Recorridos sobre grafos

Con una pizca de programación dinámica en árboles

Contexto

- Objetivo: recorrer los distintos vértices de un grafo.

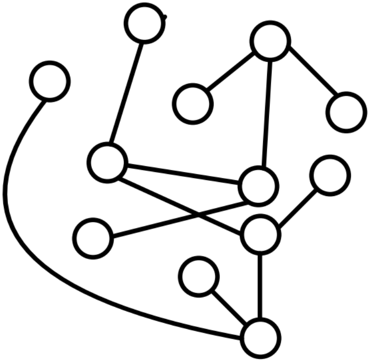
Contexto

- Objetivo: recorrer los distintos vértices de un grafo.
- Las 2 opciones que agarramos, BFS y DFS, generan 2 estructuras interesantes que vale la pena explorar.

DFS (Depth First Search)

Naturalmente se suele escribir recursivo, es el que tiene una estructura más compleja e interesante (0 subjetivo (ponele)).

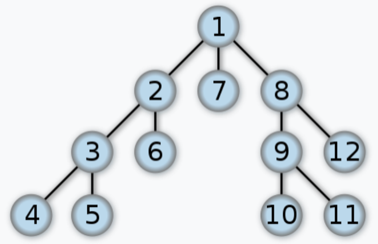
```
vector<vector<int>> aristas = ... ;
vector<bool> visitado(n, false);

void dfs(int v) {
    visitado[v] = true;
    for (int u : aristas[v]) {
        if (!visitado[u])
            dfs(u);
    }
}
```



DFS (Depth First Search)

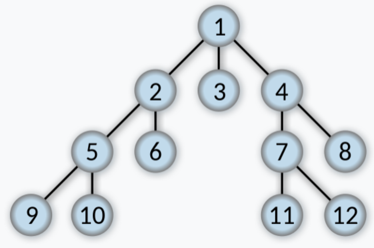
Naturalmente se suele escribir recursivo, es el que tiene una estructura más compleja e interesante (0 subjetivo (ponele)).

```
vector<vector<int>> aristas = ... ;
vector<bool> visitado(n, false);

void dfs(int v) {
    visitado[v] = true;
    for (int u : aristas[v]) {
        if (!visitado[u])
            dfs(u);
    }
}
```



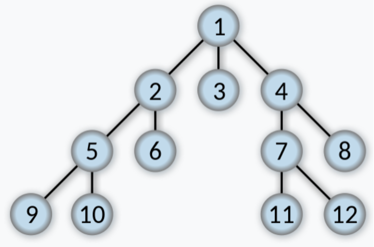
BFS (Breadth First Search)

```
vector<vector<int>> aristas = ... ;

vector<bool> visitado(n, false);
vector<int> distancia(n);

void bfs(int s) {
    visitado[s] = true;
    distancia[s] = 0;
    ...
```



BFS (Breadth First Search)

```
    ...
    queue<int> q;
    q.push(s);

    while (!q.empty()) {
        int v = q.front(); q.pop();
        for (auto u : aristas[v])) {
            if (!visitado[u]) {
                visitado[u] = true;
                distancia[u] = distancia[v] + 1;
                q.push(u);
            }
        }
    }
}
```



Problemas que se resuelven con ambos algoritmos

Chequeo de conectividad: el grafo que tengo es conexo?

Problemas que se resuelven con ambos algoritmos

Chequeo de conectividad: el grafo que tengo es conexo?

```
recorro_a_partir_del_vertice(0);
all_of(visitado.begin(), visitado.end(),
    [](bool fue_visitado) { return fue_visitado; });
```

Problemas que se resuelven con ambos algoritmos

Cómo puedo averiguar cuántas componentes conexas tengo?

Problemas que se resuelven con ambos algoritmos

Cómo puedo averiguar cuántas componentes conexas tengo?

```
int componentes = 0;
for (int i = 0; i < n; i++) {
    if (!visitado[i]) {
        componentes++;
        recorro_a_partir_del_vertice(i);
    }
}
```

Problemas que se resuelven con ambos algoritmos

Chequeo de ciclos: el grafo tiene ciclos?

