Universidad de Chile

Departamento de Ciencias de la Computación

Fake Control 1 CC40A

Semestre: Agusto 2009

Solution

Problem 1 (1.5 marks)
Un `min-max priority queue` es un tipo de datos abstracto que provee las operaciones `deleteMin` y `deleteMax`, muy similar al tipo de datos abstracto `min priority queue`. En este problema, n denota la cantidad maxima de elementos en la fila.

1. Describa un `min-max heap`, una estructura de datos tomando espacio $n+O(1)$ que implementa las operaciones `deleteMin` y `deleteMax` en tiempo $\mathcal{O}(\lg n)$ para cada operación, utilizando un solo árbol binario, y técnicas muy similares a las utilizadas para un `heap` usual.

   ————————begin solution————————

   A simple solution, since no space requirement is specified, is to simply use any efficient ordered dictionary structure, such as an AVL tree, with two additional elements $-\infty$ and $+\infty$: the smallest element is the successor of $-\infty$ and the largest element is the predecessor of $+\infty$. The search for both, and the addition and removal of elements is supported in logarithmic time.

   Should one require better space than $O(n)$, one can consider a binary tree with the same shape property as a normal heap, but with the following ordering property, where "value" indicates the value stored at a node, and $p(x)$ indicates the parent of node x: for any node x at depth $2, 4, 6, \ldots$

   $$\text{value(p(p(x)))} < \text{value(x)} < \text{value(p(x))}$$

   and for any node x at depth $3, 5, \ldots$

   $$\text{value(p(x))} < \text{value(x)} < \text{value(p(p(x)))}$$

   and for nodes x at depth 1 (i.e. the children of the root)

   $$\text{value(p(x))} < \text{value(x)}$$

   ————————end solution————————

2. Describa un algoritmo para agregar un nuevo elemento en un `min-max heap` en tiempo $\mathcal{O}(\lg n)$. Justifique su análisis de complejidad.

   ————————begin solution————————

   Add the element to the first leaf available, as in a normal heap.

   Then, going up the path to the root, check if each node respects the min max heap property relative to its depth:

   $$\begin{gathered}\text{if } x \text{ has value } a,\ p(x) \text{ has value } b \text{ and } p(p(x)) \text{ has value } c,\\ \text{then } a \in [\text{mín}(b,c), \text{máx}(b,c)].\end{gathered}$$

   When it does not, reorder the three nodes of the path so that they respect the condition and check the property upward.

   As the tree is of height $\lg n$ and as each reordering requires a finite number of operations, the whole insertion requires at most $O(\lg n)$ operations.

————————————end solution————————————

3. Describa como buscar y borrar el elemento mínimo en un `min-max heap` en un tiempo mínimo. Explique la complejidad de su algoritmo.

————————————begin solution————————————

The minimum element is at the root. To remove it, exchange it with the last element, as in a normal heap. Then, going down the tree, check if each node respects the min max heap property relative to its depth, its two children and its four grand-children. When it does not, reorder the three nodes of the path so that they respect the condition and check the property downward in the only modified subtree.

As the tree is of height $\lg n$ and as each reoredering requires a finite number of operations, the whole deletion requires at most $O(\lg n)$ operations.

————————————end solution————————————

Problem 2 (1.5 marks)
Un centro medical ha contaminado una persona a dentro de N personas. La persona contaminada debe recibir un tratamiento a dentro de los 15 proximo dias.

Existe un test de sangre que, despues de un incubacion en maquina de 15 dias, indica si el sangre probado es contaminado o no. Los cientificos del laboratorio quieren ejecutar los N pruebas in parallel usando N maquinas, pero no tienen tan maquinas.

Observa que la prueba puede detectar pequeña cantidades del virus, como por ejemplo en una mezcla. Propone un protocolo para ejecutar solamente $O(\log N)$ pruebas en paralelo, y explicarlo en un ejemplo por $N = 16$. En particular, su respuesta debria especificar

1. el conjunto de pruebas ejecutadas;

2. una corta prueba que $\log N$ prubas son ejecutadas;

3. como analisar el resultado de las pruebas;

4. un ejemplo del protocolo para $N = 16$.

