

Homework 4 (due before class, Monday, October 29, 2001)

1 Augmented Search Trees [Chapter 14 of CLRS]

1. In this problem we consider data structures for maintaining a matrix M . We want to support the following operations:

- $Init(M)$: create an empty matrix (zeros on all positions)
 - $Lookup(M, i, j)$: return the value at index (i, j) ($i, j \geq 0$)
 - $Update(M, i, j, e)$: change the value at index (i, j) to e ($i, j \geq 0$)
 - $Transpose(M)$: transpose the matrix (that is, element at index (i, j) becomes element at index (j, i))
 - $Add(M)$: return the sum of the elements in M
- (a) Assume first that we are working on a $n \times n$ matrix. Describe a data structure such that $Init$ runs in $O(n^2)$ time, and all other operations run in $O(1)$ time.
- (b) Assume now that the matrix is of arbitrary size. Describe a data structure such that $Init$ runs in $O(1)$ time, $Lookup$ and $Update$ in $O(\lg k)$ time, and $Transpose$ and Add in $O(1)$ time, where k is the number of non-zero entries in the matrix.

2 Hashing, Direct Addressing [Chapter 11 in CLRS]

2. [CLRS 11.1-4] We wish to implement a dictionary by using direct addressing on a *huge* array. At the start, the array entries may contain garbage, and initializing the entire array is impractical because of its size. Describe a scheme for implementing a direct-address dictionary on a huge array. Each stored object should use $O(1)$ space; the operations $Search$, $Insert$, and $Delete$ should take $O(1)$ time each; and the initialization of the data structure should take $O(1)$ time. (Hint: Use an additional stack, whose size is the number of keys actually stored in the dictionary, to help determine whether a given entry in the huge array is valid or not.)

3. [CLRS 11-2] Suppose that we have a hash table with n slots, with collisions resolved by chaining, and suppose that n keys are inserted into the table. Each key is equally likely to be hashed to each slot. Let M be the maximum number of keys in any slot after all the keys have been inserted. Your mission is to prove an $O(\lg n / \lg \lg n)$ upper bound on $E[M]$, the expected value of M .

- (a) Argue that the probability Q_k that exactly k keys hash to a particular slot is given by

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k}$$

- (b) Let P_k be the probability that $M = k$, that is, the probability that the slot containing the most keys contains k keys. Show that $P_k \leq nQ_k$.
- (c) Use Stirling's approximation, use (3.19) and (3.11) to show that $Q_k < e^k/k^k$.
- (d) Show that there exists a constant $c > 1$ such that $Q_{k_0} < 1/n^3$ for $k_0 = c \lg n / \lg \lg n$. Conclude that $P_k < 1/n^2$ for $k \geq k_0 = c \lg n / \lg \lg n$.
- (e) Argue that

$$E[M] \leq \Pr \left\{ M > \frac{c \lg n}{\lg \lg n} \right\} \cdot n + \Pr \left\{ M \leq \frac{c \lg n}{\lg \lg n} \right\} \cdot \frac{c \lg n}{\lg \lg n}$$

Conclude that $E[M] = O(\lg n / \lg \lg n)$.

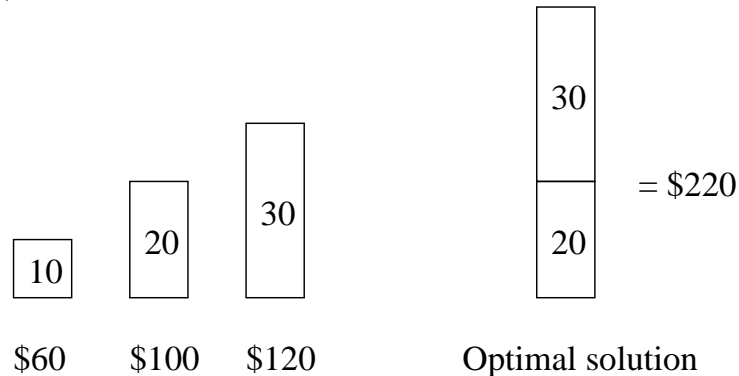
3 Dynamic programming

4. Given two sequences X and Y let $C(X, Y)$ denote the number of times that X appears as a subsequence of Y . (For example, the sequence 'ab' appears 4 times as a subsequence of 'adabcb'.) Let X_i denote the first i characters of the string X and let $X[i]$ denote the i th character (similarly for Y).

- (a) Write a recurrence for $C(X_i, Y_j)$.
- (b) Say X has length m and Y has length n . Let $C[i, j]$ be a 2-dimensional $m \times n$ matrix initialized to all zeros. Describe briefly (or write pseudocode) how to convert your solution to (a) into a dynamic programming algorithm to compute $C(X, Y)$. You may use either a bottom-up or a top-down approach.
- (c) What is the running time of your algorithm as a function of m and n ?

5. [CLRS 16.2-2] In this problem we consider the 0-1 KNAPSACK PROBLEM: Given n items, with item i being worth $v[i]$ dollars and having weight $w[i]$ pounds, fill a knapsack of capacity m pounds with the maximal possible value.

Example: Given a knapsack of capacity 50, the maximal value obtainable with three items of value \$60, \$100, and \$120 and weights 10, 20, and 30, respectively, is \$220.



The algorithm Knapsack(i,j) below returns the maximal value obtainable when filling a knapsack of capacity j using items among items 1 through i (Knapsack(n,m) solves our problem). The algorithm works by recursively computing the best solution obtainable *with* the last item and the best solution obtainable *without* the last item, and returning the best of them.

```
Knapsack(i,j)

  IF w[i] <= j THEN
    with = v[i] + Knapsack(i-1, j-w[i])
  ELSE
    with = 0
  FI
  without = Knapsack(i-1,j)
  RETURN max{with, without}
```

END

(a) Show that the running time T of Knapsack(n, m) is exponential in n or m . (*Hint*: Look at the case where $w[i] = 1$ for all $1 \leq i \leq n$ and show that $T(n, m) = \Omega(2^{\min(m,n)})$).

(b) Describe an $O(n \cdot m)$ algorithm for computing the value of the optimal solution.

6. A game-board consists of n columns. Each column consists of two numbers, which we refer to as the top number and the bottom number for that column. The top number can be any positive integer, while the bottom number is either 1, 2, or 3.

The object of the game is to travel (by a series of moves) from the first column all the way to the right of the n th column (and off the board). The top number of a column is the cost of visiting that column. The bottom number in a column is the maximal number of columns that the player is allowed to jump to the right in the next move. The cost of a game is the sum of the costs of the visited columns until the player moves off the board.

Let the board be represented in a two-dimensional array $B[n,2]$. The following recursive procedure (when called with argument 1) computes the cost of the cheapest game:

Cheapest(i)

```
  IF i>n THEN return 0
  x=B[i,1]+Cheapest(i+1)
  y=B[i,1]+Cheapest(i+2)
  z=B[i,1]+Cheapest(i+3)
  IF B[i,2]=1 THEN return x
  IF B[i,2]=2 THEN return min{x,y}
  IF B[i,2]=3 THEN return min{x,y,z}
```

END Cheapest

- (a) Analyze the asymptotic running time of the procedure.
- (b) Describe and analyze a more efficient algorithm for finding the cheapest game.