Problemas que se resuelven con ambos algoritmos

Chequeo de ciclos: el grafo tiene ciclos? Y si quiero poder guardar el ciclo? Vamos a usar algo bastante fuerte acá para hacerlo con DFS :P

Problemas que se resuelven con ambos algoritmos

Chequeo de ciclos: el grafo tiene ciclos? Y si quiero poder guardar el ciclo? Vamos a usar algo bastante fuerte acá para hacerlo con DFS :P

```
vector<int> padre(n, -1);
int comienzo_ciclo = -1, fin_ciclo = -1;

void dfs(int v) {
    for (int u : aristas[v]) {
        if (padre[u] == -1) {
            padre[u] = v;
            dfs(u);
        } else if (padre[v] != u) {
            // ciclo!
            comienzo_ciclo = v;
            fin_ciclo = u;
        }
    }
```

Problemas que se resuelven con ambos algoritmos

Chequeo de ciclos: el grafo tiene ciclos? Y si quiero poder guardar el ciclo? Vamos a usar algo bastante fuerte acá para hacerlo con DFS :P

```
padre[0] = 0
dfs(0);

vector<int> ciclo;
if (comienzo_ciclo >= 0) {
    int v = comienzo_ciclo;
    ciclo.push_back(v);
    while (v != fin_ciclo) {
        v = padre[v];
        ciclo.push_back(v);
    }
}
```

Problemas que se resuelven con ambos algoritmos

Tarea que dejo resuelta acá un poco feamente por si quieren revisar: chequeo de bipartito.

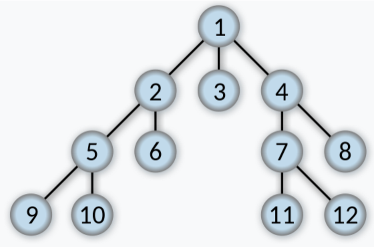
Problemas que se resuelven con ambos algoritmos

Tarea que dejo resuelta acá un poco feamente por si quieren revisar: chequeo de bipartito.

```
boolean es_bipartito;
void dfs(int v) {
    for (int u : aristas[v]) {
        if (color[u] == -1) { // si no esta pintado pinto
            color[u] = 1 - color[v];
            dfs(u);
        } else if (color[u] == color[v]) { // si ya esta pintado chequeo
            es_bipartito = false;
        }
    }
}

es_bipartito = true;
color[0] = 0;
dfs(0);
```

Estructura de BFS

BFS da una estructura que tiene pinta simpática.



Ahora supongamos que queremos calcular la cantidad de caminos con mínima cantidad de aristas entre nuestro vértice inicial 1 y el vértice 10. Cómo podemos hacer?

Estructura de BFS

Programación dinámica! Luego del BFS,

```
int cantidad_de_caminos_hasta(v) {
    if (distancia[v] == 0) return 1;
    if (memo[v] != -1) return memo[v];
    int res = 0;
    for (int vecino : aristas[v]) {
        if (distancia[vecino] + 1 == distancia[v]) {
            res += cantidad_de_caminos_hasta(vecino);
        }
    }
    memo[v] = res;
    return res;
}
```

Estructura de BFS

Ejercicio de parcial viejo de tarea: cuántos árboles distintos podría generar BFS?

Estructura de DFS

Vamos a charlar de 2 partes interesantes de la estructura del árbol que nos arma DFS:

- Los estados por los que pasa un nodo
- Los distintos tipos de aristas

Estructura de DFS

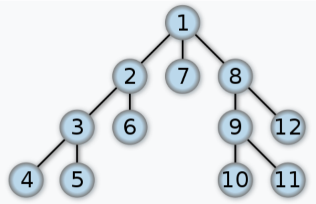
Vamos a charlar de 2 partes interesantes de la estructura del árbol que nos arma DFS:

- Los estados por los que pasa un nodo

Estados por los que pasa un nodo: son 3. Cuáles?

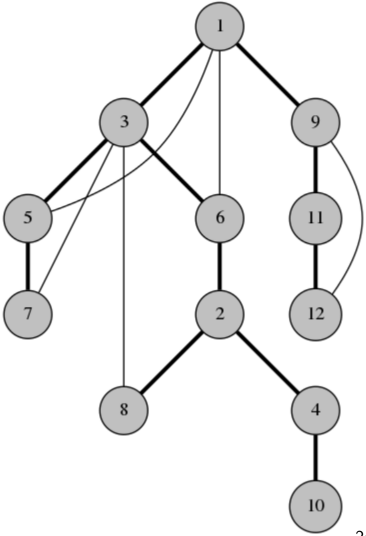
Estructura de DFS

```
vector<vector<int>> aristas = ... ;

int NO_LO_VI = 0, EMPECE_A_VER = 1,
    TERMINE_DE_VER = 2;
vector<int> estado(n, NO_LO_VI);

void dfs(int v) {
    estado[v] = EMPECE_A_VER;
    for (int u : aristas[v]) {
        if (estado[u] == NO_LO_VI)
            dfs(u);
    }
    estado[v] = TERMINE_DE_VER;
}
```



