# Problema del dominó

Mikel Dalmau

11 de Mayo de 2015

## 1 Enunciado del problema

El juego del dominó tiene 28 fichas diferentes. Cada ficha tiene un par de números enteros entre el 0 y el 6. Según las reglas del juego, las fichas se colocan formando una cadena, de manera que fichas consecutivas tienen el mismo número en los extremos que se tocan. Un ejemplo de cadena es el siguiente: (3, 5),(5, 1),(1, 3),(3, 0). Escribe un programa que, dado un conjunto de fichas de dominó, genere una cadena "respetando las reglas del juego" que utilice todas las fichas del conjunto; o bien determine que es imposible generar una cadena con ese conjunto de fichas.

## 2 Resolución

Modelamos el problema del domino como un problema de grafos, los vértices son los números de las fichas y habrá de 0 a 6, las aristas serán las fichas que tengamos, no tendrán dirección definida ni peso.

Para resolver el problema, en primer lugar hay que determinar si el grafo en cuestión contiene ciclos o caminos Eulerianos, esto es, es posible recorrer todas las aristas(fichas) sin repetir ninguna, para esto es condición necesaria en primer lugar que el grafo sea conexo, por otra parte hay que atender a la paridad del grado de los vértices(número de aristas incidentes) ya que como demostró Leonard Euler para el problema de los puentes de konigsberg si existen más de dos vértices de grado impar entonces es imposible realizar este tipo de camino.

Por último cuando el problema cumple las condiciones anteriores he utilizado el algoritmo que se conoce como algoritmo de Fleury para hallar el camino. Para encontrar el camino utilizamos una lista que indica que aristas han sido visitadas ya, comenzamos desde un nodo cualquiera y agregamos una arista si no ha sido ya utilizada, si está conectada al vértice en el que nos encontramos, y si no es un puente(eliminar dicha arista hace que el grafo sea disconexo). Una vez que se cumplan estas condiciones marcamos la arista como usada, decrementamos la matriz de adyacencia, y el grado de los vértices que se conectaban y avanzamos al vértice que nos llevaba la arista recién eliminada.

Año: 2014 - 2015
Profesor(a): Jesus Bermudez De Andres
Teléfono:
E-mail: mdalmau002@ikasle.ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

