Second Midterm Exam, CMPSC 465, Spring 2009

Practice problems

- Midterm will be on Tuesday, March 31, 8:15 PM, in 60 and 61 Willard.
- This will be open book exam, you can also have notes (several sheets or one notebook). The use of communication equipment and calculators is not permitted the exam.
- When you describe an algorithm, you should also justify its correctness and your estimate of the running time.
- In problems where you show the working of an algorithm on diagrams each diagram is repeated to provide you with a “spare one”. Indicate which diagram contains your solution.

1. **Balanced trees, including B-trees**
   
   Given a list of numbers and priorities, draw the corresponding treap, a binary search tree in which parent cannot have a smaller priority (thus the number with the largest priority is on the root).

   Example: (number, priority): (0,8), (1,28), (2,3), (3,25), (4,1), (5,17), (6,0), (7,9), (8,12), (9,6), (10,19), (11,17).

   ![Treap Diagram]
Page 222, exercise 7a (unstable/illegal nodes are in red)

We can simply insert the list elements to an initially empty 2-3 tree. We can do it faster if the list is already sorted. I will put a recursive code later (Saturday night).

Page 222, exercise 10 (pseudocode).

We can simply insert the list elements to an initially empty 2-3 tree. We can do it faster if the list is already sorted. I will put a recursive code later (Saturday night).
2. Heaps and Heapsort

Page 229, exercises 1, 3, 6.

Exercise 3. a. Finding the minimum number of keys in a heap of height \( h \): we have \( h + 1 \) levels which we can number from 0 to \( h \); all levels except the last will have sizes in the sequence 1, 2, 4, etc., and the smallest possible last level has one key. Thus we have size \( 1 + 2 + \ldots + 2^{h-1} + 1 = 2^h \).

b. If \( \lfloor \log_2 n \rfloor = h \) then \( 2^h \leq n < 2^{h+1} \), so by the previous observation, the heap of \( n \) keys must have height at least \( h \) and below \( h + 1 \), hence, exactly \( h \).

Exercise 6.

To sort in increasing order, we use heap with the maximum at the root, so when we extract the value from the root we can send it to the end of the array fragment occupied by the heap.

a. On the midterm, I expect to see diagrams for heaps, but here I save space (I will add diagrams on Saturday night):

Making heap of 1, 2, 3, 4, 5: we get 5, 4, 3, 1, 2 (we checked positions 2, 1 and moved the values lower as needed).

Maximum swapped with the end: 2, 4, 3, 1; 5
Correcting the heap: 4, 2, 3, 1; 5
Maximum swapped with the end: 1, 2, 3; 4, 5
Correcting the heap: 3, 2, 1; 4, 5
Maximum swapped with the end: 1, 2; 3, 4, 5
Correcting the heap: 2, 1; 3, 4, 5;
Maximum swapped with the end: 1; 2, 3, 4, 5
No need to correct the heap.

b. Making heap of 5, 4, 3, 2, 1: already a heap.

Maximum swapped with the end: 1, 4, 3, 2; 5
Correcting the heap: 4, 1, 3, 2; 5 \( \rightarrow 4, 2, 3, 1; 5 \)
Maximum swapped with the end: 1, 2, 3; 4, 5
Correcting the heap: 3, 2, 1; 4, 5
Maximum swapped with the end: 1, 2, 3, 4, 5
Correcting the heap: 2, 1; 3, 4, 5
Maximum swapped with the end: 1, 2, 3, 4, 5
3. Problem Reduction

Page 246, exercises 5, 9, 11. Also, 3 together with 4a.

Exercise 5.

Are all points $P_1, \ldots, P_n$ contained in a triangle with the vertices from that sequence?

We can find the convex hull, and then we check if it is a triangle. We know that a convex hull can be found in time $O(n \log n)$.

Exercise 9.

We have a graph with edges $e_0, \ldots, e_{m-1}$ and we want to find $C[1], \ldots, C[m]$ so if edges $e_i, e_j$ have a common endpoint then $C[i] \neq C[j]$.

We can translate this to a node coloring problem: we create a graph with nodes $0, \ldots, m-1$ and edges $\{i, j\}$ such that $e_i$ and $e_j$ share an endpoint. A valid coloring (array $C[m]$) for the nodes of this graph is also a valid coloring of the edges of the previous graph. Because colorings are the same, an algorithm producing the minimum number of colors for nodes of the new graph will produce the minimum number of colors for the edges of the first graph.

Exercise 11.

I though that there is a nice reduction from a problem with $k$ couples to a problem with $k-1$ couples, but the solution I got for 3 couples strongly relies on the fact that there are at most 3 men. One can show that if $k > 3$ there is no solution.