——————begin solution——————

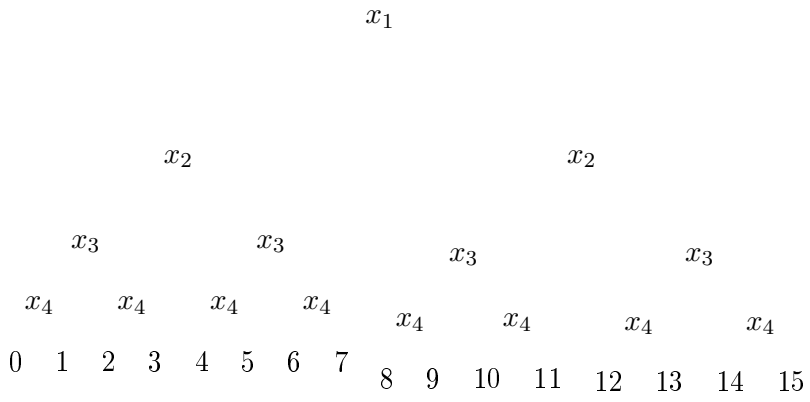
1. Prepare some potions, each containing blood from a number of samples as follows: Number the samples from 0 to $N-1$; write these numbers in binary, and look at the bits of the binary encodings from left to right; Potion p contains some blood from sample b if and only if the pth bit in the binary encoding for sample b is 1. This set of potions is the set of tests to be run.

2. Note that with $\lceil \lg N \rceil$ bits, you can encode $2^{\lceil lgN \rceil} \geq N$ different numbers in binary. The number of potions needed is the number of bits needed, so the number of tests is $n = \lceil \lg N \rceil$.

3. To determine who has been contaminated from the test results, define $x_p = 1$ if the potion p has a positive test result. The values of $(x_i)_{i \leq n}$ form a number x on n bits, and this number (between 0 and $N-1$) is the number of the contaminated sample.

4. For example, in the case where one person was contaminated among $N = 16$, the analysis of only $n = 4$ potions composed as described in the array below permits us to find this person, as shown in the tree below.

Potion 1 is a mix of the blood samples	8 to 15;
potion 2 is a mix of the blood samples	4 to 7 and 12 to 15;
potion 3 is a mix of the blood samples	$\{2, 3, 6, 7, 10, 11, 14, 15\}$;
potion 4 is a mix of the blood samples	corresponding to odd numbers.



————————end solution————————

Problem 3 (1.5 marks)
Sea un arreglo $A[1,\ldots,n]$ ordenado de n elementos, un elemento x, y la tarea de encontrar $p \in \{1,\ldots,n+1\}$ tal que $A[p-1] < x \leq A[p]$ si $p \leq n$ y $A[n] < x$ sino.

1. Cual es la definición del algoritmo de busqueda por interpolación? Cual es su complejidad en promedio si los elementos del arreglo estan selectionados uniformamente en un universo grande?

——————begin solution——————

```
interpolationSearch(A, k, n)
  i ← 1
  j ← n
  while i + 1 < j do
    guess ← i + ⌊(j − i) ∗ (k − A[i])/(A[j] − A[i])⌋
    if  k ≤ A[guess] then
      j ← guess
    else
      i ← guess
    end if
  end while
  return j
```

Si los elementos del arreglo estan selectionados uniformamente en un universo grande, su compljidad en promedio es $\mathcal{O}(\lg\lg n)$.

——————end solution——————

2. Cual es la definición del algoritmo de "doubling search"? Cual es su complejidad en el peor caso?

——————begin solution——————

```
doublingSearch(A, k, n)
  i ← 1
  j ← n
  gap ← 1
  while i + gap ≤ j and k > A[i + gap] do
    i ← i + gap
    gap ← 2 ∗ gap
  end while
  return binarySearch(A, k, i, mín{i + gap, j})
```

Si $A[p-1] < k \leq A[p]$, el algoritmo encontro p despues de $2\lg\lg p \in \mathcal{O}(\lg\lg p)$ comparaciones.

