Homework #2
Introduction to Algorithms/Algorithms 1
600.363/463
Spring 2015

Due on: Tuesday, February 10th, 5pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: On blackboard, under student assessment
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

February 11, 2015

1 Problem 1 (20 points)

Give an efficient algorithm to determine whether two sets of integers (both of size n) are disjoint. Analyze the complexity of your algorithm in terms of n.

Algorithm:

1. Put two sets (distinct elements) into two arrays. Sort the two arrays with any sorting algorithms (e.g. Merge Sort)

2. Scan through every element in the first sorted set ,and make use of the binary search to search the element in the second set. If found, return false.

3. If the algorithm iterated all the elements in the first set without returning, return true, i.e. two sets are disjoint.

Analysis: Since merge sort needs $O(n \log n)$ time to sort the two sets and each binary search takes $O(\log n)$ time, the algorithm provided above needs $O(n \log n) + O(n \log n) = O(n \log n)$ time.

2 Problem 2 (20 points)

Given two sorted arrays A, B, give a linear time algorithm that finds two entries i, j such that $|A[i] - B[j]|$ is minimized. Prove the correctness of your algorithm and analyze the running time.

Algorithm: Assume A and B are sorted in ascending order. A has m elements and B has n elements.

1. Use the merge process of the merge sort to combine A and B into a sorted array C.

2. Iterate through array C and record the minimum difference between the two consecutive elements if the elements are not from the same array A or B. Store the minimum difference and their indices in A and B.

3. Return the minimum difference min and corresponding indices i and j.

We provide pseudocode here:
$Algorithm(A, B, m, n)$

Initialize two empty arrays $L[0 \dots m]$ and $R[0 \dots n]$
Initialize an empty 2-D array $C[0 \dots m+n-1][0 \dots 2]$
for $i = 0$ **to** $m - 1$ **do**
    $L[i] = A[i]$
end for
for $j = 0$ **to** $n - 1$ **do**
    $R[j] = B[j]$
end for
$L[m] = \infty$, $R[m] = \infty$
$i = 0$, $j = 0$
for $k = 0$ **to** $m + n - 1$ **do**
    if $L[i] \leq R[j]$ **then**
        $C[k][0] = L[i]$, $C[k][1] = 0$, $C[k][2] = i$
        $i = i + 1$
    else
        $C[k][0] = R[j]$, $C[k][1] = 1$, $C[k][2] = j$
        $j = j + 1$
    end if
end for
$min = \infty$, $a = \infty$, $b = \infty$
for $l = 0$ **to** $m + n - 2$ **do**

22: **if** $|C[l][0] - C[l+1][0]| < min$ and $C[l][1] \neq C[l+1][1]$ **then**
23: $\quad min = |C[l][0] - C[l+1][0]|$
24: $\quad$ **if** $C[l][1] = 0$ **then**
25: $\quad\quad a = C[l][2], b = C[l+1][2]$
26: $\quad$ **else**
27: $\quad\quad a = C[l+1][2], b = C[l][2]$
28: $\quad$ **end if**
29: **end if**
30: **end for**
31: **return** min, a, b

Proof of Correctness:

- First, to prove the correctness of merge process in step 1, we can use a loop invariant (refer to CLRS section 2.3.1).

- Lines 1-19 correctly return a sorted array C by merging A and B. Then we try to show the correctness of lines 20-31, which perform $m + n - 1$ steps by maintaining the following loop invariant:

  At the start of each iteration of the **for** loop of lines 21-30, min stores the minimum difference between subarrays $A[0 \dots m_0]$ and $B[0 \dots n_0]$ for some $0 \leq m_0 \leq m - 1$ and $0 \leq n_0 \leq n - 1$ and $C[0 \dots l][0] = A[0 \dots m_0] \cup B[0 \dots n_0]$. Also, the corresponding indices a, b are recorded.

  We need to show that this loop invariant holds prior to the first iteration of the for loop of lines 21-30, that each iteration of the loop maintains the invariant, and that the invariant provides a useful property to show correctness when the loop terminates.

- **Initialization:** Prior to the first iteration of the loop, the minimum difference min and indices a, b are all ∞. Since first loop iteration has not been executed, subarray $C[0, l]$ has only one element. It is true that $A[0 \dots m_0]$ is empty or $B[0 \dots n_0]$ is empty. Thus, the min is the minimum difference and a, b are corresponding indices.

- **Maintenance:** To see that each iteration maintains the loop invariant, let us first consider if $|C[l][0] - C[l+1][0]| < min$ and $C[l][1] \neq C[l+1][1]$, why min indeed stores the minimum value of the difference between $A[0 \dots m_0]$ and $B[0 \dots n_0]$. This is because in a sorted array, the minimum difference only comes from two consecutive elements, i.e. $C[l][0]$ and $C[l+1][0]$. By obtaining $C[l][1]$ and $C[l+1][1]$, we know that the origins(A or B) of the two elements in C. If $|C[l][0] - C[l+1][0]|$ is not less than min, the min from previous iteration is kept. So when incrementing l, the loop invariant still holds.

- **Termination:** At termination, $l = m+n-1$. By the loop invariant, the pair of elements with minimum difference from $C[0 \ldots m+n-1]$ are stored as entires a, b. Since $C[0 \ldots l][0] = A[0 \ldots m_0] \cup B[0 \ldots n_0]$ and the original two arrays are $A[0 \ldots m-1]$ and $B[0 \ldots n-1]$, a, b are the indices of elements from A and B, respectively, and min is the minimum value recorded till the end of array C.

Running Time Step 1 takes $O(m+n)$ to merge arrays A and B and step 2 also needs $O(m+n)$ to get a pair of $A[i]$ and $B[j]$ with minimum difference.