CI2613: Algoritmos y Estructuras III

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Enero-Marzo 2015

Búsqueda en profundidad (DFS)

Búsqueda en profundidad

Búsqueda en profundidad o **depth-first search** (DFS) explora el grafo tratando de ir siempre lo "más profundo" posible:

– DFS explora las aristas no recorridas del vértice descubierto más reciente

– Una vez que todos las aristas son exploradas, la búsqueda continua (**"backtracks"**) con el vértice anterior más recientemente descubierto

– Al terminar la exploración, si todavía quedan vértices por descubrir, DFS se repite desde un tal vértice hasta que no queden vértices por descubrir

DFS es un algoritmo sistemático de exploración

Búsqueda en profundidad: Bosque de predecesores

Cada vez que se descubre un vértice v (al recorrer la lista de adyacencia de u), se coloca como "padre" de v al vértice u

DFS construye un grafo G_π de predecesores (como BFS)

G_π es un **bosque de áboles de predecesores** y no un único árbol de predecesores (como en BFS)

$G_\pi = (V, E_\pi)$ donde $E_\pi = \{(\pi[v], v) : v \in V \ \wedge \ \pi[v] \neq \text{null}\}$

Búsqueda en profundidad: Colores

DFS también colorea los vértices de blanco, gris y negro:

- Inicialmente todos los vértices son pintados de blancos
- Lo vértices recién descubiertos se pintan de gris
- Los vértices que se terminan de procesar (**finalizan**) se pintan de negro

Todos los vértices cambian de color: de blanco a gris a negro

Búsqueda en profundidad: Marcas de tiempo

DFS coloca marcas de tiempo (**timestamps**) a los vértices:

- Tiempo $d[u]$ al **descubrir** el vértice u: tiempo cuando se encuentra por primera vez y se pinta de gris
- Tiempo $f[u]$ al **finalizar** el vértice u: tiempo al terminar de recorrer la lista de adyacencia de u y cuando se pinta de negro

El tiempo es discreto, comenzando en 1 y avanzando 1 unidad en cada **evento:** descubrimiento ó finalización de algún vértice

El tiempo es siempre un entero entre 1 y $2|V|$, y es fácil verificar que $d[u] < f[u]$ para todo vértice $u \in V$ finalizado

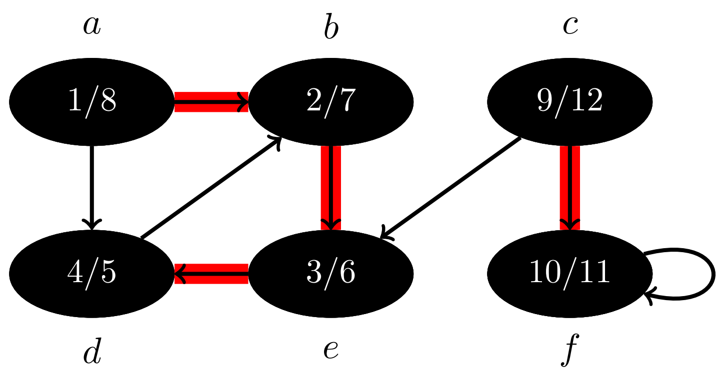
Búsqueda en profundidad: Pseudocódigo

```
void depth-first-search():
    % inicialización
    foreach Vertice u
        color[u] = Blanco
        π[u] = null             % padre de u en el bosque DFS

    % búsqueda recursiva
    time = 0
    foreach Vertice u
        if color[u] == Blanco
             DFS-Visit(u)

void DFS-Visit(Vertice u):
    time = time + 1
    d[u] = time                 % tiempo de descubrimiento de u
    color[u] = Gris
    foreach Vertice v in adyacentes[u]
        if color[v] == Blanco
            π[v] = u
            DFS-Visit(v)
    color[u] = Negro
    time = time + 1
    f[u] = time                 % tiempo de finalización de u
```


Búsqueda en profundidad: Ejemplo



Búsqueda en profundidad: Pseudocódigo

```
1  void depth-first-search():
2      % inicialización
3      foreach Vertice u
4          color[u] = Blanco
5          π[u] = null              % padre de u en el bosque DFS
6
7      % búsqueda recursiva
8      time = 0
9      foreach Vertice u
10         if color[u] == Blanco
11              DFS-Visit(u)
12
13 void DFS-Visit(Vertice u):
14     time = time + 1
15     d[u] = time                  % tiempo de descubrimiento de u
16     color[u] = Gris
17     foreach Vertice v in adyacentes[u]
18         if color[v] == Blanco
19             π[v] = u
20             DFS-Visit(v)
21     color[u] = Negro
22     time = time + 1
23     f[u] = time                  % tiempo de finalización de u
```