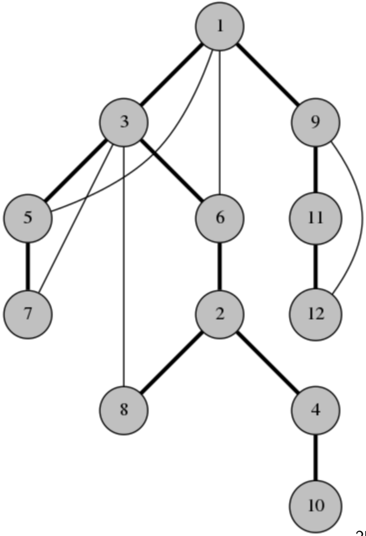
Estructura de DFS

```
vector<vector<int>> aristas = ... ;

int NO_LO_VI = 0, EMPECE_A_VER = 1,
    TERMINE_DE_VER = 2;
vector<int> estado(n, NO_LO_VI);

void dfs(int v) {
    estado[v] = EMPECE_A_VER;
    for (int u : aristas[v]) {
        if (estado[u] == NO_LO_VI)
            dfs(u);
    }
    estado[v] = TERMINE_DE_VER;
}
```



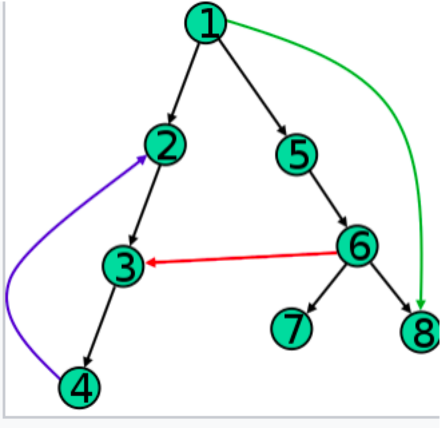
Estructura de DFS

```
vector<vector<int>> aristas = ... ;

int NO_LO_VI = 0, EMPECE_A_VER = 1,
    TERMINE_DE_VER = 2;
vector<int> estado(n, NO_LO_VI);

void dfs(int v) {
    estado[v] = EMPECE_A_VER;
    for (int u : aristas[v]) {
        if (estado[u] == NO_LO_VI)
            dfs(u);
        else // back-edge si no es el padre
    }
    estado[v] = TERMINE_DE_VER;
}
```



Estructura de DFS

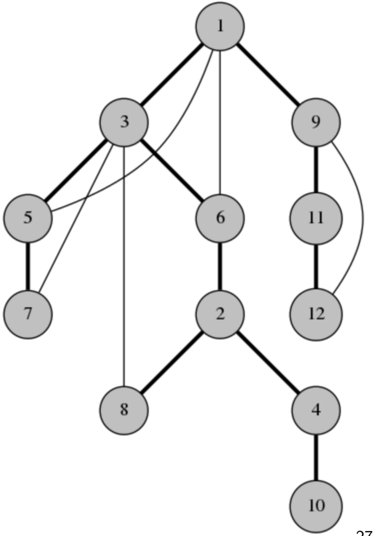
A veces sirve poner un contador para saber en qué momento se empezaron y terminaron de ver ciertos vértices. Especialmente para reconocer aristas cuando el grafo es dirigido.



Estructura de DFS

Miremos una arista (u, v), donde sin pérdida de generalidad DFS empieza a visitar a u antes que a v.

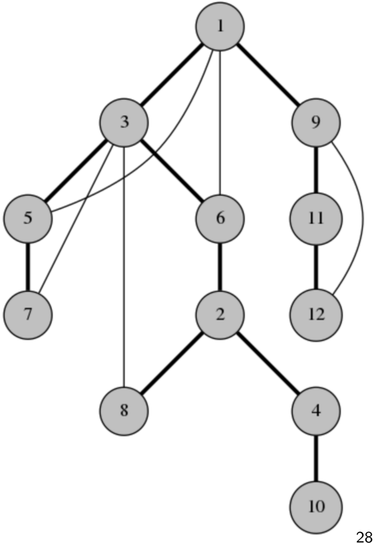
- La arista está en el árbol que arma DFS. A esto lo llamamos tree-edge.



