CSE- 3004 LAB-7 Assignment

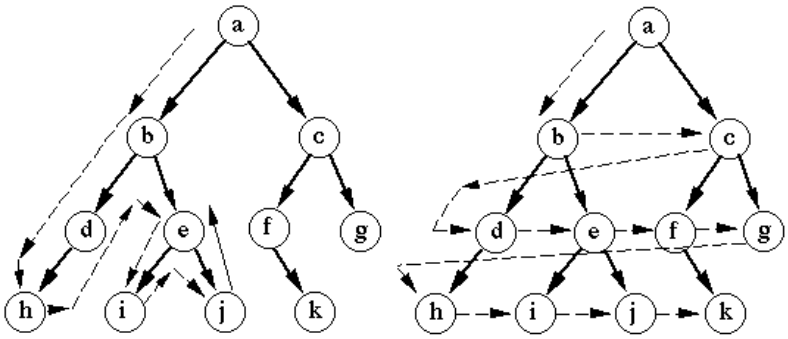
Academic year: 2020-2021 **Semester:** WIN

Faculty Name: Dr. D Sumathi Mam **Date:** 2 /9/2021

Student name: M.Taran **Reg. no.:** 19BCE7346

BRANCH AND BOUND

1.) Implement 0-1 knapsack using branch and bound:



Depth-first search Breadth-first search

a) first, implement BFS with branch and bound pruning.
Find out the behaviour. N= 4 W=16

i	p_i	w_i	$\frac{p_i}{w_i}$
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

```
import java.io.*;
import java.util.*;
class
node {
    int level;
    int profit;
```

```
    int weight;
    int bound;
}
class Main {
    public static void main(String args[]) throws
    Exception {
        int maxProfit;
        int N;
        int W;
        int wt;
        int vl;
        int maxVal;
        Scanner input = new Scanner(System.in);
        System.out.println("Please enter the number of
            items: "); N = input.nextInt();
            System.out.println("Please enter the capacity of the
                Knapsack: "); W = input.nextInt();
                int[] Vl = new int[N]; int[] Wt =
                new int[N];
                for (int i = 0; i < N; i++) {
                    System.out.println("Please enter the weight anc value
of item " + i +
                        ": ");
                    wt = input.nextInt();
                    vl = input.nextInt();
                    Wt[i] = wt;
                    Vl[i] = vl;
                }
                //for(int i = 0; i < 1000; i++){
                maxVal = knapsack(N, Vl, Wt, W);
                //}
                System.out.println(maxVal);
            }
      public static int bound(node u, int n, int W, int[] pVa, int[] wVa) {
                int j = 0, k = 0;
                int totweight =
                    0;
                int result = 0;
                if (u.weight >=
                    W) {
                    return
                    0;
```

```
            } else {
                result =
                    u.profit;
                j =
                    u.level + 1;
                totweight = u.weight;
                while ((j < n) && (totweight + wVa[j] <=
                        W)) {
                    totweight = totweight +
                        wVa[j];
                    result = result +
                        pVa[j];
                    j++;
                }
                k = j;
                if (k < n) {
              result = result + (W - totweight) * pVa[k] / wVa[k];
                }
                return result;
            }
        }
        public static int knapsack(int n, int[] p, int[] w,
            int W) {
            Queue < node > Q = new
            LinkedList < node > ();
            node u = new node();
            node v = new node();
            int[] pV = new
            int[p.length];
            int[] wV =
                new int[w.length];
            Q.poll();
            for (int i = 0; i < n; i++) {
                pV[i] =
                    p[i];
                wV[i] = w[i];
            }
            v.level = -1;
            v.profit = 0;
            v.weight = 0;
            int maxProfit = 0;
            //v.bound = bound(v, n, W, pV,
```

```
            wV); Q.add(v);
        while (Q.size() >
            0) {
            v =
                Q.remove();
            if (v.level == -
                1) {
                u.level = 0;
            } else if (v.level != (n -
                    1)) {
                u.level =
                    v.level + 1;
            }
            u.weight = v.weight +
                w[u.level];
            u.profit = v.profit +
                p[u.level];
            u.bound = bound(u, n, W, pV, wV);
            if (u.weight <= W && u.profit >
                maxProfit) {
                maxProfit = u.profit;
            }
            if (u.bound >
                maxProfit) {
                Q.add(u);
            }
            u.weight = v.weight;
            u.profit = v.profit;
            u.bound = bound(u, n, W, pV, wV);
            if (u.bound >
                maxProfit) {
                Q.add(u);
            }
        }
        return maxProfit;
    }
}
```

```
Result
compiled and executed in 58.08 sec(s)

Please enter the number of items:
4
Please enter the capacity of the Knapsack:
6
Please enter the weight anc value of item 0:
3
10
Please enter the weight anc value of item 1:
2
12
Please enter the weight anc value of item 2:
4
13
Please enter the weight anc value of item 3:
5
9
25
```

