# CMPS 102 Homework 4: Dynamic Programming

Benjamin Gordon / bcgordon@ucsc.edu

6/5/2014

## 1 Air Travel

Let us begin by defining $Cost_{p,n}$ to be the cost of the walkway on side $p$ at position $n$. Let us also define $\{L_n, R_n\}$ to be the shortest possible paths to walkways $L$ and $R$ at position $n$. We can recursively define $\{L_n, R_n\} = \{min\left[\left(L_{n-1} + Cost_{l,n}\right), \left(R_{n-1} + k + Cost_{l,n}\right)\right], min\left[\left(R - 1 + cost_{r,n}\right), \left(L_{n-1} + k + Cost_{r,n}\right)\right]\}$.

This recursion holds because when we are computing the optimal path including a given walkway, we are using the already-computed optimal path for all the previous walkways. So when we start from the beginning, we will calculate the optimal path up to and including the given walkway, for every walkway.

For a data structure, we will use two arrays. One array will hold the optimal path up to and including a given walkway for all the left side walkways, and the other will hold the same values for the right side. We will fill each array in sequential order from 1 to $n$.

The size of each array is $n$, and for each spot, we do a couple of arithmetic problems based on the previous two spots in the array and do a comparison. Therefore, each spot in the array is filled in constant time, and this algorithm runs in $O\left(n\right)$ time.

## 2 Sightseeing

Our recursive property is: $OPT\left(n\right) = min\left[\left(OPT\left(n-1\right) + \left(r \times d_n\right)\right), \left(b + OPT\left(n-5\right)\right)\right]$. This recursion holds because at each point, we have two options - we can take a taxi to the next destination, or we can take a limo to the point from five destinations back. Since $OPT\left(n-1\right)$ and $OPT\left(n-5\right)$ are guaranteed to be correct, we know that we will choose the smallest cost path for $OPT\left(n\right)$ .

For a data structure, we will use a linked list. Each node will contain two things: the $OPT\left(index\right)$ value, and an array with length equal to the index. Each spot in this index will contain either a $T$ or an $L$, representing whether a taxi or limo was used for this distance. That way, we can return an actual schedule for the $OPT$ value.

Since we have a linked list of length $n$, and for each computation of $OPT$, we simply do two lookups, some arithmetic, and a comparison, this algorithm runs in $O(n)$ time.

# 3 A Museum

To solve this problem, we need to maximize facts gained per minute over our entire time at the museum. This means we need to know the number of facts we will gain per minute for every possible amount of time spent at every exhibit. Let us define $G_i(x)$ to be the number of facts *per minute* we will have gained if we spend $x$ minutes at exhibit $i$. We can now do some preprocessing, and make an $n \times m$ matrix, call it $G$, of all the $G_i(x)$ values for every exhibit. This will take $mn$ time, as it's just one calculation per element of the matrix. While we compute $G$, we will also pay attention to the point at which $f_i(x) = s$, where i is some exhibit $i \leq n$, and set all points $G_i(m)$ where $m > x$ equal to 0, so the algorithm will not select them.

When we run $OPT(m)$, it will find the maximum reachable $G_i(x)$ value in the museum, and spend enough minutes to take that value. Call this number of minutes $y$. Then it will denote all values in $G$ which are associated with that exhibit, and for which $m \leq y$, as taken, and call $OPT(m-y)$.

Thus we have $OPT(m) = max_{1 \leq y \leq m}(max_{1 \leq k \leq n}(G_k(y))) + OPT(m-y)$

This recursion holds because the key to solving this problem is maximizing facts per minute. This solution doesn't just consider the maximum amount of facts for each minute; it considers the amount of facts gained for all minutes spent at any given exhibit, and always takes enough time to get the greatest possible return given the amount of time available.

By evaluating the taken values in $G$ once the algorithm has completed, we can find the optimal schedule. The running time for this algorithm is $O(m^2 n)$, because at each stage it scans through at most $mn$ elements of $G$, and does this at most $m$ times.

# 4 Pesky Children

Define $S$ to be a set of $j$ grades less than $n$, $m$ to be the number of students in each grade, $A_i$ to be the number of $A$ votes in grade $i$, and $B_i$ to be the number of $B$ votes in grade $i$. Let $X_p$ and $Y_{j-p}$ be the number of $A$ votes in bus 1 and 2 when they have $p$ and $j-p$ grades respectively. OPT will return true if it is possible to have an assignment that meets the requirement, and false if it does not.

There are two cases in which $OPT(j, p, X_p, Y_{j-p})$ will return true. Either:

- $OPT(j-1, p-1, X_{p-1}, Y_{j-p})$ is true, and $(p-1) \geq \left\lfloor \frac{(j-1)}{2} \right\rfloor$, and $X_{p-1} + A_j \geq (p-1) \times m/4$ (this is the case in which the last grade added is added to bus 1), OR

- $OPT(j-1, p, X_p, Y_{j-p-1})$ is true, and $(j-p-1) \geq \left\lfloor \frac{(j-1)}{2} \right\rfloor$, and $Y_{j-p-1} + A_j \geq (j-p-1) \times m/4$ (the case where the last grade added is added to bus 2)

We can say that each bus has a majority vote for restaurant $A$ after $j$ grades if there is a majority vote for $A$ in each bus after the $j$th grade is added. If the last grade added was added to bus 1, and bus 2 already has a majority vote, then we have an overall majority vote for $A$ if adding a grade to bus 1 results in a majority vote there as well. The same goes for bus 2; if there was already a majority vote in bus 1, and adding a grade to bus 2 results in a majority vote for $A$ there, then we have an overall majority vote for $A$.

At each computation of OPT, we should save which grade is going into each bus. We will keep an array of size $n/2$ for each bus, and fill each index in either array with a grade number. Since there are $n$ array spots to fill in total, and each time we calculate OPT, we need to reconsider how every grade is placed in either bus, the algorithm runs in $n^3$ time.

# 5 Typesetting

I noticed immediately that this problem was similar to the least squares problem discussed in class. Let us define line $L_i$ to be the line starting at word $i$. Each line must begin at some word $w_i$ and end at some word $w_j$; therefore, assuming that the algorithm has computed every prior line optimally, we only need to find the length of this line with minimal slack. We can do this by adding on words $w_i + 1$, $w_i + 2 \ldots$ to line $L_i$ one by one until we go over the maximum length $L$. Let us also define $S_{i,j}$ to be the slack of the line from $i$ to $j$. Our algorithm therefore looks like this: $OPT(w_n) = \min_{1 \leq j \leq n} \left[ (S_{j,n})^2 + OPT(w_{j-1}) \right]$

Computing OPT takes n time, because we consider every word in the list once. Since we compute OPT $n$ times, our recurrence is $O(n^2)$.