CS 667 : Homework 4(Due: Mar 27, 2012)

This Homework is Problems 1-7 and worth 250 points. You can replace some of these points with Problem 6 or 7, but you can only submit 250 points worth of problems.

Problem 1. (25 POINTS)
Perfect power revisited (BIT model). (You have the benefit of the solution of this problem on the WORD model.) Determine whether $n = x^y$ is an integer power of an integer (i.e. perfect power) in $O(\lg^3 n)$ bit operations. Can you show that this can be done in $O(M(\lg n))$ operations where $M(t)$ is the time to multiply two t-bit integers?

Note: If you can't prove the $O(\lg^3 n)$, partial credit will be given for solutions that can come close to it. For the second question involving $M(.)$, a YES or NO with or without justification suffices.

Problem 2. (25 POINTS)
Solve the recurrence for the number of operations in Strassen's method exactly, i.e. solve

$$M(n) = 7M(n/2) + (18/4)n^2.$$

What is the base case? Strassen or Cannon? Explain. How many operations are required for the base case? Explain. Does it matter whether one chooses the base case to be $n = 1$ or $n = 2$? Be very precice with your calculations. An answer $M(n) = \Theta(n^{\lg 7}$ would not be a satisfactory answer.

Problem 3. (25 POINTS)
You are given n distinct polynomials $f_1(x), \ldots, f_n(x)$ each one of degree $n-1$ and thus of degree bound n, where each polynomial looks similar to the generic polynomial $f(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. You are also given n points $c_1, c_2, \ldots, c_n$. Give an algorithm that evaluates the n polynomials at these n points in overall time $O(n^{\lg 7})$.

Problem 4. (25 POINTS)
(a) What is the largest k such that if you can multiply 5×5 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o(n^{\lg 7})$? What would the running time of this algorithm be?
(b) V. Pan has discovered a way of multiplying 68×68 matrices using 132464 multiplications, 70×70 matrices using 143640 multiplications, 72×72 matrices using 155424 multiplications. Which method yields the best asymtotic running time when used in a divide and conquer matrix multiplication algorithm? How does it compare to Strassen's algorithm?

Problem 5. (50 POINTS)
NJIT introduces its own three different types of stamps; 1 cent, 5 cent, and 6 cent stamps. We would like to know what the minimum number of stamps is for an n cent letter. Give an efficient algorithm that if given n as input, it prints as output the minimum collection of stamps that has value exactly n. Analyze the time and space requirements of your algorithm. Prove its correctness. For example, you can mail a 30 cent letter by using five 6-cent stamps; other alternatives include six 5-cent ones, or thirty 1-cent ones, or say three 6-cent, two 5-cent, and two 1-cent stamps.

Problem 6. (50 POINTS)
Let $M_t(n)$ be the time to multiply two $n \times n$ triangular matrices and $I_t(n)$ the time to invert an $n \times n$ triangular matrix. Let $M(n), I(n)$ be the corresponding times for general matrices.
(a) Show that $M_t(n) = \Theta(M(n))$, i.e. if you triangular multiply, then you can multiply and the inverse (which is obvious).
(b) Show that $I_t(n) = \Theta(M(n))$.
Hints. The page 18 of Subject 4 trick might help if slightly modified for (a). Do something similar to page 19 but now A becomes a lower-triangular matrix for (b)!

Problem 7. (50 POINTS)
Transitive closure in graphs and matrix multiplication. *This problem requires more reading than thinking. However it highlights an interesting relationship.*

For this problem you need to review elementary graph theory concepts such as the adjacency matrix representation of graphs (section 22.1 of CLRS) and potentially depth-first-search (22.3). In addition we describe the problem of transitive closure and its relationship to matrix multiplication and all-pairs shortest path. The all-pairst shortest path algorithm by Floyd-Warshall is of interest (section 25.2 of CLRS) for the latter and also former problems.

For a graph $G = (V, E)$ represented through its adjacency matrix A (i.e. $A(i, j) = 1$ if there is an edge from i to j and 0 otherwise) we would like to determine the transitive closure matrix C where $C(i, j) = 1$ if there is a path from i to j and 0 otherwise. Let n be the number of vertices and m the number of edges of G. For general graphs it is obviously $m = O(n^2)$. Naturally, if there is an edge from i to j there is also a path. We assume our graph G is simple (i.e. no self-loops from i to i and no multiple edges). A related problem is the all pairs-shortest path problem. There, we use in addtion to A a distance matrix D where for $A(i, j) = 1$ we show the distance between i and j through $D(i, j)$. In the all-pairs shortest path problem we want to find for all i and j the path of the lowest cost from i to j that minimizes the sum of the distances for the edges in the path.

