**Quiz 4 Group D**

**Programming:**

**10. Design an algorithm to check is a point is inside the triangle or not.**

// C++

class Point {

public:

double x;

double y;

Point(double a, double b):x(a),y(b){}

};

class Triangle {

public:

Point\* A;

Point\* B;

Point\* C;

double area(){

return abs(A->x \* (B->y - C->y) + B->x \* (C->y - A->y) + C->x \* (A->y - B->y))/2.0;

}

Triangle(Point\* one, Point\* two, Point\* three):A(one),B(two),C(three){}

};

class Solution {

public:

// idea: assume the point is P, then the area ABC = PAB + PBC + PCA

bool solve (Point\* P, Triangle\* T){

Triangle\* PAB = new Triangle(P,T->A,T->B);

Triangle\* PBC = new Triangle(P,T->B,T->C);

Triangle\* PCA = new Triangle(P,T->C,T->A);

return T->area() == PAB->area() + PBC->area() + PCA->area();

}

};

**11. Design an algorithm to check palindrome sequence.**

// not sure about the requirement: check if the string array is palindromic?

// C++

class Solution {

public:

bool isPalindrome(string input){

string reversed = reverse(input);

for (int i = 0; i < 1 + input.size()/2; ++i) {

if (reversed[i] != input[i]) {

return false;

}

}

return true;

}

string reverse(string input) {

int left = 0;

int right = input.size() - 1;

reverseHelper(input, left, right);

return input;

}

void reverseHelper(string& input, int left, int right) {

if (input.size() <= 1) {

return;

}

int i = left;

int j = right;

while(i < j) {

char temp = input.at(i);

input.at(i) = input.at(j);

input.at(j) = temp;

++i;

--j;

}

}

};

**12. Given an array, design an algorithm to return the longest decreasing sub-array.**

// C++

class Solution {

public:

// assume the array is not null

vector<int> solve(vector<int> input) {

int length = 1;

int maxLength = 1;

int idx\_end = 0;

vector<int> result;

for (int i = 1; i < input.size(); ++i) {

if (input[i] < input[i - 1]) {

length = length + 1;

if (length > maxLength) {

maxLength = length;

idx\_end = i;

}

}

else {

length = 1;

}

}

for (int j = idx\_end - maxLength + 1; j < idx\_end + 1; ++j) {

result.push\_back(input[j]);

}

return result;

}

};

**13. Implement the Sudoku algorithm.**

Reference: http://yucoding.blogspot.com/2013/12/leetcode-question-sudoku-solver.html

// C++

class Solution {

public:

bool find(vector<vector<char> > &cur, int &i, int &j){

for(int ii = 0; ii < 9; ii++){

for(int jj = 0; jj < 9; jj++){

if(cur[ii][jj] == '.'){

i = ii;

j = jj;

return true;

}

}

}

return false;

}

set<char> valid(int i, int j, vector<vector<char> > &cur){

set<char> se;

se.insert('1');

se.insert('2');

se.insert('3');

se.insert('4');

se.insert('5');

se.insert('6');

se.insert('7');

se.insert('8');

se.insert('9');

for (int ii=0;ii<9;ii++){

se.erase(cur[ii][j]);

se.erase(cur[i][ii]);

}

for (int ii=0;ii<3;ii++){

for (int jj=0;jj<3;jj++){

se.erase(cur[(i/3)\*3+ii][(j/3)\*3+jj]);

}

}

return se;

}

void solveSudoku(vector<vector<char> > &board) {

queue<vector<vector<char> > > que;

que.push(board);

vector<vector<char> > cur;

set<char> se;

int i=0;

int j=0;

while (!que.empty()){

cur = que.front();

que.pop();

if (find(cur,i,j)==false){

board = cur;

return;

}else{

se = valid(i,j,cur);

for(set<char>::iterator it = se.begin(); it != se.end(); ++it){

cur[i][j]= \*it;

que.push(cur);

}

}

}

}

};

**14. Consider a chess board with each cell has a value on it, you can only walk right or down, find the path from up-left to down-right which has largest value.**

// C++

class Solution {

public:

int solve(vector<vector<int> > input){

int n = input.size();

vector<vector<int> > M (n, vector<int> (n));

M[0][0] = input[0][0];

for(int i = 1; i < n; ++i){

M[0][i] = M[0][i - 1] + input[0][i];

M[i][0] = M[i - 1][0] + input[i][0];

}

for(int i = 1; i < n; ++i) {

for (int j = 1; j < n; ++j) {

M[i][j] = max(M[i - 1][j], M[i][j - 1]) + input[i][j];

}

}

return M[n - 1][n - 1];

}

};