So this puzzle, at the very least, is too hard to be given during an exam. I will post a solution on Sunday, so the curious among you can check.

Solution for $k = 3$: $h$ is a husband, $w$ is a wife, $B$ is the boat. Columns indicate who and what is on each of the two river banks.

<table>
<thead>
<tr>
<th></th>
<th>hw · hw · hw B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hw · hw · hw B</td>
</tr>
<tr>
<td>1</td>
<td>hw · hh</td>
</tr>
<tr>
<td>2</td>
<td>hw · hw · h B</td>
</tr>
<tr>
<td>3</td>
<td>hhh</td>
</tr>
<tr>
<td>4</td>
<td>hw · hh B</td>
</tr>
<tr>
<td>5</td>
<td>hw</td>
</tr>
<tr>
<td>6</td>
<td>hw · hw B</td>
</tr>
<tr>
<td>7</td>
<td>ww</td>
</tr>
<tr>
<td>8</td>
<td>www B</td>
</tr>
<tr>
<td>9</td>
<td>w</td>
</tr>
<tr>
<td>10</td>
<td>ww B</td>
</tr>
<tr>
<td>11</td>
<td>hw · hw · hw B</td>
</tr>
</tbody>
</table>

However, if $k > 3$, then we cannot reach a state that either has two (or more) husbands on the far side of the river and the boat on the near side, or three (or more) husbands on the far side of the river, and the boat there. This is because when we have one husband on each side, we must have all wives with husbands, so wives cannot move alone (away from husbands) and husbands cannot move alone either, UNLESS all husbands leave a river bank together.
But if they go together to the near side of the river, we are back at the initial state. If they go together to the far side of the river, there are only two of them on the near side, so at least two on the far side, without a boat there. We can never reach that state.

Exercise 3 and 4a.

Exercise 3: For $k = 1$ the number of paths with $k$ edges from $i$ to $j$ is zero if there is no edge, and one if there is an edge, so it is $A_{ij}$ where $A$ is the adjacency matrix.

For $k = 0$ the number of paths with $k$ edges from $i$ to $j$ is zero if $i \neq j$ and one if $i = j$, so it is given by the identity matrix $I_{ij}$, or $A^0$.

For larger $k$, the number of paths with $k$ edges from $i$ to $j$ is the sum over possible second nodes of paths of the products: (1 if there is an edge from $i$ to $m$, 0 otherwise) times (the number of paths from $m$ to $j$ with $k - 1$ edges), and this sum is the product of $i$-th row of $A$ with $j$-th column of $A^{k-1}$, so it is an entry of $A^k$.

Exercise 4a: we need to check if $A^3$ has a non-zero. It suffices to compute $A^3$. Thus it suffices to compute twice a product of two $n \times n$ matrices.

We reduced the problem of fast checking if there is a triangle to the problem of fast matrix multiplication, and we know about the Strassen algorithm that works in time $O(n^{\log_2 7})$ which is faster than $O(n^3)$. 

5
4. Counting sort and Radix sort

Sort given sequences alphabetically using Radix sort, show the situation after each round.
Example: CAB, CA, ABD, DA, ADA, CDA, BCD, D, AB, AA, C, BD.

In a round we create lists with a particular value of the character that we “look at”; I separate these lists by semicolons to make the method easier to follow.

In the first round, we sort according to the 3rd characters
CAB, ABD, ADA, CDA, BCD → ADA, CDA; CAB; ABD, BCD

we add in front sequences with two characters and sort according to the 2nd characters:
CA, DA, AB, AA, BD, ADA, CDA, CAB, ABD, BCD →
CA, DA, AA, CAB; AB, ABD; BCD; BD, ADA, CDA

we add in front sequences with one characters and sort according to the 1st characters:
D, C, CA, DA, AA, CAB, AB, ABD, BCD, BD, ADA, CDA →
AA, AB, ABD, ADA; BCD, BD; C, CA, CAB, CDA; D, DA

Page 254, exercise 6 (Hint: you need to number nodes in a tree so that in each subtree you have a consecutive interval; then for each node store three numbers: its number, the smallest number in its subtree, the largest number in its subtree.)

The hint can be translated into recursive code using global counter counter initialized to 0:

```c
Number(tree node t)
{
    t->num = counter++
    for (each child c of t)
    {
        Number(c)
        t->range = counter
    }
    Now s is a descendant of t if and only if
    t->num < s->num && s->num < t->range
}
```

5. Hashing

page 270, exercise 1, 10.

