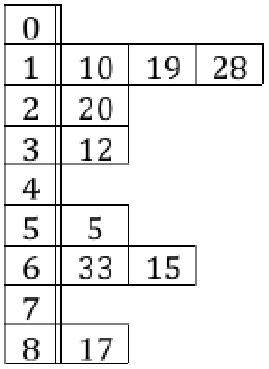
MCS 401 Spring 2020
Homework 4

April 9, 2020

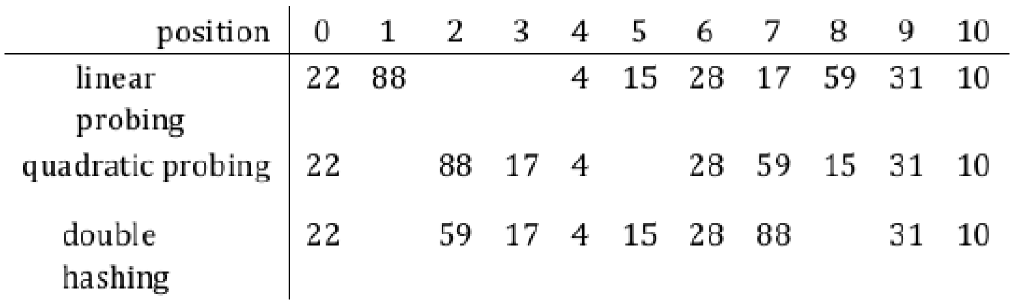
1. Problem 1 [Exercise 11-2-2, page 261]: The following table shows how to put 5, 28, 19, 15, 20, 33, 12,17,10 into a hash table with $h(k) = k \mod 9$.



0			
1	10	19	28
2	20		
3	12		
4			
5	5		
6	33	15	
7			
8	17		

2. Problem 2 [Problem 11-4-1, page 277]:

   The table below shows how to insert 10,22,31,4,15,28,17,88,59 into a hash table with $m = 11$ slots using the various techniques specified in the problem:



position	0	1	2	3	4	5	6	7	8	9	10
linear probing	22	88			4	15	28	17	59	31	10
quadratic probing	22		88	17	4		28	59	15	31	10
double hashing	22		59	17	4	15	28	88		31	10

3. Problem 3 [Exercise 15-4-4, page 396]:

Let the row sequence and column sequence have m and n entries respectively. Without loss of generality we can assume that $n \leq m$, since if otherwise we can just switch the two sequences.

When computing a particular row of the c table, no rows before the previous row are needed. Thus only two rows, and hence $2 \cdot n$ entries, are needed to be kept in memory at a time. (Note: Each row of c actually has $m + 1$ entries, however we don't need to store the column of 0's but make the program "know" that those entries are 0.) We can thus do away with the c table as follows:

- Use two arrays of length n, previous-row and current-row, to hold the appropriate rows of c.
- Initialize previous-row to all 0 and compute current-row from left to right.
- When current-row is filled, if there are still more rows to compute, copy currentrow into previous-row and compute the new current-row.

Actually only a little more than one row's worth of c—$n + 1$ entries—are needed during the computation. The only entries needed in the table when it is time to compute $c[i,j]$ are

- $c[i,k]$ for $k \leq j-1$ (i.e., earlier entries in the current row, which will be needed to compute the next row);
- and $c[i-1,k]$ for $k \geq j-1$ (i.e., entries in the previous row that are still needed to compute the rest of the current row).

This is one entry for each k from 1 to n except that there are two entries with $k = j-1$, hence the additional entry needed besides the one row's worth of entries. We can thus do away with the c table as follows:

- Use an array a of length $n + 1$ to hold the appropriate entries of c. At the time $c[i,j]$ is to be computed, a will hold the following entries:
  - $a[k] = c[i,k]$ for $1 \leq k < j - 1$ (i.e., earlier entries in the current "row"),
  - $a[k] = c[i - 1,k]$ for $k \geq j - 1$ (i.e., entries in the previous "row"),
  - $a[0] = c[i,j - 1]$ (i.e., the previous entry computed, which couldn't be put into the "right" place in a without erasing the still-needed $c[i - 1,j - 1]$).
- Initialize a to all 0 and compute the entries from left to right.
  - Note that the 3 values needed to compute $c[i,j]$ for $j > 1$ are in $a[0] = c[i,j - 1]$, $a[j - 1] = c[i - 1,j - 1]$, and $a[j] = c[i - 1,j]$.
  - When $c[i,j]$ has been computed, move $a[0]$ ($c[i,j - 1]$) to its "correct" place, $a[j - 1]$, and put $c[i,j]$ in $a[0]$.

4. Problem 4 [Exercise 16-1-5, page 422]:

   Let $\{a_1, \ldots, a_n\}$ be a set of activities and let us denote a value of activity a_j as v_j. Assume the activities are sorted in a way that for $i < j$ the finishing time $f_i \leq f_j$. We set S_{ij} to be a set of all activities which start after a_i is finished and before a_j is started. Note that this definition implies that $S[ii+1] = \emptyset$ as well as $S[ji] = \emptyset$ for $j \geq i$. Let $c[ij]$ denote a a value of the optimal solution restricted to S_{ij}. Then we can fill the table as follows:

$$c[ij] = \begin{cases} 0, & S_{ij} = \emptyset \\ \max_{a_k \in S_{ij}} \{c[ik] + v_k + c[kj]\}, & S_{ij} \neq \emptyset \end{cases}$$

   We create $n+1 \times n+1$ table indexed by $0, \ldots, n$ and initialize $c[ij] = 0$ for $i \geq j, i, j \in [0, n+1]$ and $c[i, i+1] = 0$ for $i \in [o, n]$ as base cases. We output $c[0, n+1]$ as an optimal solution.

   Sorting the activities takes polynomial time. Each S_{ij} has $|S_{ij}| \leq n$ and there are at most $\binom{n+1}{2} = O(n^2)$. Therefore, constructing all $S_{i,j}$ takes polynomial time. Finding k that maximizes the value $c[ij]$ also takes polynomial time. Hence, computing each $c[ij]$ takes polynomial time as well. The algorithm fills each entry of $O(n^2)$ table exactly once. Thus, the algorithm runs in polynomial time.

   The optimal substructure of the problem is proved on the page 416 of the book.

5. Problem 5 [Extra problem]:

   (a) Let T denote the dynamic programming table and v denote an $m \times n$ matrix. To find the maximum cost path, we fill the table T as follows:

$$T[1][1] = 0$$

$$T[i][j] = \max\{T[i-1][j] + |v[i-1][j] - v[i][j]|, T[i][j-1] + |v[i][j] - v[i][j-1]|\}$$

   Note that this is different from first taking the max subpath and then adding the corresponding cost - it is possible that the max subpath plus corresponding cost is actually less than another subpath plus corresponding cost, if the cost (absolute difference) is large.

   (b) The max path is $9 \to 13 \to 3 \to 10 \to 1 \to 4$, giving a cost of 33.