Búsqueda en profundidad: Análisis de tiempo

Entrada: grafo $G = (V, E)$ representado con **listas de adyacencia**
(Tamaño de la entrada es $\Theta(V + E)$)

Utilizamos la técnica de **análisis agregado**:

1. Sin contar el tiempo en la llamada recursiva, los lazos 3–5 y 9–11 toman tiempo $\Theta(V)$
2. DFS-Visit(u) es llamado exactamente una vez por cada vértice u: ya que u debe ser blanco y lo primero que se hace es pintarlo de gris
3. El lazo 17–20 se ejecuta, sin contar las llamadas recursivas, $\Theta(\delta(u)^+)$ veces.
4. Como $\sum_u \delta(u) = 2|E|$, el costo total del lazo 17–20 es $\Theta(E)$

El tiempo total de DFS es $\Theta(V + E)$ (i.e. **tiempo lineal**)

Búsqueda en profundidad: Paréntesis

Teorema (Paréntesis)

Sea $G = (V, E)$ un grafo (dirigido o no). En cualquier recorrido DFS de G, para cualquier par de vértices u y v, exactamente una de las siguientes condiciones se cumple:

(i) *los intervalos $[d[u], f[u]]$ y $[d[v], f[v]]$ son* **disjuntos**, *y uno no es descendiente del otro en el bosque DFS*

(ii) *$[d[u], f[u]]$ está contenido en $[d[v], f[v]]$, y u es* **descendiente** *de v*

(iii) *$[d[v], f[v]]$ está contenido en $[d[u], f[u]]$, y v es* **descendiente** *de u*

Búsqueda en profundidad: Paréntesis

Prueba: observe que para todo par de vértices u, v se cumple: $d[u] \neq f[v]$, y $d[u] \neq d[v]$ si $u \neq v$

Consideramos los casos $d[u] < d[v]$ y $d[u] > d[v]$:

1. $d[u] < d[v]$: consideramos los subcasos $d[v] < f[u]$ y $d[v] > f[u]$.

– Si $d[v] < f[u]$, v fue descubierto cuando u era todavía gris. Por lo tanto, v es un descendiente de u

Como v se descubre luego de descubrir u, todos sus aristas son exploradas y se finaliza antes que u; i.e. $f[v] < f[u]$ y tenemos $[d[v], f[v]] \subset [d[u], f[u]]$

– Si $d[v] > f[u]$, concluimos $d[u] < f[u] < d[v] < f[v]$ y los intervalos son disjuntos. En este caso, ninguno de los dos fue descubierto mientras el otro era gris, y por lo tanto ninguno es descendiente del otro

2. $d[u] < d[v]$: similar (intercambie u y v en el argumento de arriba) □

Búsqueda en profundidad: Paréntesis

Corolario

El vértice v es un descendiente de u en un bosque DFS de un grafo G (dirigido o no) si y sólo si $d[u] < d[v] < f[v] < f[u]$

Búsqueda en profundidad: Pseudocódigo

```
void depth-first-search():
    % inicialización
    foreach Vertice u
        color[u] = Blanco
        π[u] = null                 % padre de u en el bosque DFS

    % búsqueda recursiva
    time = 0
    foreach Vertice u
        if color[u] == Blanco
             DFS-Visit(u)

void DFS-Visit(Vertice u):
    time = time + 1
    d[u] = time                     % tiempo de descubrimiento de u
    color[u] = Gris
    foreach Vertice v in adyacentes[u]
        if color[v] == Blanco
            π[v] = u
            DFS-Visit(v)
    color[u] = Negro
    time = time + 1
    f[u] = time                     % tiempo de finalización de u
```


Búsqueda en profundidad: Caminos blancos

Teorema (Caminos blancos)

En un bosque DFS de un grafo G (dirigido o no), el vértice v es descendiente de u si y sólo si al momento de descubrir u (a tiempo $d[u]$), existe un camino desde u a v hecho totalmente de vértices blancos

Búsqueda en profundidad: Caminos blancos

Prueba:

$\Rightarrow$: Si $u = v$, el camino de u a v solo contiene a u que es blanco cuando se asigna valor a $d[u]$ (i.e. a tiempo $d[u]$)

Sea v un descendiente propio de u en el bosque DFS. Por el Corolario, $d[u] < d[v]$ y v es blanco a tiempo $d[u]$