Estructura de DFS

Miremos una arista (u, v), donde sin pérdida de generalidad DFS empieza a visitar a u antes que a v.

- La arista está en el árbol que arma DFS. A esto lo llamamos tree-edge.
- La arista no está en el árbol DFS.

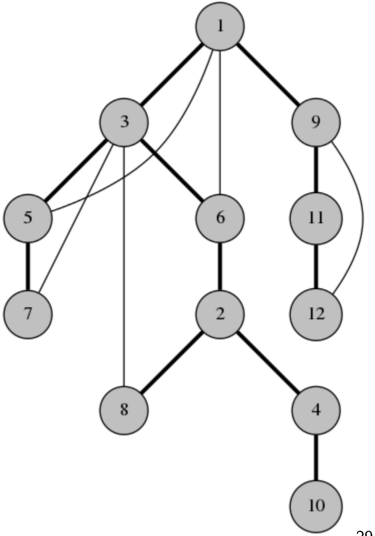


Estructura de DFS

Miremos una arista (u, v), donde sin pérdida de generalidad DFS empieza a visitar a u antes que a v.

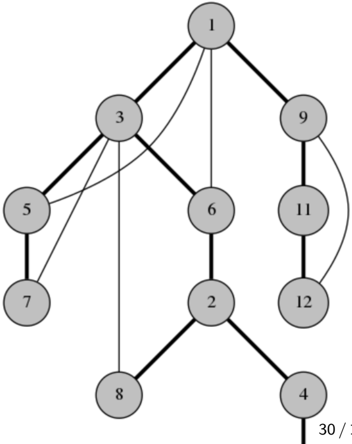
- La arista está en el árbol que arma DFS. A esto lo llamamos tree-edge.
- La arista no está en el árbol DFS. Esto significa que, antes de terminar el recorrido que partió en u, v fue visitado, al armar el subárbol que partió de algún otro hijo de u. Esto quiere decir que u es ancestro de v. A este tipo de aristas son las que llamamos back-edges.

Esta propiedad de que las back-edges conectan a nodos con ancestros de su árbol DFS está buena.



Puentes

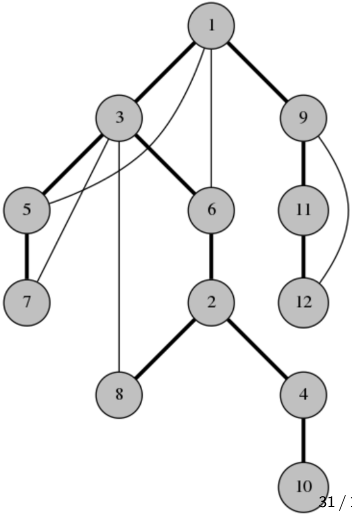
Vamos a quemar un ejercicio de la guía, que es clásico: qué aristas son puentes del grafo? De tarea, si no recuerdo mal, cambia muy poco la solución: qué nodos son puntos de articulación?



Puentes

Cómo encontramos qué aristas son puentes de un grafo?

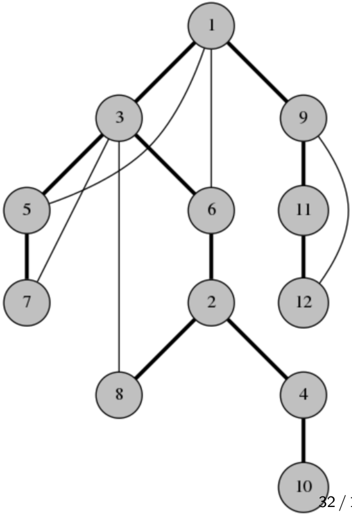
- Definamos arista puente



Puentes

Cómo encontramos qué aristas son puentes de un grafo?

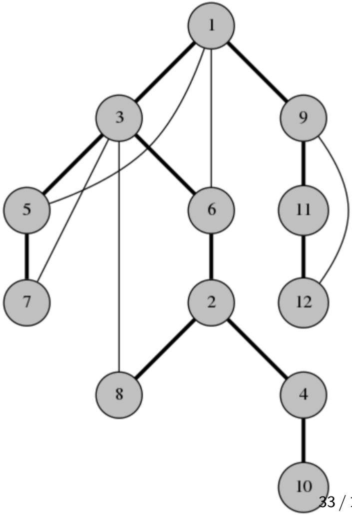
- Al quitarlas aumenta la cantidad de componentes conexas. Asumimos (u, v) con u descubierto antes que v.
- Una back-edge puede ser un puente?