Problem: given a number array $A[n]$, find a pair of entries with the minimum difference. Describe an algorithm that runs, on the average, in $O(n)$ time.

Solution: pick a random sample of the array, say, $S[n/2]$. Solve the problem for $S$, this takes $T(n/2)$ time. We get a minimum difference for $S$, say $L_S$. We can find non-empty lists with the same value of $\lceil A[i]/L_S \rceil$. We have hash table of non-empty lists.

Then in each list we can find the minimum difference by brute force. According to what we learned in class, the number of pairs that are in the same list is $O(n)$. We can also find minimum and maximum in each list. Then if list with value $v$ is non-empty, we check if list with value $v - 1$ is also non-empty. If not, then the differences of elements of list $L$ with smaller elements are larger than $L$ and they do not have to be considered.

If yes, we check just one such difference, with the maximum of list $v - 1$. 
6. Dynamic programming

Compute \( N(5,5,6) \), the number monotone paths from \((0,0)\) to \((5,5)\) with area 6.

A monotone path can be represented as a non-decreasing sequence that indicates how many grid squares are below the path in the columns of the grid. As the path is monotone, these numbers cannot decrease, and they add to the area under the path.

\[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 2 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 3 \\
0 & 0 & 1 & 2 \\
0 & 1 & 1 & 1 \\
0 & 0 & 2 & 2 \\
0 & 1 & 1 & 2 \\
1 & 1 & 1 & 2 \\
\end{array}
\]

page 292, exercise 1, 9.

Exercise 1: the graph is a path \(0 \rightarrow 1 \rightarrow 2 \rightarrow 3\).

Allowing paths of length 0 changes the matrix to

\[
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{array}
\]

Allowing 0 as intermediate node does not change the matrix, because there are no edges to 0. Allowing 1 as an intermediate node adds path \(0 \rightarrow 1 \rightarrow 2\) so the new matrix is

\[
\begin{array}{cccc}
1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{array}
\]

Allowing 2 as an intermediate node adds \(0 \rightarrow 2 \rightarrow 3\) and \(1 \rightarrow 2 \rightarrow 3\) so the new matrix is

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
\end{array}
\]

which is the final result.

Exercise 9.

To make Floyd algorithm incorrect, we need a negative cost cycle, e.g. the graph could be a directed cycle \(0 \xrightarrow{1} 1 \xrightarrow{1} 2 \xrightarrow{3} 0\).
Example of a problem: Subset Sum. Given are numbers $x_0, \ldots, x_1$ and target number $t$, the question is if there exists a subset $S \subset \{0, \ldots, n-1\}$ such that $\sum_{i \in S} x_i = t$.

Recursive subproblem: for $k \leq n$, $s \leq n$ we define $SP(k, s)$ (abbreviation of “Sum Possible”) which is true if and only if there exists $S \subset \{0, \ldots, k-1\}$ such that $\sum_{i \in S} x_i = s$.

Basic case: $SP(0, s) = [s = 0]$ (here $[s = 0]$ is true or false).

Recurrence: $SP(k+1, s) = SP(k, s)$ or $SP(k, s - x_k)$.

Resulting code:

```c
for (i = 1; i < n; SP[i++] = 0);
SP[0] = 1;
for (k = 0; k < n; k++) = 0
    for (s = t; s >= x[k]; s--) = 0
        SP[s] ||= SP[s-x[k]];
```

Running time $O(nt)$, memory neede $O(n)$.

Example of a problem: Longest Common Subsequence. Given are two sequences, $x_0, x_1, \ldots, x_{m-1}$ and $y_0, y_1, \ldots, y_{n-1}$.

A subsequence is obtained from a sequence by deleting some of its entries without changing the order of the remaining elements.

We want to know the length of the longest common subsequence.

E.g. if we have CABAB and ABBACBACA, ABAB is the longest common subsequence so we should compute 4.

Recursive subproblem: $LCS(a, b)$, the length of the longest common subsequence of $x_0, x_1, \ldots, x_{a-1}$ and $y_0, y_1, \ldots, y_{b-1}$.

Basic case: $LCS(0, b) = LCS(a, 0) = 0$ (one of the sequences is empty).

Recurrence:

$$LCS(a+1, b+1) = \max \left\{ LCS(a+1, b) \ (y_b \text{ not used}) \right\}$$

This recurrence can be easily converted into a program that fills two-dimensional array $LCS$ in time $O(mn)$.

One can observe that it suffices to keep two rows or two columns in the memory, because entries in column $i$ are computed using entries from column $i$ and $i-1$. This would give memory requirement of $O(m)$.  

8