OHJ-2016 Utilization of Data Structures

Class Exam
Friday 15 May 2009

Imed Hammouda

Make sure you read the questions carefully before giving your answer. Put your name and student number on each answer sheet.
This exam consists of **3** pages / **5** exercises. The maximum amount of points is 30. Each exercise is worth 6 points.
Written material, mobile phones and calculators are NOT allowed in the exam.
Good luck!

Exercise 1

For each of the following ideas, give a brief explanation (not to exceed 3 lines) and examples of problems and/or algorithms. (1p/point)

(a) divide and conquer
(b) breadth-first search
(c) balanced data structures
(d) use of randomness in algorithms
(e) greedy algorithms
(f) NP completeness

Exercise 2

(a) True or false? (0.25p/point)
1. If the efficiency of the algorithm is in $\Omega(\lg n)$, it is for sure also $\Theta(\lg n)$.
2. If the efficiency of the algorithm is in $O(\lg n)$, it is for sure also $\Theta(\lg n)$.
3. If the efficiency of the algorithm is in $O(n^2)$, it is for sure also in $O(n)$.
4. If the efficiency of the algorithm is in $\Omega(n^2)$, it is for sure also in $\Omega(n)$.
5. If the efficiency of the algorithm is $\Theta(n\lg n)$, it is for sure also in $\Omega(n\lg n)$.
6. If the efficiency of the algorithm is $\Theta(n\lg n)$, it is for sure also in $O(n\lg n)$.
7. If the efficiency of the algorithm is $\Theta(n)$, it is for sure also in $\Omega(\lg n)$.
8. If the efficiency of the algorithm is $\Theta(n)$, it is for sure also in $O(\lg n)$.
10. All priority queue operations can be implemented in $O(\lg n)$ time.
11. All hash table operations can be implemented in $\Theta(1)$ time.
11. Searching for an element in a singly linked list is in $\Omega(1)$.
12. The worst case efficiency of Quicksort is $O(n\lg n)$.

(b) For the following code fragment, use summations to calculate the exact number of times the inner for loop executes, give a tight upper bound (Big-Oh analysis) of the running time. (1.5p)

```
for (int i = 0; i < n*n; i++)
      for (int j = 0; j < i; j++)
            sum++;
```

(c) Use the definition of Θ-notation to show that
$3n^2 + 5n - 20 = \Theta(n^2)$ (1.5p)

Exercise 3

Consider an abstract datatype for an integer set with the following operations:

- *insert(s, i)* returns a copy of set s with integer i inserted, but raises exception Overflow if not enough memory is left for this insertion
- *member (s, i)* returns true if integer i is in set s, and returns false otherwise
- *remove(s, i)* returns a copy of set s with integer i removed

- *findMax (s)* returns the maximum element in non-empty set s
- *extractMax (s)* returns a copy of non-empty set s with its maximum element removed
- *closest(s, i)* returns i if integer i is in set s, and returns an element of s that is numerically closest to i otherwise, assuming that s has at least two elements

Consider the following data structures for implementing this abstract datatype:

1. Unsorted array	2. Sorted array
3. Hash table	4. Balanced binary search tree
5. List	6. Heap

Consider only arrays that must always remain filled from the left, and where we maintain the index of the rightmost actually used cell. A cheap operation on a data structure with n elements has an average-case runtime of `O(1)` or `O(lgn)`.

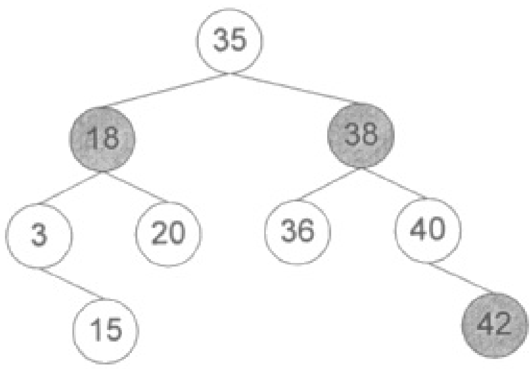
For each application scenario below, list the numeric identifiers of all the data structures enumerated above that exhibit the desired behavior, and state any assumptions you made:

(a) We want at least *member* to be cheap. (1.5 p)
(b) We want at least *member* , *insert* , and *remove* to be cheap. (1.5 p)
(c) We want at least *insert* , *findMax* , and *extractMax* to be cheap. (1.5 p)
(d) We want at least *closest* to be cheap. (1.5 p)

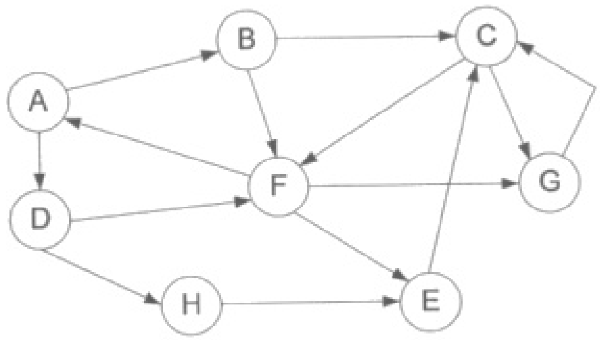
Exercise 4

(a) Draw a Trie with the alphabet a, b that contains the strings ε, a, ab, ab, aab, abb, abba, aabb, where ε indicates an empty string. (2 p)

(b) Is the following tree a legal red-black tree? Why/why not? The grey nodes indicate red nodes in the picture. (2 p)



(c) Show the order in which the breadth-first search goes through the graph below. The node A is the source and the edges are handled in an alphabetical order. Write your answer in the following format: "P grey, Q grey, P black...". (2 p)



Exercise 5

Given an array of integers a[0],...,a[N-1], below are three alternative methods for computing the *max-diff* property of the array. All three make use of methods min and max, defined elsewhere, for computing the minimum or maximum of a pair of integers. Throughout this problem, N denotes the number of items in the given array.

```
//---------------------- method 1 ----------------------------
    int method1(int a[]) {
        int N = a.length;
        int maxdiff = 0;
        for (int j = 0; j < N; j++)
            for (int i = 0; i < j; i++)
                maxdiff = max(maxdiff, a[j] - a[i]);

        return maxdiff;
    }
//-----------------------------------------------------------
```

```
//--------------------- method 2 ---------------------------
    int method2(int a[]) {
        int N = a.length;
        int maxdiff = 0;
        int minsofar = a[0];
        for (int k = 0; k < N; k++) {
            maxdiff = max(maxdiff, a[k] - minsofar);
            minsofar = min(minsofar, a[k]);
        }
        return maxdiff;
    }
//-----------------------------------------------------------
```

```
//---------------------- method 3 ----------------------------
    private static int helper3(int a[], int lt, int rt) {
        if (rt <= lt)      return 0;

        int s = (rt + lt) / 2;
        int mdl = helper3(a, lt, s);
        int mdr = helper3(a, s+1, rt);

        // compute min{a[lt], ..., a[s]}
        int minleft = a[lt];
        for (int i = lt; i <= s; i++)
            minleft = min(minleft, a[i]);

        // compute max{a[s+1], ..., a[rt]}
        int maxright = a[s+1];
        for (int j = s+1; j <= rt; j++)
            maxright = max(maxright, a[j]);

        int answer = ?????;
        return answer;
    }

    public static int method3(int a[]) {
        return helper3(a, 0, a.length - 1);
    }
//-----------------------------------------------------------
```

(a) Based on the definition of *method1* and *method2* above, explain what is the *max-diff* property of an array. (1 p)

(b) What is the worst-case running time of *method1* and *method2* as a function of N. Explain! (2 p)

(c) In *method3*, we left out the final computation of *answer* prior to returning it. In one line, how should *answer* be computed at this point as a function of the other variables that have already been computed? (2 p)

(d) What is the worst-case running time of *method3* as a function of N. Explain! (1 p)