

Feb 07, 08 10:47

as1-2008-sol.txt

Page 1/4

A sample C++ program for AS1

```

***** file main.cc *****

#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include "as1-2008-monge.h"

int main(int argc, char **argv) {

    if(argc != 3) { // quit if 2 args aren't passed
        cout << "Usage: as1 <inputfilename> <outputfilename>\n";
        return 0;
    }

    int m=0, n=0;
    const int SIZE = 100;
    char buffer[SIZE];

    ifstream infile(argv[1], ios::in); //open files
    ofstream outfile(argv[2], ios::app);

    if(!infile) {
        cerr << argv[1] << " could not be opened...aborting\n";
        exit(1);
    }

    if(!outfile) {
        cerr << argv[2] << " could not be opened...aborting\n";
        exit(1);
    }

    infile >> m >> n ; // get the size of the m x n matrix

    infile.get();
    Monge matrix(m,n);

    int row = 1;
    while(!infile.eof()) {
        infile.getline(buffer, SIZE);
        // input each line of numbers into each row of matrix
        matrix.Add(buffer, row);
        row++;
    }

    outfile << endl << argv[1] << endl;

    // If the matrix is NOT Monge, The program should end here.
    if(matrix.Is_Monge(outfile)) {
        matrix.comp = 0; // reset the comparison counter
        matrix.Recursive(outfile);
        outfile << "Faster algorithm used " << matrix.comp << " comparisons\n";

        matrix.comp = 0; // reset the comparison counter
        matrix.Straightforward1();
        outfile << "Very Straightforward algorithm used " << matrix.comp << " compar
isons\n";

        matrix.comp = 0; // reset the comparison counter
        matrix.Straightforward2();
        outfile << "Straightforward algorithm used " << matrix.comp << " comparisons
\n\n";
    } else
        cout << "\nProgram aborted here since the array was not Monge.\n\n";
}

```

Feb 07, 08 10:47

as1-2008-sol.txt

Page 2/4

```

infile.close();
outfile.close();
return 0;
}

***** file monge.cc *****

#include <iostream.h>
#include <fstream.h>
#include <cstdlib>
#include <cstring>
#include "as1-2008-monge.h"

// constructor, set array to initial empty state
Monge::Monge(int M, int N) {

    m = M;
    n = N;
    comp = 0;

    f = new int[m+1];

    // Create an (m+1)x(n+1) matrix
    // Use 1-based instead of 0-based for indexing
    array = new double*[m+1];
    for(int i=0; i<=m; i++)
        array[i] = new double[n+1];

    //initialize the matrix and the min array to all zeros

    for(int i=0; i<=m; i++) {
        for(int j=0; j<=n; j++)
            array[i][j] = 0.0;
        f[i] = 0;
    }
    f[0] = 1;
}

// Add a row of number into the matrix
void Monge::Add(char* buffer, int row) {

    char *tokenPtr;

    tokenPtr = strtok(buffer, " ");
    int col = 1;

    // tokenize the buffer line into words
    while(tokenPtr != NULL) {
        // Converts the string to type 'double' and store in array
        array[row][col] = atof(tokenPtr);
        tokenPtr = strtok(NULL, " ");
        col++;
    }
}

// Check to see if the given matrix is "Monge"
// Return 1 if it is Monge, otherwise return 0
int Monge::Is_Monge(ofstream &outfile) {

    for(int i=1; i<=m; i++)
        for(int j=1; j<=n; j++) {
            // check the "Monge condition"
            if(array[i][j]+array[i+1][j+1] > array[i][j+1]+array[i+1][j]) {
                outfile << "No. This is not a Monge array\n";
                return 0;
            }
        }
}

```

Feb 07, 08 10:47

as1-2008-sol.txt

Page 3/4

```

    }
    outfile << "Yes. This is a Monge array\n";
    return 1;
}

// Find the leftmost minimum INDEX for a given row.
// Sweep through the row starting from 'begin' to 'end' column
int Monge::find_min(int row, int begin, int end) {
    int minindex = begin;
    for(int i=begin+1; i<=end; i++) {
        comp++; // increment the comparison counter
        if(array[row][i] < array[row][minindex])
            minindex = i;
    }
    // cout << "ROW: " << row << " MIN = " << minindex
    // << " COMP = " << comp << endl;
    return minindex; // return index
}

// The most straightforward alg. to find the min element
// in each row. Always starts from the first col and ends
// at the last col
void Monge::Straightforward1() {
    int min[m+1]; // an array that keeps index of min elements in each row

    for(int i=1; i<=m; i++)
        min[i] = find_min(i,1,n);
}

// Another version of straightforward alg. to find the min element
// in each row. It starts from the index found in the previous row,
// and ends at the last col.
void Monge::Straightforward2() {
    int min[m+1];

    min[1] = find_min(1,1,n); // find minindex for the first row
    for(int i=2; i<=m; i++)
        min[i] = find_min(i, min[i-1], n);
}

// Recursively find the minindex for even- and odd-numbered rows
void Monge::RecursiveMin(int size, int factor, int odd) {
    if(size == 1)
        f[factor] = find_min(factor, f[0], n); // base case
    else {
        // the size of matrix is reduced in half every time
        // the RecursiveMin gets called
        RecursiveMin(size/2, 2*factor, (size/2)%2);

        // Find minindex in the even-numbered rows first, starting from the minindex
        // from the previous even row and ending at the minindex from the next
        // even row. Then find the minindex in the odd-numbered rows according to t
he
        // current number of rows in each iteration.
        if(!odd) { // if the size is even
            for(int i=1; i<=size; i+=2) {
                f[i*factor] = find_min(i*factor, f[(i-1)*factor], f[(i+1)*factor]);
            }
        } else { // if the size is odd
            for(int i=1; i<=(size-1); i+=2)
                f[i*factor] = find_min(i*factor, f[(i-1)*factor], f[(i+1)*factor]);
            f[size*factor] = find_min(size*factor, f[(size-1)*factor], n);
        }
    }
}

```

Feb 07, 08 10:47

as1-2008-sol.txt

Page 4/4

```

    }
}

// A faster alg to find the minimum elements
void Monge::Recursive(ofstream &outfile) {
    RecursiveMin(m, 1, m%2); // find f[i] for even- and odd-numbered rows

    // print the leftmost minimum elements into output file
    for(int i=1; i<=m; i++)
        outfile << array[i][f[i]] << "\n";

    outfile << endl;
}

***** file monge.h *****

class Monge{
public:
    Monge();
    Monge(int m, int n); // constructor

    double **array; // matrix (a 2-D array)
    void Add(char*, int); // add integers to array row by row

    int Is_Monge(ofstream &); // check to see if the matrix is Monge
    void Straightforward1(void); // find min in each row. (very straightforward)
    void Straightforward2(void); // another version of finding min
    int find_min(int, int, int); // Finding the index of the minimum element,
    // given the row number, start, and end col.
    void Recursive(ofstream &); // A faster alg. using recursion to find min.
    void RecursiveMin(int, int, int); // Recursively find the min in
    // even-numbered, then odd-numbered rows
    int comp; // Comparison counter of the matrix elements.

    int m; // size of m x n matrix
    int n;
    int *f; // f[i] keeps the index of the column
    // containing the leftmost minimum element
    // in row i
};

```