Como v es arbitrario, todos los descendientes de u son blancos a tiempo $d[u]$, incluidos todos los vértices en el camino de u a v en el bosque DFS

Búsqueda en profundidad: Caminos blancos

Prueba:

$\Leftarrow$: Suponga que existe un camino blanco de u a v a tiempo $d[u]$, pero que v no termina como descendiente de u en el bosque DFS

Podemos asumir que todos los otros vértices en el camino excepto v son descendientes de u en el bosque (si no tome como v el primer vértice en el camino que no es descendiente de u)

Claramente, $u \neq v$. Sea w el predecesor de v en el camino: w es entonces descendiente de u en el bosque y, por el Corolario, $f[w] < f[u]$

Como v es blanco a tiempo $d[u]$, v se descubre luego de descubrir u. Por otro lado, v debe descubrirse antes de finalizar w (ya que $(w, v) \in E$). Entonces,

$$d[u] < d[v] < f[w] < f[u]$$

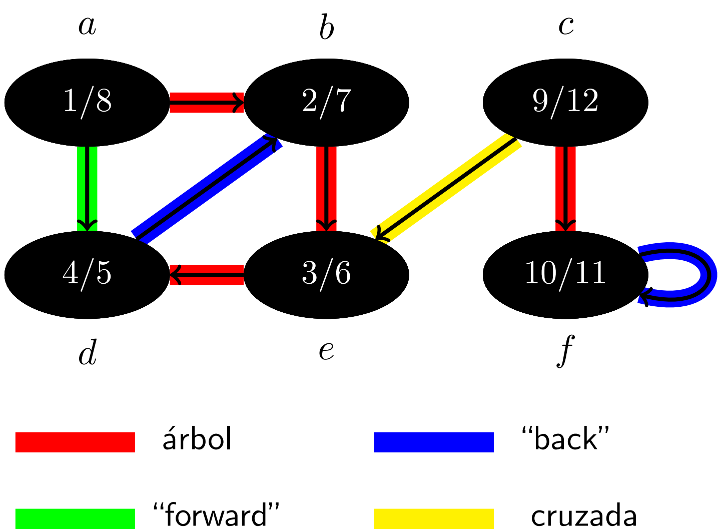
El Teorema de Paréntesis implica $[d[v], f[v]] \subset [d[u], f[u]]$. Por el Corolario, v es descendiente de u en el bosque DFS resultante $\square$

Clasificación de aristas en el bosque DFS

Las aristas del grafo son clasificadas dependiendo de cuando se descubren. Esta información es importante en ciertas aplicaciones

1. **Aristas de árbol:** son las aristas que definen el bosque (V, E_π): $(u, v) \in E_\pi$ si y sólo si v se descubrió al explorar (u, v)
2. **Aristas hacia atrás ("back"):** aristas (u, v) que conectan un vértice u con alguno de sus **ancestros** v en el bosque (e.g. los lazos son aristas "back")
3. **Aristas hacia adelante ("forward"):** aquellas aristas (u, v) que no están en el bosque pero que conectan a un vértice u con alguno de sus **descendientes propios** v
4. **Aristas cruzadas:** todas las otras aristas

Búsqueda en profundidad: Ejemplo



Clasificación de aristas en el bosque DFS

DFS puede clasificar las aristas (u, v) a medida que las explora:

1. **Aristas de árbol:** v es BLANCO
2. **Aristas hacia atrás ("back"):** v es GRIS
3. **Aristas hacia adelante ("forward"):** v es NEGRO y $d[u] < d[v]$
4. **Aristas cruzadas:** v es NEGRO y $d[u] > d[v]$

Ejercicio: demostrar que la detección propuesta es correcta

Clasificación de aristas en el bosque DFS

Si G es no dirigido, cada arista $\{u, v\}$ aparece como (u, v) y (v, u)

El tipo de $\{u, v\}$ es el de la **primera arista** (u, v) ó (v, u) **encontrada**

Teorema

Un recorrido DFS sobre un grafo no dirigido no produce ni aristas cruzadas ni aristas "forward"

Prueba: sea $(u, v) \in E$ y suponga (sin perder generalidad) que $d[u] < d[v]$

DFS descubre y finaliza v antes de finalizar u (por Teorema de Paréntesis)

– Si la primera vez que se explora $\{u, v\}$ es en la dirección (u, v), v es blanco a tiempo $d[u]$ y (u, v) es una arista de tipo árbol

– Si la primera exploración es en la dirección (v, u), v es gris y por lo tanto la arista es de tipo "back" $\square$