————————————end solution————————————

3. Da un algoritmo de busqueda por *extrapolación* inspirado de ambos los algoritmos de busqueda por interpolacion y de "doubling search", que adivina una position g basado en las dos ultimas posiciones comparadas a x (no se necessita una complejidad).

————————————begin solution————————————

```
extrapolationSearch(A, k, n)
  i ← 1
  j ← n
  while i + 1 < j do
    guess ← i + ⌊(j − i) ∗ (k − A[i])/(A[j] − A[i])⌋
    if  k ≤ A[guess] then
      j ← guess
    else
      i ← guess
    end if
  end while
  return j
```

Si los elementos del arreglo estan selectionados uniformamente en un universo grande, su compljidad en promedio es $\mathcal{O}(\lg \lg n)$.

————————————end solution————————————

Problem 4 (1.5 marks)

1. Cuales son las definiciones de las notaciones $O()$, $\Omega()$ y $\Theta()$?

——————————begin solution——————————

- $f(n) \in O(g(n))$ si
$$\exists c, n_0 > 0 tq \forall n > n_0, f(n) \leq cg(n)$$
- $f(n) \in \Omega(g(n))$ si
$$\exists c, n_0 > 0 tq \forall n > n_0, f(n) \geq cg(n)$$
(o si $g(n) \in O(f(n))$)
- $f(n) \in \Theta(g(n))$ si
$$\exists c_1, c_2, n_0 > 0 tq \forall n > n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$
$$f(n) \in O(g(n)) y f(n) \in \Omega(g(n))$$

——————————end solution——————————

2. Da el algoritmo para "Radix Sort".

——————————begin solution——————————

Considera un arreglo A de tamaño n sobre alfabeto de tamaño σ.

- si $\sigma = n$, se recuerdan que bucket sort puede ordenar A en tiempo $O(n)$
- si $\sigma = n^2$, bucket sort puede ser utilizado para ordenar A en tiempo $O(n)$ (utilizando la stabilidad de buket sort) :
  - 1 ves con los $\lg n$ bits de la derecha
  - 1 ves con los $\lg n$ bits de la izquierda
- si $|A| = n^c$, bucket sort puede ser utilizado para ordenar A en tiempo $O(cn)$ con espacio $2n + \sigma \approx 3n$ ($\sigma \approx n$ a cada iteracion de bucket sort)

El espacio se puede reducir a $2n+\sqrt{n}$ con $\lg n/2$ bits a cada iteracion de bucketsort, cambiando la complejidad solamente por un factor de 2.

En final, si A es de tamano n sobre un alfabeto de tamano σ, radix sort puede ordenar A en tiempo $\mathcal{O}(n\lceil\frac{\lg\sigma}{\lg n}\rceil)$

——————————end solution——————————

3. Describa el algoritmo para calcular el k-esimo elemento de un arreglo desordenado A.

——————————begin solution——————————

```
* ElijePivot(A,l,r)
  1. g <- (r-l+1)/5
  2. Hace g grupos de tamaño 5
  3. Calcula la mediana de cada grupo en un arreglo B[1,..,g]
  4. x <- LinealSelect(B,1,g,g/2)   (recursivamente)
  5. return p.

* LinealSelect(A,l,r,k)
  1. si l-r+1 = 1
     - return A[l]
  2. x <- ElijePivot(A,l,r)
  3. partitiona en A[i]<x, A[p]=x y A[j]>x, donde i<p<j
     (i.e. encontra la posición p de x en el arreglo ordenado)
  4. si k=p
     - return x
  5. sino si k<p
     - return LinealSelect(A,l,p-1,k)
  6. sino (si k>p)
     - return LinealSelect(A,p+1,r,k)
```

——————————end solution——————————