CIS 313: Intermediate Data Structure

week of Mar 11

tenth week of the term
linear time sorts

• Counting sort, Radix sort

• for Counting sort, we sort n elements, each in the range 0 to k (k fixed)
  • sometimes k=n
  • use element as array index

• simple version: count the number of items with value i, for $0 \leq i \leq k$
• use i as index to an array C
• then for each i print C[i] copies of i
• problem: i may be a key to larger element, need associated info
Counting-Sort$(A, B, k)$

1. let $C[0..k]$ be a new array
2. for $i = 0$ to $k$
3. \hspace{1cm} $C[i] = 0$
4. for $j = 1$ to $A.length$
5. \hspace{1cm} $C[A[j]] = C[A[j]] + 1$
6. \hspace{1cm} // $C[i]$ now contains the number of elements equal to $i$.
7. for $i = 1$ to $k$
8. \hspace{1cm} $C[i] = C[i] + C[i - 1]$
9. \hspace{1cm} // $C[i]$ now contains the number of elements less than or equal to $i$.
10. for $j = A.length$ down to 1
11. \hspace{1cm} $B[C[A[j]]] = A[j]$
12. \hspace{1cm} $C[A[j]] = C[A[j]] - 1$
example run: n=8, k=5

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(C[i]\) contains the number of elements \(i\)

\(C[i]\) now contains the number of elements \(\leq i\)

next 3 goes into location 6
radix sort

• sort on last digit (use *stable sort*)
• sort on next to last digit, etc
• old punch card readers did this

• if you have 100 bins, can sort on last two digits
• put into bins 0,1,2,...,99
RADIX-SORT(A,d)
  for i=1 to d
    use stable sort to sort array A on digit i

lemma 8.3
Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT correctly sorts these numbers in $O(d(n+k))$ time if the stable sort it uses takes $O(n+k)$ time.
Lemma 8.4
Given $n$ $b$-bit numbers and any positive integer $r \leq b$, RADIX-SORT correctly sorts these numbers in $O\left((b/r)(n+2^r)\right)$ time if the stable sort it uses takes time $O(n+k)$ time for inputs in the range $0$ to $k$.

Idea:
- break numbers into groups of $r$ bits
- view it as consisting of $b/r$ digits of $r$ bits each
- each digit has value $0$ to $2^r$
- sort on each digit is $O(n+2^r)$
- done $b/r$ times
1. Consider the use of lemma 8.4 of the text (p 199) to sort \( n \) numbers of \( b \) bits each. Suppose we choose \( b = \lg n \lg \lg n \).

(a) What is the largest integer that can be expressed (in unsigned binary) using \( b \) bits?
(b) If \( r \) is chosen to be \( \lg n \), how long does RADIAXSORT take, according to the lemma?
(c) How long does it take if \( r = \lg \lg n \) is chosen instead?
(d) Which choice of \( r \) is faster?
order statistics

• we say an element of an array $A$ has rank $k$ if $k-1$ elements of $A$ are less than or equal to the element
• one way to find an element of rank $k$ is to sort $A$, then look in location $k$
• thus, to find the median
  • sort $A$
  • return element in location $n/2$
• but that takes time $O(n \lg n)$
• we want to do it faster: linear or almost linear time
revisit partition

- partition puts small elements (low rank) on the left
- large elements (high rank) on the right
what partition does

- pivot usually last element
- Randomized-Partition chooses random element to be x

the sizes of the left and right sides depend on the value of x
Randomized-Select(A,p,r,i)
1. if p==r
2. return A[p]
3. q = Randomized-Partition(A,p,r)
4. k = q-p+1
5. if i==k   //pivot is answer
6. return A[q]
7. else if i<k
8. return Randomized-Select(A,p,q-1,i)
9. else return Randomized-Select(A,q+1,r,i-k)

*text version*

**Randomized-Select(A,p,r,i)**
1. if $p=r$
2. return $A[p]$
3. $q = \text{Randomized-Partition}(A,p,r)$
4. $k = q-p+1$
5. if $i=k$    //pivot is answer
6. return $A[q]$
7. else if $i<k$
8. return $\text{Randomized-Select}(A,p,q-1,i)$
9. else return $\text{Randomized-Select}(A,q+1,r,i-k)$

**time:**

$O(n^2)$ worst case (low probability)

$O(n)$ expected time
other order statistics

• find all the quintiles: the elements of rank $n/5$, $2n/5$, $3n/5$, $4n/5$
• can be done in linear expected time
  • run quicksort until those positions are fixed
worst case linear time

• randomized select is $O(n^2)$ worst case
• there is an $O(n)$ worst case method: median of medians
  • break list into $n/5$ groups of 5
  • find median of each of those groups (directly)
  • recursively find the median $x$ of those $n/5$ medians
  • use $x$ as a pivot element
  • then continue recursively

• recurrence relation $T(n) = T(n/5)+T(7n/10)+O(n)$
  • solution $T(n)=O(n)$
  • bad constants so not practical
  • section 9.3
exercise 9-1

• given n numbers, we want the i largest in sorted order. Compare:
• sort the numbers, list the i largest
• build max priority queue from the numbers, call extract max i times
• use order statistic method to find ith largest, partition around that number, sort the i largest numbers
• something with min priority queue?