École Polytechnique Fédérale de Lausanne
Prof. Eisenbrand

March 1, 2018

Discrete Optimization (Spring 2018)

Assignment 2

Problem 5 can be **submitted** until March 9 12:00 noon into the box in front of MA C1 563. You are allowed to submit your solutions in groups of at most three students.

Problem 1

Describe an algorithm that multiplies two n-bit integers in time $O(n^2)$. You may assume to have a subroutine $Sum(d, e)$ which returns the sum of two n-bit natural numbers d and e in time $O(n)$.

Solution:

One can apply the classical algorithm for multiplying two decimal numbers, with the difference of using base-2 instead of base-10. We call $Sum(d, e)$ as a subroutine and assume that it returns an $(n+1)$-bit representation of $d + e$.

Input: Two natural numbers a and b in their binary representations
$a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$, respectively.
Output: The binary representation $c_0, \ldots, c_{2n}$ of $a \cdot b$.

$(c_0, c_1, \ldots, c_{2n}) := (0, 0, \ldots, 0)$
for $i = 0, \ldots, n-1$
  if $(b_i = 1)$
    $(c_i, \ldots, c_{i+n}) := Sum((c_i, \ldots, c_{i+n-1}), a)$
return $c_0, \ldots, c_{2n}$

We can assume that subroutine $Sum(d, e)$ is implemented similarly to the algorithm from Assignment 1, Problem 8, thus one call of it requires $O(n)$ time. Thus each of the n loop iterations takes $O(n)$ time.[1] The initial assignment before the loop is performed in $O(n)$ time as well. Overall execution time is then $O(n) + n \cdot O(n) = O(n^2)$.

Problem 2

Suppose $a, b \in \mathbb{N}$ are two n-bit integers, where n is a power of 2. Consider the first and the last $n/2$ bits of a, and denote their corresponding decimal numbers with a' and a'', respectively. Likewise decimal numbers b' and b'' correspond to the first and the second half of the bit-representation of b.

i) Show that $a = a' + a'' \cdot 2^{n/2}$ and $b = b' + b'' \cdot 2^{n/2}$.

ii) Show that $a \cdot b = a' \cdot b' + (a' \cdot b'' + a'' \cdot b') \cdot 2^{n/2} + a'' \cdot b'' \cdot 2^n$.

iii) Show that $(a'b'' + a''b') = (a' + a'')(b' + b'') - a' \cdot b' - a'' \cdot b''$.

iv) Design a recursive algorithm for n-bit integer multiplication whose running time $T(n)$ satisfies the recursion

$$T(n) \leq 3 \cdot T(n/2) + c \cdot n,$$

[1] The most expensive operation inside the loop is the call of $Sum(d, e)$ subroutine, while the assignment can be implemented to run in constant time.

where $c > 1$ is some constant.

Hint: You can assume that there is a constant c' such that two n-bit numbers can be added and subtracted using at most $c'n$ basic operations.

v) *Unroll* the recursion above three times.

vi) Conclude that two n-bit numbers can be computed in $O(n^{\log_2(3)})$ elementary bit operations.

Solution:

i) By the definition we have that

$$a = \sum_{i=0}^{n/2-1} a_i \cdot 2^i + \sum_{i=n/2}^{n-1} a_i \cdot 2^i = a' + 2^{n/2} \sum_{i=0}^{n/2-1} a_{n/2+i} \cdot 2^i = a' + 2^{n/2} \cdot a''.$$

The result analogously follows for b.

ii) By using i) one has $a \cdot b = (a' + a'' \cdot 2^{n/2})(b' + b'' \cdot 2^{n/2}) = a' \cdot b' + (a' \cdot b'' + a'' \cdot b') \cdot 2^{n/2} + a'' \cdot b'' \cdot 2^n$.

iii) By performing basic arithmetics one easily verifies that the statement is true.

iv) Algorithm: $Mult(a, b)$
Input: Two natural numbers a and b in their binary representations $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$, respectively.
Output: The binary representation $c_0, \ldots, c_{2n}$ of $a \cdot b$.

$(c_0, c_1, \ldots, c_{2n}) := (0, 0, \ldots, 0)$
if $n = 1$
  return $a_1 \wedge b_1$
else
  $p_1 := Mult(a', b')$
  $p_2 := Mult(a'', b'')$
  $p_3 := Mult(a' + a'', b' + b'') - p_1 - p_2$
  $c := p_1 + (\underbrace{0, \ldots, 0}_{n/2}, p_3) + (\underbrace{0, \ldots, 0}_{n}, p_2)$
return $c_0, \ldots, c_{2n}$

From iii) one can see that ab can be obtained by three multiplications of $n/2$-bit numbers [2], namely $(a' + a'')(b' + b'')$, $a'b'$ and $a''b''$, and six subtractions/additions of numbers that can be represented with $2n + 1$ bits. Each of those subtractions/additions can be done in $c' \cdot n$ basic operations following Assignment 1, Problem 8. Thus the total number of operations is given by the recursive relation $T(n) \leq 3 \cdot T(n/2) + c \cdot n$, where $c = 6c'$.

v) By enrolling the recursion three times we obtain that

$$T(n) \leq 3 \cdot (3 \cdot (3 \cdot T(n/8) + c \cdot n/4) + c \cdot n/2) + c \cdot n = 3^3 T(\frac{n}{2^3}) + 3^2 c \frac{n}{2^2} + 3c\frac{n}{2} + cn.$$

[2] Strictly speaking, $a' + a''$ can be an $(n/2+1)$-bit number. However, this does not change the asymptotic behavior.

vi) From v) we establish the closed form formula

$$T(n) \leq \sum_{i=0}^{\log_2 n} (3/2)^i cn \leq 3(3/2)^{\log_2 n} cn = \theta(n^{\log_2 3}),$$

where we used that $\log_{3/2} n = \frac{\log_2 n}{\log_2(3/2)}$ which is equivalent to $\log_2 n = \log_{3/2}(n)(\log_2(3) - 1)$.

