Grafos bipartitos

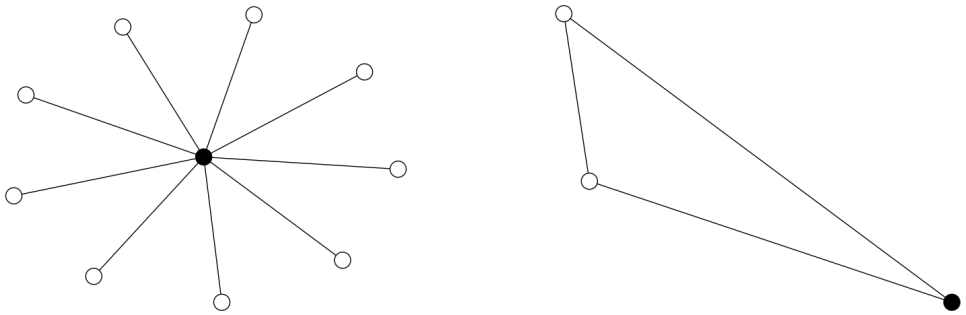
Jose Antonio Gallardo Monroy, Alan Misael Castañón López
Vanesa Ávalos Gaytan

Primeramente se debe definir lo que es un grafo bipartito, se darán ejemplos y se verá la solución de algunos problemas. Un grafo se denotará como $G(N, E)$ donde N es el conjunto de nodos y E las aristas que los unen. Cada arista tendrá la forma (i, j), dicha arista une al nodo i y j. Los grafos tratados son bidireccionales, si existe la arista (i, j) se puede ir de i a j y de j a i.

Definición

Un grafo $G(N, E)$ se dice que es bipartito si existe una partición $P = \{A_1, A_2\}$ de N tal que si $a, b \in A_i$, $(a, b) \notin E$.

Es fácil ver que la extensión del concepto es no vacía y que no todo grafo es bipartito. Se utilizará frecuentemente la idea de coloreo, todos los nodos coloreados del mismo color pertenecerán al mismo A_i. También se dará por hecho que al hablar de A_i, $i = 1, 2$.



Se puede apreciar que el grafo de la izquierda es bipartito; haciendo A_1 el conjunto que contiene a los nodos blancos y A_2 el que contiene al nodo negro, se tiene una partición de N y no hay aristas entre nodos que pertenecen al mismo A_i. Si se encuentran los A_i con dichas características el grafo es bipartito, pero dado un grafo ¿cómo saber si es bipartito o no? por ejemplo el de la derecha; si suponemos que es bipartito un nodo inicial se puede colorear de cualquier color (blanco o negro), concluyendo que todos sus vecinos tienen que ser del color contrario, en este caso hay un conflicto al tener dos nodos blancos unidos por un arista, así que este grafo no es bipartito.

Verificar si un grafo es bipartito

Supongamos que el grafo es bipartito, al tomar u un nodo cualquiera del grafo, S.P.G. $u \in A_1$, así que todos sus vecinos deben estar en A_2. Una vez identificado v como vecino de u, se tiene que $v \in A_2$, por ello todos los vecinos de v deben estar en A_1. En este punto se tendrán nodos coloreados y otros sin colorear, pero usando los datos generados anteriormente habría que ver si se pueden colorear nodos que estan pendientes usando la misma estrategia, si ya no se pueden colorear nodos entonces el grafo no es conexo y para seguir con el chequeo se haría lo mismo con un nuevo nodo incoloro. Si en este proceso se intenta colorear un nodo que ya estaba pintado con el color contrario, el grafo no es bipartito, pero si el proceso termina sin problemas se tendrá una partición válida. Los algoritmos dfs y bfs son ideales para verificar si un grafo es bipartito, al final se muestran implementaciones en C++.

A continuación se comentan algunos problemas que pueden encontrarse en jueces en linea, el formato será de la forma (juez en linea - ID del problema - nombre del problema). No se presentará el código que da solución a los problemas porque difiere muy poco a los dados en la parte final.

COJ - 2927 - Jorge's Party

Situación: Jorge quiere hacer una fiesta en su casa, después de pensar un poco se dio cuenta que algunas personas hacen grupos con otros que ya conocen y se impiden hacer nuevas amistades, así que Jorge quiere ver si es posible separar a todos sus invitados en 2 grupos de tal manera que ningún par de personas en el mismo grupo se conozca. Si se tiene un número n de personas y se sabe quien conoce a quien, determinar si hacer dicha separación es posible.

Solucion: Como se sabe quien conoce a quien, se puede modelar un grafo donde cada nodo es una persona y si se conocen hay un arista que los une. Cada grupo de personas sería un A_i, si el grafo es bipartito entonces puedo acomodarlos y ningún par de personas en el grupo se conocerá, de lo contrario dicha tarea es imposible.