b. Implement Best-First search with Branch and Bound technique.

```
import java.io.*;
import java.util.*;

class Graph
{
    private int V;
    private LinkedList<Integer> adj[];
    private Queue<Integer> queue;

    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; i++)
        {
            adj[i] = new LinkedList<>();
        }
        queue = new LinkedList<Integer>();
    }


    void addEdge(int v,int w)
    {
        adj[v].add(w);
    }
```

```
    void BFS(int n)
    {

        boolean nodes[] = new boolean[V];
        int a = 0;

        nodes[n]=true;
        queue.add(n);

        while (queue.size() != 0)
        {
            n = queue.poll();
        System.out.print(n+" ");

            for (int i = 0; i < adj[n].size(); i++)
       {
                a = adj[n].get(i);
                if (!nodes[a])
                {
                    nodes[a] = true;
                    queue.add(a);
                }
            }
        }
    }

    public static void main(String args[])
    {
        Graph graph = new Graph(6);

        graph.addEdge(0, 1);
        graph.addEdge(0, 3);
        graph.addEdge(0, 4);
        graph.addEdge(4, 5);
        graph.addEdge(3, 5);
        graph.addEdge(1, 2);
        graph.addEdge(1, 0);
        graph.addEdge(2, 1);
        graph.addEdge(4, 1);
        graph.addEdge(3, 1);
        graph.addEdge(5, 4);
        graph.addEdge(5, 3);

        System.out.println("The Breadth First Traversal of the graph is as
```

```
follows :");

        graph.BFS(0);
    }
}
```

Result

compiled and executed in 1.174 sec(s)

```
The Breadth First Traversal of the graph is as follows :
0 1 3 4 2 5
```

c. Compare both the results and write down the summary.

Branch and bound explores the search space exhaustively by branching on variables (=test out the values of the variables). This creates several subproblems e.g. branching on a binary variable creates a problem in which the variable =0 and a problem in which it =1. You could then proceed and solve them recursively. The 'bounding' aspect of the technique consists of estimating bounds on the solutions that can be attained in the subproblem. If the subproblem can only yield bad solutions(compared to a previously found solution) you can safely skip the exploration of that part of the search space.
Best first tries to find a solution as fast as possible by looking at the most interesting part of the search space first (most interesting = estimate). It does not split the search space, only tries to reach a/the solution as fast as possible.
Both approaches try to skip the investigation of parts of the search space. Their use and efficiency depends on the problem setting. Best first requires you to specify a criterium for 'the most interesting solution to explore', which can sometimes be hard/impossible. Branch and bound is only interesting if the bound you can put on the subproblems are meaningful/not too broad.

II. Backtracking:

Assume there are 5 courses for 19 bce students. Faculty allotted for the courses are 5 in number.
C1-F1,C2-F2,C3-F3,C4-F4,C5-F5.
Prepare a timetable for the courses from Mon-Fri. Per week the number of hours to courses are 2. No back to back class can be allotted. Only 1 first hour could be given. Course1 and 2 could not be given on day3. F4 and F5 cannot handle classes on day4. No classes could be given from 2-3 slots.

CODE :

OUTPUT :