Problem 3

The *determinant* of a matrix $A \in \mathbb{R}^{n \times n}$ can be computed by the recursive formula

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}),$$

where A_{1j} is the $(n-1)(n-1)$ matrix that is obtained from A by deleting its first row and j-th column. This yields the following recursive algorithm (see the lecture notes, Example 1.4).

Input: $A \in \mathbb{R}^{n \times n}$
Output: $\det(A)$
if $(n = 1)$
  return a_{11}
else
  $d := 0$
  for $j = 1, \ldots, n$
    $d := (-1)^{1+j} \det(A_{1j}) + d$
  return d

Let $A \in \mathbb{R}^{n \times n}$ and suppose that the n^2 components of A are pairwise different.

i) Suppose that B is a matrix that can be obtained from A by deleting the first k rows and k of the columns of A. How many (recursive) calls of the form $\det(B)$ does the algorithm create?

ii) How many different submatrices can be obtained from A by deleting the first k rows and some set of k columns? Conclude that the algorithm remains exponential, even if it does not expand repeated subcalls.

Solution:

i) Let $i_1, \ldots, i_k$ be the indices of the columns of A that were removed to obtain B. We have to count the number of nodes of the form $\det(B)$ in the recursion tree of the algorithm. Each node of the tree can be identified by its level (nodes of the form $\det(A_{ij})$ are at level i) and a sequence of column indices representing the columns of A that are not columns of the submatrix called by the node. Hence the nodes of the form $\det(B)$ are at level k of the tree, and their sequences are permutations of $i_1, \ldots, i_k$ (note that there is only one submatrix of A equal to B since all the entries are pairwise different). Hence there are $k!$ such nodes.

ii) The number of such submatrices is clearly $\binom{n}{k}$, which is exponential for instance for $k = n/2$ (assume n even for simplicity):

$$\binom{n}{\frac{n}{2}} = \frac{n \cdots (\frac{n}{2} + 1)}{\frac{n}{2} \cdots 1} \geq 2^{\frac{n}{2}}.$$

Hence, even if the algorithm calls $\det(B)$ only once for any submatrix B, it remains exponential.

Problem 4

In this exercise, you will see that matrix multiplication is in some sense not harder than matrix inversion.
Suppose that $I(n)$ with $I(n) = \Omega(n^2)$ is a function that satisfies $I(3\,n) = O(I(n))$ and that a non-singular $n \times n$ matrix can be inverted using $I(n)$ arithmetic operations. Show that two $n \times n$ matrices A and B can be multiplied using $O(I(n))$ arithmetic operations.
Hint: Construct an upper triangular $3n \times 3n$*-matrix that contains* A *and* B*.*

Solution:
By following the hint one can construct the following matrix

$$D = \begin{pmatrix} I & A & 0 \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix} \tag{1}$$

with its inverse

$$\begin{pmatrix} I & -A & AB \\ 0 & I & -B \\ 0 & 0 & I \end{pmatrix}, \tag{2}$$

where I is the $n \times n$ identity matrix. Constructing the matrix D can be done in time $\theta(n^2)$ which is $O(I(n))$ since $I(n) = \Omega(n^2)$. By inverting D we get AB as an entry of the resulting block matrix. Thus, multiplying A and B can be done as efficiently as the inversion of D, i.e. by using $O(I(n))$ arithmetic operations.

Problem 5 ($\star$)

Let M_{2^k} be a matrix of order $n := 2^k$, where $k \in \mathbb{N}_{>0}$ such that it is recursively defined as follows:

$$M_{2^k} = \begin{pmatrix} M_{2^{k-1}} & M_{2^{k-1}} \\ M_{2^{k-1}} & -M_{2^{k-1}} \end{pmatrix} \tag{3}$$

and $M_1 = [1]$. Prove that $|\det(M_n)| = n^{n/2}$, i.e. that the Hadamard bound is tight.

Solution:
We prove the statement by induction on $k \in \mathbb{N}_0$ that $M_{2^k}^2 = 2^k I_{2^k}$. For $k = 0$ one has $M_{2^0} = M_1 = [1] = 2^0 I_{2^0}$. Assume that the statement holds for k and prove it for $k+1$.

$$M_{2^{k+1}}^2 = \begin{pmatrix} M_{2^{k-1}} & M_{2^{k-1}} \\ M_{2^{k-1}} & -M_{2^{k-1}} \end{pmatrix} \begin{pmatrix} M_{2^{k-1}} & M_{2^{k-1}} \\ M_{2^{k-1}} & -M_{2^{k-1}} \end{pmatrix} = \begin{pmatrix} 2M_{2^k}^2 & 0 \\ 0 & 2M_{2^k}^2 \end{pmatrix} \overset{I.H.}{=} \begin{pmatrix} 2 \cdot 2^k I_{2^k} & 0 \\ 0 & 2 \cdot 2^k I_{2^k} \end{pmatrix} = 2^{k+1} I_{2^{k+1}} \tag{4}$$

By using the statement above we have that $\det(M_n^2) = \det(nI_n) = n^n$ so $|\det(M_n)| = n^{n/2}$.