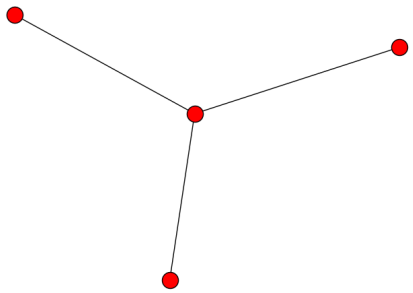
UVa - 11080 - Place the Guards

Situación: En el país de Ajabdesh se tienen calles y cruces entre estas, el rey quiere ubicar guardias en los cruces de tal manera que cada calle sea vigilada, el guardia puede vigilar el cruce y cualquier calle que forme parte de ese cruce. El problema es que los guardias son rudos y si hay dos guardias vigilando la misma calle pelearán entre ellos. Dados las calles y cruces, el rey quiere saber el menor número de guardias necesario para que toda calle y cruce sean vigilados sin que algún guardia peleé con otro.

Solución: Se puede modelar un grafo donde cada nodo es un cruce y las calles son aristas. Dado un cruce se puede colocar un guardia o no, si se coloca un guardia no se puede colocar un guardia en cualquier cruce vecino, porque pelearían; si no se coloca se debe colocar un guardia en todo cruce vecino, de lo contrario las calles quedarán sin protección. Planteado lo anterior, se hace la analogía a colorear cada nodo en blanco o negro dependiendo si pongo un guardia ahí o no, pero no puede haber dos nodos vecinos con el mismo color, justo por lo anterior, así que esto se puede únicamente si el grafo es bipartito. Después de hacer el coloreo se van a tener nodos pintados de negro y blanco, como quiero colocar el menor número de guardias tomo $min\{|A_i|\}$ y lo añado a la cuenta. Es posible que el grafo no sea conexo, de tal manera que la decisión greedy anterior se debe hacer en cada islita, con la salvedad de que alguno de los dos sea 0, es decir, si se tiene 1 nodo blanco y ningún nodo negro, no se toda el mínimo, porque quedaría desprotegida esa parte de la ciudad, así que se toma en cuenta el nodo blanco, adaptado esto a nuestro problema no tiene sentido, pero hay que estar preparados.

UVa - 11396 - Claw Decomposition

Situación: Dado un grafo $G(N, E)$ simple 3-regular, ver si es posible encontrar subgrafos que tengan la forma siguiente:



Además cada subgrafo no puede compartir aristas.

Solución: Aquí la clave esta en ver que hay un centro y debe tener grado 3, de tal manera que si un nodo en el grafo es centro, sus 3 vecinos deben ser extremos de la figura, al ser G un grafo simple asegura que no hay aristas paralelas, por ello lo que es extremo en un subgrafo no puede ser centro en otro (debería tener 3 aristas para tomar como extremos pero una ya fue tomada por el subgrafo inicial, dejando 2 aristas que no dan basto para ser centro), tiene que ser extremo, así que los 3 vecinos de este extremo deben ser centros. Por lo anterior se tiene que el problema se resume a checar si el grafo dado es bipartito o no.

Códigos en C++ para verificar si un grafo es bipartito utilizando dfs y bfs, respectivamente. Ambos son solución del problema UVa - 10004 - Bicoloring.

```cpp
#include <iostream>
#include <vector>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

int n,m,u,v;
vvi G;
vi color;
bool bip;

void dfs(int u)
{
	for(int j=0;bip && j<G[u].size();j++)
	{
		v = G[u][j];

		if(color[v] == 0)
		{
			color[v] = -color[u];
			dfs(v);
		}
		else
		{
			if(color[u] == color[v])
				bip = 0;
		}
	}

}

int main()
{
	cin.tie(0);
	ios::sync_with_stdio(0);

	while(cin >> n && n)
	{
		cin >> m;

		G.clear();
		G.resize(n);
		color.assign(n,0);

		for(int i=0;i<m;i++)
		{
			cin >> u >> v;
			G[u].push_back(v);
			G[v].push_back(u);
		}

		bip = 1;

		for(int i=0;bip && i<n;i++)
			if(!color[i])
			{
				color[i] = 1;
				dfs(i);
			}

		if(bip)
			cout << "BICOLORABLE.\n";
		else
			cout << "NOT BICOLORABLE.\n";

	}

	return 0;
}
```

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

int main()
{
        cin.tie(0);
        ios::sync_with_stdio(0);

        vvi G;
        vi color;
        int n,m,u,v;
        bool bip;

        while(cin >> n && n)
        {
                cin >> m;
                G.clear();
                G.resize(n);
                color.assign(n,0);

                for(int i=0;i<m;i++)
                {
                        cin >> u >> v;
                        G[u].push_back(v);
                        G[v].push_back(u);
                }

                bip = 1;

                for(int i=0;bip && i<n;i++)
                        if(!color[i])
                        {
                                queue<int> q;
                                q.push(i);
                                color[i] = 1;

                                while(!q.empty() && bip)
                                {
                                        u = q.front();
                                        q.pop();

                                        for(int i=0;i<G[u].size();i++)
                                        {
                                                v = G[u][i];

                                                if(!color[v])
                                                {
                                                        color[v] = -color[u];
                                                        q.push(v);
                                                }
                                                else
                                                {
                                                        if(color[u] == color[v])
                                                                bip = 0;
                                                }
                                        }
                                }
                        }

                if(bip)
                        cout << "BICOLORABLE.\n";
                else
                        cout << "NOT BICOLORABLE.\n";
        }


        return 0;
}
```

Referencias

- Competitive Programming - 2da edición - Steven Halim
- https://uva.onlinejudge.org/
- http://coj.uci.cu/index.xhtml