, 5, 4, 3, 1

Graph G2 :-



The above graph contains no Hamiltonian cycle.

The backtracking algorithm finds all the Hamiltonian cycles in a graph. The graph may be directed or undirected. Only distinct cycles are output.

The worst case time complexity of the backtracking algorithm to find all Hamiltonian cycles in a graph is $O(n!)$

Backtracking Algorithm to find all Hamiltonian cycles in a graph:-

```
void Hamiltonian (int k)
{
do {
    Next Value (k);
    if (!x [k]) return;
    if (k == n)
    {
    for (int i=1; i<=n; i++)
        cout << x [i] << " ";
        cout << "1", cout << "\n";
}
else Hamiltonian (k+1);
}
while (1);
```

```
=> void Next value (int k)
    do {
        x[k] = (x[k] + 1) % (n+1);
    if (!x[k]) return;
    if (G [x [k-1]] [x [k]])
{
for (int j=1; j<=k-1; j++)
    if (x [j] == x[k]) break;
    if (j==k)
    {
if ((k<n) || ((k==n) && G[x[n]]
                        [x [1]]))
return;
}
} while (1);
}
```