Puentes

Cómo encontramos qué aristas son puentes de un grafo?

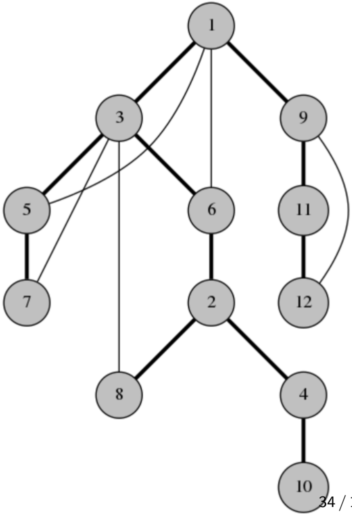
- Al quitarlas aumenta la cantidad de componentes conexas. Asumimos (u, v) con u descubierto antes que v.
- Una back-edge nunca es un puente
- Qué tiene que pasar en términos de back-edges para que una arista sea puente?



Puentes

Cómo encontramos qué aristas son puentes de un grafo?

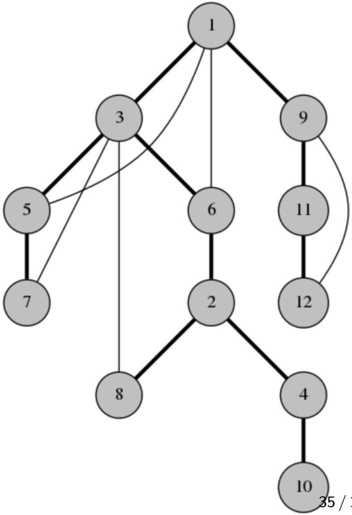
- Al quitarlas aumenta la cantidad de componentes conexas. Asumimos (u, v) con u descubierto antes que v.
- Una back-edge nunca es un puente
- Si hay una back-edge (a, b) con a descendiente de v y b ancestro de u, tenemos un camino de u a b, uno de b a a y uno de a a v. Mismas componentes conexas.
- Por otro lado, si sabemos que hay un camino alternativo de u a v que no pasa por la arista (u, v), el camino visto desde v comienza en el subárbol de v y sus descendientes, y termina fuera de el. La primera arista que pasa de un sector al otro no es una tree-edge (porque habría habido un ciclo), por ende es una back-edge, y "cubre" la arista (u, v)



Puentes

Cómo encontramos qué aristas son puentes de un grafo?

- Al quitarlas aumenta la cantidad de componentes conexas. Asumimos (u, v) con u descubierto antes que v.
- Una back-edge nunca es un puente
- Las aristas que son puentes son aquellas tree-edges que no tienen una back-edge que las "cubra"
- Cómo podemos encontrar qué aristas son puentes?

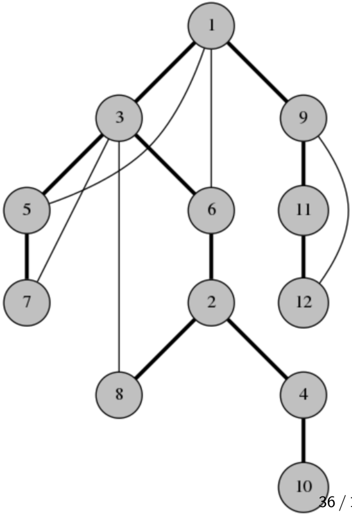


Puentes

Cómo encontramos qué aristas son puentes de un grafo?

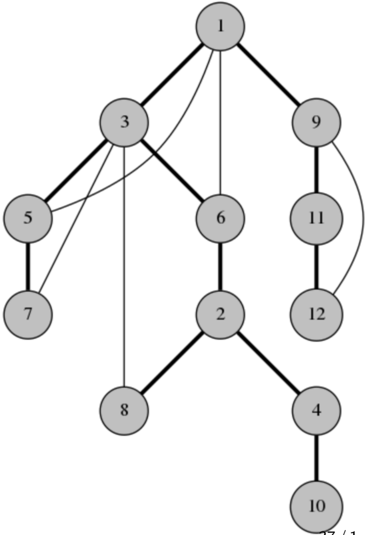
- Al quitarlas aumenta la cantidad de componentes conexas. Asumimos (u, v) con u descubierto antes que v.
- Una back-edge nunca es un puente
- Las aristas que son puentes son aquellas tree-edges que no tienen una back-edge que las "cubra"
- Usamos programación dinámica! La cantidad de back-edges que cubren la arista entre v y su padre es

$$cubren(v) = \sum_{w \text{ hijo de v}} cubren(w) - backEdgesConExtremoSuperiorEn(v) + backEdgesConExtremoInferiorEn(v).$$