One easy way to solve the transitive closure problem is to reduce it to the all-pairs problem (see section 25.2) and thus a solution for the latter gives a solution to the former. This involves setting the D matrix to be A (plus some additional details in dealing with diagonal elements) i.e. the distance of any edge is equal to 1. This shows that $C_t(n) = O(D_t(n))$ i.e. the running time for transitive closure is big-Oh of the all-pairs shortest path problem running time.

Another way to solve the transitive closure problem is to perform n depth-first-search operations one per starting vertex of the graph. This would take $O(n(m + n))$ since one depth-first-search requires $O(n + m)$ time. Thus $C_t(n) = O(mn + n^2)$. Given that $m = O(n^2)$ this would imply a $C_t(n) = O(n^3)$. A cubic running-time algorithm begs to be associated with matrix multiplication!

(a) For a graph $G = (V, E)$ with adjacency matrix A, let $X = 1 + A + \ldots + A^k + \ldots + A^{n-1}$, where n is the number of vertices. Show that $C = X$, in other words a non-zero $X(i, j)$ entry indicates a path from i to j in G i.e. $C(i, j) = 1$ when interpreting a $+$ as `OR` and and $\cdot$ as `AND`.
Hint. Use induction and explain A^k.

(b) Show that $M(n) = O(C_t(n))$ where $C_t(n)$ is the time to compute $X = 1 + Z + \ldots + Z^{n-1}$ for abitrary Z (Z can be an adjacency matrix A or just any matrix). As a reminder, $M(n)$ is the time to multiply two $n \times n$ matrices.

Hint. Modify the D matrix of page 18 of Subject 4.

Conclusion (no proof needed). Then if one could also show that $C_t(n)$ is $O(M(n))$ time, we would show that transitive closure is as hard as matrix multiplication. (Note that the expression for X has not one but $n - 2$ matrix multiplication products plus sums.)

Problem 8. (75 POINTS)
Strassen vs Cannon's method.

```
// n can be any integer dimension ; i.e. you have to take care and make it
//   a power of two if necessary
// *A, *B, *C are one dimensional arrays of n*n elements (double)
// A[j*n+i] is the i-th row and j-th column element of a two dimensional array
// For Java use one dimensional arrays
Strassen(double *A, double *B, double *C, int n)  //Does Strassen for arbitrary n
ReadMatrix(double **A,int *n, file input-file); //Allocates space for A and reads A
SetMatrix(double *A,double *B,int n); //Allocates space for A and reads A
PrintMatrix(double *A,int n,  file output-file,)//Prints A into file output-file
Cannon(double *A, double *B, double *C, int n);
```

You need to implement the following interface

```
% ./strassen  input-A input-B output-C
or
% java strassen input-A input-B output-C
% ./cannon    input-A input-B output-C
or
% java cannon   input-A input-B output-C
```

where `input-A, input-B` are files containing input matrices A, B and `output-C` is the file that will contain the output $A \times B$ of Strassen's method or the standard $O(n^3)$ Cannon's method. All three files have the same format. The first line contains the dimension n in the form of an integer. Subsequent lines contain in the form of doubles the input elements in row-major format. That is the first $n = 5$ values `1.0 2.0 3.0 4.0 5.0` are the elements of the first row of the 5×5 array. The next 5 values `6.0 7.0 8.0 9.0 and 10.0` are the elements of the second row and so on. Files `input-A, input-B` are read through `ReadMatrix` and file `output-C` is written by `PrintMatrix`. `SetMatrix` allows one to set copy B into A internally.

```
5
1.0 2.0 3.0 4.0 5.0
6.0 7.0
     8.0 9.0 10.0
1.0
  2.0 3.0 4.0 5.0
1.0
  2.0 3.0 4.0 5.0
1.0 2.0 3.0 4.0 5.0
```

Note. The input array(s) can be of dimension say 17×17. After reading such matrices it's up to you to decide how to store such a matrix; Strassen (textbook description) can only deal with 16×16 or 32×32 matrices but not a 17×17 one. When you print the results into `output-C` make sure it is that of a 17×17 matrix and not that of a matrix of some other dimension. For other assumptions, deviations or instructions, provide a `readme.txt` file with your code.
Benchmarking. Time the two method for your choice of $n = 64$, $n = 256$, and $n = 1024$ non-zero non-trivial matrices. Indicate corresponding running times. For Cannon's method also show a MegaFlop rate ($n \times n$ matrix multiplications requires $n^3 + n^2(n-1)$ operations). Strassen should be able to beat Cannon's method for at least one of those cases on your and AFS machines! Or find that n... So 25 of the 75 points will be for benchmarking.

