Pontificia Universidad Católica Madre y Maestra
Facultad de Ciencias de la Ingeniería

Tarea #2:

2 Sat Problem

Presentado Por:

Angel González (2013-0157)

Materia:

Diseno y Analisis de Algoritmos

Profesor:

Juan Núñez.

Marco Teorico

En informática, 2-satisfacibilidad (abreviado como 2-SAT o simplemente 2SAT) es el problema de determinar si un conjunto de dos valores, variables con restricciones en los pares de variables (binarios de Boole o) puede ser los valores que satisfacen todas las restricciones asignado. Es un caso especial del problema de Boole satisfacibilidad general, lo que puede implicar limitaciones en más de dos variables, y de los problemas de satisfacción de restricciones, que pueden permitir más de dos opciones para el valor de cada variable. Pero en contraste con los problemas más generales, que son NP-completa, 2-satisfacibilidad puede ser resuelto en tiempo polinómico.

Instancias del problema 2-satisfacibilidad se expresan típicamente como 2-CNF, también conocido como fórmulas Krom. Una representación alternativa es la gráfica implicación, que expresa las variables de una instancia y sus negaciones como vértices en un grafo, y las limitaciones de los pares de variables como bordes dirigidos. Un ejemplo puede ser resuelto en tiempo polinómico por la resolución. soluciones de tiempo lineal incluyen un método basado en marcha atrás y otro método basado en los componentes fuertemente conectados de la gráfica implicación.

2-satisfacibilidad se puede aplicar a la geometría de visualización y problemas en los que una colección de objetos tienen dos posibles ubicaciones y el objetivo es encontrar una colocación para cada objeto que evite solapamientos con otros objetos. Otras aplicaciones incluyen datos de agrupación para minimizar la suma de los diámetros de los grupos, la clase y deportes de programación, y las formas que se recuperan de información acerca de sus secciones transversales.

Codigo

```
#include <iostream>
#include <vector>
#include <stack>
#include <queue>
#include <algorithm>
#include <map>
#include <cmath>
#include <set>
#include <deque>
#include <cstdio>
#include <cstring>
#include <iomanip>
#include <sys/resource.h>

#define pb push_back
#define mp make_pair

using namespace std;

typedef long long ll;
typedef pair<int, int> pi;
typedef pair<long long, long long> pll;

const int MOD = 1e9 + 7;

map<int, int> component;
map<int, vector<int> > edges;
map<int, vector<int> > rev_edges;
map<int, int> seen;
int counter = 0;
stack<int> S;

int x[2000005];
int y[2000005];

void dfsOne(int node){
```

```
	seen[node] = true;

	vector<int> cur = edges[node];

	for( int i = 0; i < cur.size(); i++){
		int to = cur[i];

		if( seen.find(to) == seen.end() ){
			dfsOne(to);
		}
	}

	S.push(node);
}

int seens = 0;

void dfsSecond(int node){

	seens++;

	seen[node] = true;

	vector<int> cur = rev_edges[node];

	for(int i = 0; i < cur.size(); i++){
		int to = cur[i];

		if( seen.find(to) == seen.end())
			dfsSecond(node);
	}

	component[node] = counter;

}
int main(){


	int n; cin >> n;

	for(int i = 0; i < n; i++){
		cin >> x[i] >> y[i];
		edges[-x[i]].push_back(y[i]);
		rev_edges[y[i]].push_back(-x[i]);
	}
```

```
	for(int i = 0; i < n; i++){

		if( seen.find( x[i] ) == seen.end() )
			dfsOne( x[i] );
		if( seen.find( -y[i] ) == seen.end() )
			dfsOne( -y[i] );
		if( seen.find( y[i] ) == seen.end() )
			dfsOne( y[i] );
		if( seen.find( -x[i] ) == seen.end() )
			dfsOne( -x[i] );
	}

	seen.clear();

	counter = 0;

	while( !S.empty() ){
		int v = S.top();
		S.pop();

		if( seen.find(v) == seen.end() ){
			counter++;
			dfsSecond(v);
		}
	}


	for(int i = 0; i < n; i++){

		if( component[ x[i] ] == component[ -x[i] ] ){
			cout << "NO ES SATISFACIBLE" << endl;
			return 0;
		}

		if( component[ y[i] ] == component[ -y[i] ] ){
			cout << "NO ES SATISFACIBLE" << endl;
			return 0;
		}
	}

	cout << "ES SATISFACIBLE" << endl;

	return 0;
}
```

Corrida

La corrida fue un poco problematica para el Algoritmo de Kosaraju que fue implemetado. Puesto que utiliza una gran cantidad de memoria del stack, y puede ser problematico para la computadora. La ventaja es que utilizar Kosaraju, nos permite tener una solucion en tiempo lineal.

Angels-Mac-mini:CP angelgonzalez$ ./T3 < data.txt
ES SATISFACIBLE