10/2/2020

Branch & Bound

0/1 Knapsack problem using Least Cost Search

Name of Student: N. I. Md. Ashafuddula
Roll: 18204016
Program: M. Sc. Engineering in CSE Program
Course: CSE-6406
Course Title: Advanced Algorithm Design and Analysis

The Complete Algorithm: –

- Cost calculation, $\boldsymbol{C} = -\sum_{i=1}^{n} \text{Pi} * \text{Xi}$ [Take with fraction]
- Upper bound, $\boldsymbol{U} = -\sum_{i=1}^{n} \text{Pi} * \text{Xi}$ [Take without fraction]
- Initialize **U = infinity**.
- Take an **empty queue, Q**
- A **dummy node** of the decision tree is created and insert or enqueue it to Q. **Cost/Profit and Weight** of dummy node be 0.
- **Do following while Q is not empty.**
  - An item from Q is created. Let the extracted item be X.
  - Calculate Cost, C and Upper bound, U of next level node from function. If the new U is lesser than previously taken Upper bound U then U will be updated by new U.
  - If the new U is greater than previously taken U node then the node will be killed & no more further exploration will happen.
  - Consider this case when next level node is not treated or considered as part of solution and add a node to queue with level as next, but weight and profit without treating or considering next level nodes.
  - If the summation of weights of all live node is greater than M or Capacity of bag then this will be treated as infeasible node & no more further exploration will happen.
- Tree traverse from start node to last live node will produce our optimal solution.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef enum { NO, YES } BOOL;

int N;
int vals[100];
int wts[100];

int cap = 0;
int mval = 0;

void getWeightAndValue (BOOL incl[4], int *weight, int *value) {
  int N = 4;
        int i, w = 0, v = 0;
        for (i = 0; i < N; ++i) {
                if (incl[i]) {
                        w += wts[i];
                        v += vals[i];
                }
        }
        *weight = w;
        *value = v;
}

void printSubset (BOOL incl[4]) {
        int i;
        int val = 0;
        int N = 4;
        printf("Included = { ");
        for (i = 0; i < N; ++i) {
                if (incl[i]) {
                        printf("%d ", wts[i]);
                        val += vals[i];
                }
        }
        printf("}; Total value = %d\n", val);
}
```

```
void findKnapsack (BOOL incl[4], int i) {
	int cwt, cval;
	int N = 4;
	getWeightAndValue(incl, &cwt, &cval);
	if (cwt <= cap) {
		if (cval > mval) {
			printSubset(incl);
			mval = cval;
		}
	}
	if (i == N || cwt >= cap) {
		return;
	}
	int x = wts[i];
	BOOL use[N], nouse[N];
	memcpy(use, incl, sizeof(use));
	memcpy(nouse, incl, sizeof(nouse));
	use[i] = YES;
	nouse[i] = NO;
	findKnapsack(use, i+1);
	findKnapsack(nouse, i+1);
}

int main(int argc, char const * argv[]) {
	printf("Enter the number of elements: ");
	scanf(" %d", &N);
	BOOL incl[N];
	int i;
	for (i = 0; i < N; ++i) {
		printf("Enter weight and value for element %d: ", i+1);
		scanf(" %d %d", &wts[i], &vals[i]);
		incl[i] = NO;
	}
	printf("Enter knapsack capacity: ");
	scanf(" %d", &cap);
	findKnapsack(incl, 0);
	return 0;
}
/**
Sample input & Output:
```

```
Enter the number of elements: 4
Enter weight and value for element 1: 2 10
Enter weight and value for element 2: 4 10
Enter weight and value for element 3: 6 12
Enter weight and value for element 4: 9 18
Enter knapsack capacity: 15
Included = { 2 }; Total value = 10
Included = { 2 4 }; Total value = 20
Included = { 2 4 6 }; Total value = 32
Included = { 2 4 9 }; Total value = 38
*/
*/
```