Philippe WS CSC 349

Lab 7

Document 1, Problem 3

*Counter Examples*

1. Capacity: W = 4

|  |  |  |
| --- | --- | --- |
| Id | Weight | Value |
| 1 | 4 | 6 |
| 2 | 2 | 4 |

1. Capacity: W = 4

|  |  |  |
| --- | --- | --- |
| Id | Weight | Value |
| 1 | 4 | 4 |
| 2 | 3 | 3 |

1. Capacity: W = 4

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Weight | Value | Ratio |
| 1 | 3 | 12 | 4 |
| 2 | 2 | 7 | 3.5 |

*Problem*

1. V(W): A function representing the maximum value of a knapsack of capacity W obtained from a set of items n, with each item i having weight wi, value vi, and infinite occurrences,
2. V(W) = max1 ≤ i ≤ n && w(i) < W { V(W-wi) + vi }, V(0) = 0.
3. The table is a 1D array containing the maximum values of knapsacks of capacity 1 to n;
4. *Pseudo Code Filling Table*  
   //items[n], the list of n items, where each item object contains its id, weight, and value.  
   //W, the maximum capacity of the knapsack we are trying to fill.

fillKnapsackTable( items[n], W ){

resultTable[W];  
for( j = 1 to W ){

sackCurrentMax = resultTable[j-1];   
possibleMax = max1 ≤ i ≤ n && w(i) < W{ items[i].value +

resultTable[j-items[i].weight] };

if(possibleMax > sackCurrentMax)  
sackCurrentMax = possibleMax;

resultTable[j] = sackCurrentMax;

}

}

1. Pseudo Code Traceback

traceBack(resultTable, items[n]){

tracebackArray = []

i = resultTable[resultTable.length-1];

while( i > 0){

while j = 0 to N

if( resultTable[i] == resultTable[i - items[j].weight]

+ items[j].value)

tracebackArray.add( items[j].item )

i = resultTable[i – items[j].weight]

break;

}

return tracebackArray;

}

1. Since iterating through the list of items is constant time, then there are a constant C number of operations for any knapsack of capacity W. Thus, the time complexity is O(C\*W) or O(n).

Document 1, Problem 4

1. C(i) gives the optimal cost of consulting business operations for a period of i months.
2. C(i) = min{ C(i-1) + M + SFi, C(i-1) + NYi }if (location == NY) ||

min{ C(i-1) + M + NYi, C(i-1) + SFi } if (location == SF)

1. A 1D array containing the optimal consulting business operations costs up to index/month i.
2. Pseudo Code Iterative Table Fill  
   //Inputs: SF[i] the set of costs each month for SF. NY[i] the set of costs each month for //NY[i]. M, the cost to move between the two cities.  
   minBusinessCost( SF[i], NY[i], M){

result[][] = [2][i];

result[0] = 0; result[1] = min{ SF[1], NY[1] };

bool inSF = SF[1] <= NY[1] ? inSF = true : inSF = false;

cities[0] = null;

for (n = 1 to i){

if( inSF ){  
if( result[n-1] + M + NY[n] < result[n-1] + SF[n] ){  
 result[0][n] = result[0][i-1] + M + NY[n];}else{  
 result[0][n] = result[0][n-1] + SF[n];

inSF = false;  
}

}else{  
if( result[n-1] + M + SF[n] < result[n-1] + NY[n] ){  
 result[0][n] = result[0][n-1] + M + SF[n];  
 inSF = true;  
}else{  
 result[0][n] = result[0][n-1] + NY[n];  
}

}

result[1][n] = inSF;

}

return result;  
}

1. Pseudo Traceback  
   traceback( result[2][i] ) {

//Since in the second dimension we stored the true or false value of whether or not //we were in city 1, simply follow that and prescribe the values as necessary to //some sort of string array.

string[] cities = [i];  
for( n = 1 to i ){

cities[n-1] = result[1][i] ? “SF” : “NY”;  
}

return cites;

}

1. There are a constant number of comparisons per iteration of filling out the table, thus, for any number of i months, there are C comparisons. Thus C\*i, and complexity of

Document 2, Problem 6

1. Opt(mi) is a function representing the maximum profit that can be obtained from the set of valid restaurants placed at further mi.
2. Opt(mi) = max{ Opt( Cl(mi-k) ) + pi, Opt(mi-1) }, Opt(0) = 0; Cl(x) is the closest valid restaurant location contained in the set mn to the value distance x.
3. The table is a one-dimensional array contain the maximum value possible for i restaurants, where i is the count of restaurant locations and index in the array.
4. Pseudo code Table Filling  
   optimalFireStonePlacement( m[n], p[n], int k ){

Cl[n]; //An array containing the closest compatible previous restaurant of any //placement i ≤ n.

resultTable[n];

resultTable[0] = 0;

for(i = 1 to n){

resultTable[i] = max{ resultTable[ Cl[i] ] + p[i], resultTable[i-1];

}

}

1. Pseudo Code Traceback

traceback(Opt[n], p[j]){

locations = [];

largestRevenue = Opt[n];

for( j = n to 1){

if( Opt[j] > Opt[j-1] && Opt[j] == largestRevenue){

locations.add( j );

largestRevenue -= p[j];

}

}

return locations;

}

Document 2, Problem 7

1. Calendar

   Description automatically generated with medium confidence
2. Opt(k, i) = Opt(k-1, i) + maxx ≤ |Cmp(i)|{ S(Cmp(i)) },   
   Where S(x) is the is set of scores derived from the set of all column patterns. Cmp(x) is the list of all column patterns compatible with pattern i.

Opt(0, x) = -∞, for 1 ≤ x ≤ 8.

Pseudo Code Table Fill

optimalPebblePlacement( Board[4][n] ){

//An array with 8 rows 2 columns, containing all possible configurations of //columns. Either the row number of the pebble placement or -1 if not placed.

Config[8][2];

//An array of lists containing all compatible patterns for any pattern i. Could also //be represented as an 8x8 compatibility matrix.

Cmp[8][~];

//Initialize the result table.

resultTable[7][n];

for( i = 0 to 7 )

resultTable[i][0] = 0;

//Create table

for ( col = 1 to n ){

for( p = 0 to 7 ){

max = 0;

val1 = Board( Config[p][0], Config[p][1] );

for( i = 0 to Cmp[p].length ){

val2 = resultTable[i][col-1];

if(val1 + val2 > max) max = val1 + val2;

}

resultTable[p][col] = max;

}

}

return resultTable;

}

Pseudo Code Traceback

traceback( resultTable[p][n], Cmp[p] ){

patternChoices[n];

maxVal = maxi=0-7 { resultTable[i][n-1] };

maxIndex = resultTable.getIndex(Val);

patternChoice[n-1] = maxIndex;

for( i = n-1 to 1 ){

for( p in Cmp(maxIndex) ){  
 if(maxValue < resultTable[p][i] ) {

maxIndex = p;

maxValue = resultTable[p][i];

}

}

patternChoices[i-1] = p;

}

}