Gonzalo Gomez

Israel Gomez

David Hunter

4/1/2013

Dynamic Programming: Finding an Optimal Schedule to Maximize Grade

Theory

Approach:

The problem of finding the optimal schedule for N projects, given their individual function for grade returned for a certain amount of hours, H, is solved easiest by breaking down the problem into smaller recursive problems. The approach that we took is similar to the problem solved in class of dividing the work of scanning pages evenly across N people. This is done by partitioning H hours into N regions, using N-1 partitions. We will use the recursive idea that the kth partition starts right after we place the (k-1)st divider. We place the divider between the ith and (i+1)th hour for some i. By changing the positions of the dividers during each recursive call, this ensures that we will find each possible combination to divide the H hours among the N projects in order to maximize grade. In order to make the program more dynamic, for each max grade path that is calculated, that value will be stored. For each recursive call, the values for each H and N will be checked, and if the corresponding grade for those values has already been calculated, then the max grade will be returned without having to be recalculated, or without doing any more recursive calls, thus greatly speeding up the run time of the program.

Pseudocode:

Cell Class {

Integer maxGrade;

Integer parentH;

Integer parentN;

}

Main Class:

//H is an integer representing the number of hours

//N is an integer representing the number of projects

//mag is a 2 dimensional array that stores objects of type Cell and has dimensions H+1 by N+1

mag[][] = mag[H+1][N+1]

Integer maxAverageGrade(H, N) {

Integer max = 0;

if mag[H][N] has already been computed then return mag[H][N].maxGrade

if N = 1 then maxGrade of the cell contained at mag[H][N] = f1(H) and return that value

if H = 0 then

Cell cell

cell.maxGrade = 0

mag[H][N] = cell

return 0

for (int i = 0; i <= H; ++i) {

Cell tempCell = new Cell

tempCell.maxGrade = maxAverageGrade(H-i, N-1) + fN(i)

tempCell.maxGrade = tempCell.maxGrade/N

if tempCell.maxGrade > max {

max = tempCell.maxGrade

tempCell.parentH = H - i

tempCell.parentN = N – 1

mag[H][N] = tempCell

}

}

return mag[H][N].maxGrade

}

Traceback Algorithm:

pathMaxGrade(H, N) {

if N = 1 print “Project “ + N + “ takes “ + H + “ hours” and return

if H = 0

print “Project " + N + " takes " + H + " hours"

pathMaxGrade(H, N-1)

return

else

“Project “ + N + “ takes “ + H – mag[H][N].parentH + “ hours” and return

pathMaxGrade(mag[H][N].parentH, mag[H][N].parentN)

}

Complexity:

/////////////doooooooooooooooooooo///////////////////////////////

Implementation Example:

/////////////////////////////DOOOOOOOOO/////////////////////////

Group Effort:

The agreed upon group effort percentage for each member is 33 1/3%. We agreed that each member of the group did equal amount of work and participated equally, and therefore should receive equal grade. We met as a group each time we worked on it, so every member worked on each part of the project.