Problem 9. (75 POINTS)
Schur decompsition. Implement the algorithm for inversion based on Schur decomposition (Subject 4, pages 14-17). Your algorithm should work with any dimension n input matrix A. Adjustments to the dimension should be internal; if the input is of dimension n so should the output even if internally you are using a dimension higher than n. The three functions of the previous problem `ReadMatrix, SetMatrix, PrintMatrix` can/must be reused for I/O. In order to avoid (and be able to deal with) problems with singularities you may wish to read the last page of the Subject 4 notes.

```
// n can be any integer dimension ; i.e. you have to take care and make it
//   a power of two if necessary
// *A, *B, *C are one dimensional arrays of n*n elements (doubles)
// A[j*n+i] is the i-th row and j-th column element of a two dimensional array
// For Java use one dimensional arrays
RecursiveInverse(double *A, double *B, int n); //Find B=A**-1 Inverse per Sub 4
MatrixMultiply(double *A, double *B, double *C, int n);  // C= A*B
ReadMatrix(double **A,int n, file input-file); // Read A from file
SetMatrix(double *A,double *B,int n); //Allocates space for A and reads A
PrintMatrix(double *A,int n,  file output-file,)//Prints A into file output-file

A matrix of the following form might be used as input for testing purposes.
Test Input Matrix ( double *mat )
  for(j=0;j<n;j++)
    for(i=0;i<n;i++) {
       if (i>j) mat[j*n+i] = (double) 0.5*i+1.0;
           else mat[j*n+i] = (double) 0.5*j+0.5;
  }
```

You need to implement the following interface

```
% ./reinverse input-A output-B
or
% java reinverse input-A output-B
```

where `input-A, output-B` are files containing input/output matrices A, B. All have the same format. For other assumptions, deviations or instructions, provide a `readme.txt` file with your code; none of the assumptions/deviations however should restrict the generality of the problem.

Testing matrices that you might decide to use include the following.

- $a_{ii} = 1$ if $i \neq n$, $a_{ij} = 0$ if $i < j < n$, and $a_{ij} = (-1)^{i+j-1}$ if $i > j$ or $j = n$.
- $b_{ij} = |i - j|$.
- $c_{ij} = n - |i - j|$.
- $d_{ij} = \max\{i, j\}|$.
- $f_{ii} = 1$ and other elements are $f_{ij} = 1/n$.

Problem 10. (50 POINTS)
Perfect Power. Enough said. This is a modification to the HW 1 Problem that deals with arbitrarily long integers.

```
% ./perfp  file
% java perfp file
```

If `file` contains the following integer

```
14151641115259968036378199863952645303374293370109242945177798687073
09443839711552006978906009633265377011668990199308081848697977234671
43213860286156570348801141697422485860858847822591621804132329644783
00407295686404276394866098293063134760729404405601927008773432441997
88534237850211370208768763290393557983592179408140523125410988337853
80097907455483636864218318356926828453029537544262212645444988647883
85813177353964166540078477942388388712534567783527565431329210003322
61374243654095244223902210441127804237930076592985940546359286593961
58366245591095584669988187465399291657102034423728880499566254670015
28219413777247740076088840171803589799673112512388393396582206313769
99160592950726013280271480477098006491558397072511078019010077180843
50874580310171166086851885248699174293623186425272254134399399608583
66520443426888955167725548732899527174857936822356517180542189659837
39152480949563825364351809995885714072719448823085105048348296787934
23673795228374420826559012471812162095800905929851021460290655293142
67800579455420301098442374463860550639112324752348977039032989100217
95725036770057907661032175153648822976183883974753760682891488279937
24268169405805664080039389733008589582542622750410299896586236878864
981788429782089728
```

the result could be (there are other answers as well)

```
Perfect Power!   x= 12321312312  y= 123
```

Otherwise a

```
Not a Perfect Power!
```

will be printed. The integer stored in the file would contain no more than 10000 radix-10 digits. (You read digits until there are no more to the end of file, skipping non-digits to deal with `file`.)