Daniel Campos

November 15th,2012

Algorithms Section 5

HW 6

1. Problem 6.8 (DG)(doing)

int LCS(string X,string Y)

{

if (Y.length() > X.length())//the X must be longer

swap(X,Y);

int m = X.length(),n=Y.length();//size calues

vector< vector<int> > c(2, vector<int>(n+1,0));

int i,j;

for (i=1;i<=m;i++)

{

for (j=1;j<=n;j++)

{

if (X[i-1]==Y[j-1])

c[1][j]=c[0][j-1]+1;

else

c[1][j]=max(c[1][j-1],c[0][j]);

}

for (j=1;j<=n;j++)

c[0][j]=c[1][j];

}

return (c[1][n]);

}

The run time is O(mn) since it will loop the external loop for n times and the internal loop based on the x size. for m times based on the y size.

1. Problem 6.17 (DG) Hint similar to knapsack problem.(doing)

Input: Denominations of coins x1,...,xn,Value V

Output: true if making change is possible false otherwise

Declare an array D of size V + 1

D[0] = true

for i = 1 to V

D[i] = false

for v = 1 to V :

for j = 1 to n:

if xj ≤ v://coin must be smaller than value

D[v] = D[v] ∨ D[v − xj]

else:

D[v] = false

return D[V ]

Complexity Analysis: There are two for loops one loop runs V times while the other

loops run n times. Therefore the complexity of the algorithm is O(nv)

1. Problem 6.18 (DG)

Recurrence:

M(n,V)=M(n-1,V-x(n)) ∨ M(n-1,V) where 1 would be able to make change and 0 would be not able to make change. In the former case, the algorithm will output 1 if change can be made for amount v-x(n) using the first n-1 denominations. In the latter case the algorithm should output positive if change can be made using the first n-1, if both are true than the output will be true. Complexity is O(nv) since you have built a table of n rows and V colums.

Declare an array D of size V + 1

D[0] = true

for i = 1 to V

D[i] = false

for v = 1 to V :

for j = 1 to n-1:

if xj ≤ v://coin must be smaller than value

D[v] = D(V-x(n)) ∨D(V);

else:

D[v] = false

return D[V ]

Non dynamic version is under// ignore just for me

bool can\_make\_change\_using\_coins(vector<int>coins ,int v)

{

for(int i=0;i<coins.size();i++)//runs through the coins

{

if(v>=coins[i])//as long as coin is removable it will

{

v=v-coins[i];//removes one coin

}

}

if(v==0)//the amount is returnable

{

return true;

}

return false;

}

1. Problem 6.19 (DG)

Recurrence: If (k > 1, M(k, V ) = max{i:x(i)<=V} )

M(k – 1,V – x(i))

else {M(k, V ) = (x1 == V ) ∨ (x2 ==V ) ∨ …∨ (x(n) == V )}

Where M(k, v) = 1 if change can be made

Complexity: O(nV K) as you have to build a table of K rows and V columns, but the initial recurrence requires you to examine n possibilities

Non dynamic version is under// ignore just for me

bool can\_make\_change\_using\_k\_coins(vector<int>coins, int k, int v)

{

int coins\_used=0;//amount of used coins

for(int i=0;i<coins.size();i++)

{

if(v>=coins[i])//as long as the the amount is larger than the coin size

{

int temp=v/coins[i];//will give you amount of coins that can be removed

v=v%coins[i];//modulus to remove the amount coins from v

coins\_used+=temp;//ups the amount of coins used

}

}

if(coins\_used<=k&&v==0)//if the amount of coins used > k and v = 0 so true

{

return true;

}

return false;

}

1. Problem 15.1-3 (page 370)

Function max\_profit

Input:price pi,…pn rods,cost of cut c

Output:max profit

r = [0] \* (n + 1)

for k in range(1, n + 1):

{ ans = p[k]

for i range(1, k):

{ ans = max(ans, p[i] + r[k - i]-c)//c is cost of a cut

r[k] = ans

}

}

return r[k];