Institut for Matematik og Datalogi
Syddansk Universitet

March 2, 2009
JFB

DM508 – Algorithms and Complexity – 2009
Lecture 7

Lecture, March 2

We finished NP-Completeness and began on amortized analysis from chapter 17 of the textbook and Fibonacci heaps from chapter 20. We covered up through the analysis of the Create, Insert, Union, and Minimum operations on Fibonacci heaps.

Lecture, March 4 at 12:15

We will finish with Fibonacci heaps (starting with the Extract-Min operation) and begin on string matching from chapter 32.

Lecture, March 9

We will continue with string matching.

Problems to be discussed on March 11

- 20.3-1.
- 20.4-1.
- 20-1, 20-2a.
- Consider the set-up from Problem 6 on the first lecture note, with a company with two service representatives, originally in Detroit and San Francisco, with requests for service coming from Detroit, San Francisco and Chicago. Traveling between Chicago and Detroit costs f, while traveling between Chicago and San Francisco costs df. The company wants to minimize its costs.

  Consider the following algorithm, *Savings*: The representative in San Francisco, called "Rep A", keeps a savings account to pay for tickets, as does the representative in Detroit, called "Rep B". These accounts start at zero. Given a request, if a service representative is already there, do nothing (except empty the account for the representative there if the request was San Francisco or Detroit). If not, and the request is not for Chicago, move the representative in Chicago there (and empty the account for that representative). If the request is for Chicago and no representative there, if the amount in Rep A's account is less than or equal to $(d-1)f$, then move the representative from Detroit to Chicago and have Rep A add f minus the amount in Rep B's account to its savings account and Rep B empty its account. Otherwise, move the representative from San Francisco to Chicago and have Rep B add df minus the amount that was in Rep A's account to its account and Rep A empty its account.

The following questions are an analysis of *Savings* compared to any other algorithm O using amortized analysis. Let $S_A(L)$ be the savings that Rep A has in its account after processing the requests in L and $S_B(L)$ be the same for Rep B. Let the potential function be $\Phi(L) = 2M(L) + C(L)$, where $M(L)$ is the cost that would be necessary (after both *Savings* and O process L) to move *Saving*'s representatives to the same places as O's (minus $S_A(L)$ if Rep A would have to move and plus $S_A(L)$ otherwise, minus $S_B(L)$ if Rep B would have to move and plus $S_B(L)$ otherwise), and $C(L)$ is the cost to fly between the two cities where *Savings* currently has representatives.

Note that the savings accounts have value zero if the representative is in Chicago or if the representative has just returned to its home city. If the one algorithm has representatives in Chicago and San Francisco, while the other has representatives in Chicago and Detroit, we say that to move the representatives to the same places, both servers move (Rep A should never go to Detroit and Rep B should never go to San Francisco.)

1. Suppose *Savings* has representatives in Chicago and San Francisco. What is the actual cost of servicing a request in Detroit, and what is the amortized cost (consider each of the possible cases of where O could originally have had its representatives)?
2. Suppose *Savings* has representatives in Chicago and Detroit. What is the actual cost of servicing a request in San Francisco, and what is the amortized cost (consider all possible cases)?
3. Suppose *Savings* has representatives in Detroit and San Francisco and Rep A has $S_A(L)$ in its savings account. What is the actual cost of servicing a request in Chicago, and what is the amortized cost (consider all possible cases)?
4. Argue that on any sequence of requests *Savings* has cost at most twice that of any other algorithm plus an additive constant of at most df. (Remember to take into account potential changes when none of *Savings*'s representatives moves.)

Assignment due Monday, March 16, 12:15

Note that this is part of your exam project, so it must be approved in order for you to take the exam in April, and you may not work with or get help from others not in your group. You may work in groups of two or three. You may write your solutions in English or Danish, but write very neatly if you do it by hand.

1. Consider a counter which, instead of being binary, is kept in ASCII form, base 10, and assume that m digits are stored. Thus, the number two hundred fifteen (215) is stored in an array, C, of length m, where $C[0] = 53$ (00110101 in binary, the ASCII code for 5), $C[1] = 49$, $C[2] = 50$, and $C[3] = C[4] = ... = C[m-1] = 48$. Give an algorithm for incrementing this type of counter and analyze it using the potential function method. You may assume that you can increment an ASCII value, check an ASCII value, or store an ASCII value in one time unit.

2. Suppose that you want a data structure where you can do searching for keys in logarathmic time, and you want to be able to insert elements into the data structure, but you want to use arrays, rather than balanced binary search trees. Assume that you can compare keys or copy elements in constant time, and that you can create an empty array of size 2^s in time proportional to 2^s. Suppose that at some time there are m elements in your data structure. Let $k = \lceil \log_2(m+1) \rceil$ be the number of bits in the binary representation of m, and define m_i for $0 \leq i \leq k$ to satisfy $m = \sum_{i=0}^{k-1} m_i 2^i$ and $m_i \in \{0, 1\}$. The m elements will be stored in k arrays, $S_0, S_1, ..., S_{k-1}$, where each S_i is capable of storing 2^i elements and is either empty or full, depending on whether $m_i = 0$ or $m_i = 1$, respectively. Thus, the total number of elements stored is $m = \sum_{i=0}^{k-1} m_i 2^i$, as required. Each array should be sorted, but one cannot assume any relationship between the elements in different arrays. The following is an example of how the 11 elements $\{1, 3, 4, 7, 10, 11, 13, 18, 21, 23, 27\}$ might be stored: $m = 11$, $k = 4$, $S_0 = [7]$, $S_1 = [3, 13]$, $S_2 = []$, $S_3 = [1, 4, 10, 11, 18, 21, 23, 27]$.

   (a) Give an algorithm for searching for a key in this data structure. Analyze its worst-case running time.

   (b) Give an algorithm for insertion of a new element in this data structure. Analyze its running time using the potential function method.

3. What is the prefix function computed by the KMP algorithm for the string $P = cabbcabbacbcabba$.