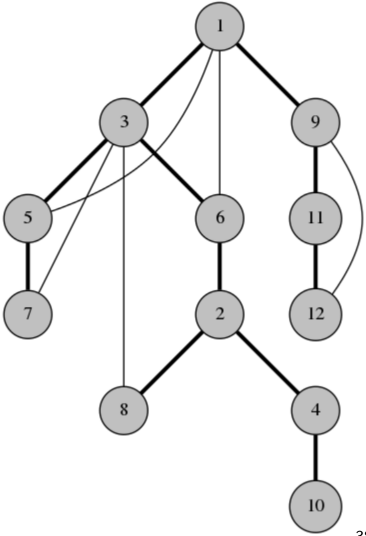
Puentes

```
vector<vector<int>> tree_edges(n);
...
void dfs(int v, int p = -1) {
    estado[v] = EMPECE_A_VER;
    for (int u : aristas[v]) {
        if (estado[u] == NO_LO_VI) {
            tree_edges[v].push_back(u);
            dfs(u, v);
        } else if (u != p) {
            back_edges_con_extremo_inferior_en[v]++;
            back_edges_con_extremo_superior_en[u]++;
        }
    }
    estado[v] = TERMINE_DE_VER;
}
```



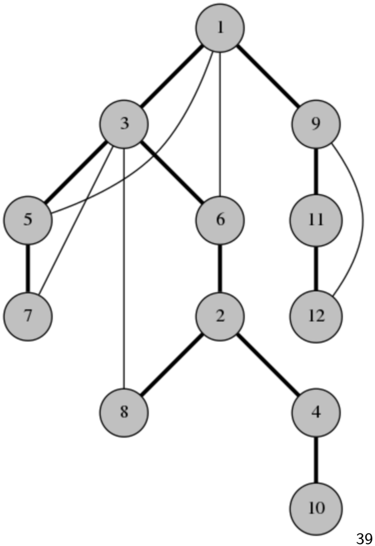
Puentes

```
vector<int> memo(n, -1);
int cubren(int v, int p = -1) {
    if (memo[v] != -1) return memo[v];
    int res = 0;
    for (int hijo : tree_edges[v]) {
        if (hijo != p) {
            res += cubren(hijo, v);
        }
    }
    res -= back_edges_con_extremo_superior_en[v];
    res += back_edges_con_extremo_inferior_en[v];

    memo[v] = res;
    return res;
}
```



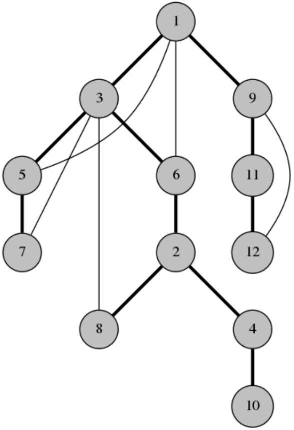
Puentes

```
int componentes = 0;
for (int i = 0; i < n; i++) {
    if (!visitado[i]) {
        dfs(i);
        componentes++;
    }
}
int puentes = 0;
for (int i = 0; i < n; i++) {
    if (cubren(i) == 0) {
        puentes++;
    }
}
puentes -= componentes; // hacky
```



Yapa: dirigir un grafo no dirigido

Queremos "dirigir" las aristas de un grafo para que nos quede fuertemente conexo. Se puede siempre? Pista:



Algunos problemitas dificiles que usan fuerte la estructura del árbol de DFS

- Strongly connected components (clásico): `https://cses.fi/problemset/task/1683`
  - (gran spoiler ponerlo acá) Contransmutation: `https://codingcompetitions.withgoogle.com/codejam/round/0000000000051679/0000000000146185#problem`
- Two Avenues (muy dificil, para que entre en tiempo en el judge hay que usar algunas estructuras de datos especiales): `https://codeforces.com/contest/1648/problem/F`
- We need more bosses (una idea simpática mirando fijo los puentes): `https://codeforces.com/contest/1000/problem/E`

Algunas fuentes / recursos

- `https://cp-algorithms.com/graph/depth-first-search.html`
- `https://cp-algorithms.com/graph/bridge-searching.html`
- `https://codeforces.com/blog/entry/68138`