# 3 Código del Programa

Las siguientes páginas contienen todo el código utilizado, en lenguaje c++.

```
#include <vector>
#include <cstdlib>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <cmath>
#include <time.h>
#include <string.h>

using namespace std;

typedef struct{
    int nodo1;
    int nodo2;
}arista;

typedef struct {
    int** aristas;
    vector<int> gradoNodos;
    vector<arista> vAristas;
    int N;
    int A;
}grafo;

//Realiza un recorrido en profundidad desde el nodo parametro
void visita(grafo *G,vector<bool> *pV, int nodo){
    pV->at(nodo)=true;
    for(int i= 0; i<7; i++){
        if(G->aristas[nodo][i] != 0 and pV->at(i)==false){
            visita(G,pV,i);
        }
    }
}
//Detemina si un grafo es conexo y responde verdadero o falso
bool conexo(grafo *G){
    vector<bool> visitados(7);
    int i = 0; bool found = false;
    while(i < 7 and !found){
        if(G->gradoNodos[i] != 0){
            found = true;
        }
        i++;
    }
    visita(G,&visitados,i);
    for(i=0; i<7; i++){
        if(G->gradoNodos[i] != 0 and visitados[i]==false){
            return false;
        }
    }
    return true;
}
//Cuenta el grado de cada vertice
int contarGradoVertices(grafo *G){
    int i,j, grado, impares;
    impares = 0;
    for(i=0; i<G->N; i++){
        grado = 0;
        for(j=0;j<G->N;j++){
            if(G->aristas[i][j] == 1){
                grado++;
                if(i == j)
                    grado++;
```

```
                }
            }
            G->gradoNodos[i]=grado;
            if(grado % 2 != 0){
                impares++;
            }
        }
    return impares;
}
void inicializarGrafo(grafo *G){
    int** aristas;
    aristas = new int* [7];
    vector<int> nodos(7);
    for (int j = 0; j < 7; j++){
        nodos[j]=0;
        aristas[j] = new int [7];
        for(int h=0; h< 7 ; h++)
            aristas[j][h] = 0;
    }
    G->gradoNodos = nodos;
    G->aristas=aristas;
    G->N = 7;
    G->A = 0;
}
void imprimirMatriz(grafo *G){
    for(int i = 0; i < G->N; i++){
        for(int j = 0; j < G->N; j++){
            cout << G->aristas[i][j] <<  "\t";
        }
        cout << endl;
    }
};
grafo leerFichero(string pFname){
    int i=0,A,nodo1,nodo2,j; arista a;
    grafo G;
    inicializarGrafo(&G);
    ifstream fe(pFname.c_str());
    //numero de fichas de domino
    fe >> A;
    G.A = A;
    G.vAristas = vector<arista>(A);
    for(i = 0; i < A ; i++){
        fe >> nodo1;
        fe >> nodo2;
        G.aristas[nodo1][nodo2]++;
        a.nodo1=nodo1; a.nodo2=nodo2;
        G.vAristas[i]=a;
        if(nodo1 != nodo2)
            G.aristas[nodo2][nodo1]++;
    }
    fe.close();
    return G;
};
//Elimina una arista del grafo y comprueba si hace que el grafo resultante no sea conexo
bool puente(grafo pG, int nodo1, int nodo2){
    grafo G=pG;
    G.aristas[nodo1][nodo2] = 0;
    G.aristas[nodo2][nodo1] = 0;
    G.gradoNodos[nodo1]--;
    G.gradoNodos[nodo2]--;
    if(conexo(&G)){ return false; }
    else{ return true;}
}
void camino_Euleriano(grafo *G){
    vector<bool> aristasVisitadas(G->A);
    std::fill(aristasVisitadas.begin(),aristasVisitadas.end(),false);
```

```
    //Buscamos un nodo desde el que empezar a recorrer el grafo
    int nodoActual=0; bool found=false;
    while(nodoActual < 7 and !found){
        if(G->gradoNodos[nodoActual] != 0){
            found = true;
        } nodoActual++;
    }
    for(int i=0; i<G->A; i++)
        for(int j=0; j<G->A; j++)
            if(!aristasVisitadas[j] and (G->vAristas[j].nodo1 == nodoActual or G->vAristas[j].
                nodo2 == nodoActual) and !puente(*G,G->vAristas[j].nodo1,G->vAristas[j].nodo2)
                ){
                aristasVisitadas[j]=true;
                //cout << "vuelta"<<i<<j<<endl;
                G->aristas[G->vAristas[j].nodo1][G->vAristas[j].nodo2]--;
                G->aristas[G->vAristas[j].nodo2][G->vAristas[j].nodo1]--;
                G->gradoNodos[nodoActual]--;
                if(G->vAristas[j].nodo1 == G->vAristas[j].nodo2){
                    G->gradoNodos[nodoActual]--;
                }else{
                    if(G->vAristas[j].nodo1 == nodoActual){
                        nodoActual= G->vAristas[j].nodo2;
                    }else{
                        nodoActual= G->vAristas[j].nodo1;
                    }
                }
                cout<<nodoActual<<" - ";//Escribimos los nodos visitados en la pantalla
            }
    cout << " fin de busqueda"<<endl;
}
void domino(string fname){
    grafo G;
    G = leerFichero(fname);
    //imprimirMatriz(&G);

    int impares = contarGradoVertices(&G);
    if(!conexo(&G)){cout << "El grafo no es conexo no, no existen caminos eulerianos"<< endl;
        return;}
    else{ cout << "El grafo es conexo."<< endl;}

    if(impares == 2){cout << "El grafo tiene: " << impares << " vertices de grado impar, posee
         un ciclo euleriano."<< endl;
        camino_Euleriano(&G);}
    else if( impares == 0 ){cout << "El grafo tiene: " << impares << " vertices de grado impar
        , posee un camino euleriano."<< endl;
        camino_Euleriano(&G);}
    else{cout << "El grafo tiene: " << impares  << " vertices de grado impar, no contiene
        ciclos ni caminos eulerianos."<< endl; return;}
}
int main(int argc, char** argv) {
    domino("JuegoCompleto");
    domino("JuegoIncompleto");
    domino("JuegoInsatisfactible");
    return 0;
}
```

# 4 Pruebas

**JuegoCompleto** .

28
0 0
0 1
0 2
0 3
0 4
0 5
0 6
1 1
1 2
1 3
1 4
1 5
1 6
2 2
2 3
2 4
2 5
2 6
3 3
3 4
3 5
3 6
4 4
4 5
4 6
5 5
5 6
6 6

**JuegoIncompleto** .
11
0 1
0 2
1 2
1 3
1 5
2 3
2 4
3 6
4 4
4 5
5 6

**JuegoInsatisfactible** .
11
0 1
1 1
1 2
1 3
2 2
2 3
2 4
3 5
4 4
4 5
6 6

## Bibliografía

[cppRef] http://es.cppreference.com .

[PGui] http://pier.guillen.com.mx/algorithms/10-graficas/10.7-camino_euleriano.htm

[Das] Algorithms, S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani.