Optimización III (525551)

Módulo I: Algoritmos polinomiales

Pr. Julio Aracena.

Departamento de Ingeniería Matemática
Facultad de Ciencias Físicas y Matemáticas
Universidad de Concepción

Primer semestre, 2021

Algoritmo polinomial

Definición: Un algoritmo $\mathcal{A}$, con entrada x de tamaño $size(x) = n$, se dice polinomial si su tiempo de ejecución es $O(n^k)$, con $k \in \mathbb{N}$.

Algoritmo polinomial

Definición: Un algoritmo $\mathcal{A}$, con entrada x de tamaño $size(x) = n$, se dice polinomial si su tiempo de ejecución es $O(n^k)$, con $k \in \mathbb{N}$.

Ejemplo:

- El algoritmo Burbuja recibe como entrada un arreglo a de tamaño n y su tiempo de ejecución es $O(n^2)$. Luego, Burbuja es un algoritmo polinomial (cuadrático).

Algoritmo polinomial

Definición: Un algoritmo $\mathcal{A}$, con entrada x de tamaño $size(x) = n$, se dice polinomial si su tiempo de ejecución es $O(n^k)$, con $k \in \mathbb{N}$.

Ejemplo:

- El algoritmo Burbuja recibe como entrada un arreglo a de tamaño n y su tiempo de ejecución es $O(n^2)$. Luego, Burbuja es un algoritmo polinomial (cuadrático).
- El algoritmo de Euclides del MCD recibe como entrada $n, m \in \mathbb{N}$ con tamaño $\log(n) + \log(m)$ y su tiempo de ejecución es $O(\log(n) + \log(m))$, luego es polinomial (lineal).

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Algorithm Primo Básico(m)

Input: $m \in \mathbb{N},\ m \geq 3$

for $k = 2$ to $m - 1$ **do**
    $r \leftarrow m \mod k$
    if $r = 0$ **then**
        return m no es primo
    end if
end for
return m es primo

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Tiempo de ejecución:

Algorithm Primo Básico(m)

Input: $m \in \mathbb{N}$, $m \geq 3$

for $k = 2$ to $m - 1$ **do**

  $r \leftarrow m \mod k$

  if $r = 0$ **then**

    return m no es primo

  end if

end for

return m es primo

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Algorithm Primo Básico(m)

Input: $m \in \mathbb{N}, \ m \geq 3$

```
for k = 2 to m − 1 do
    r ← m mód k
    if r = 0 then
        return m no es primo
    end if
end for
return m es primo
```

Tiempo de ejecución:

1. En cada iteración del ciclo for se realizan 2 operaciones elementales (ie O(1) operaciones elementales).

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Algorithm Primo Básico(m)

```
Input: m ∈ ℕ, m ≥ 3
for k = 2 to m − 1 do
    r ← m mód k
    if r = 0 then
        return m no es primo
    end if
end for
return m es primo
```

Tiempo de ejecución:

1. En cada iteración del ciclo for se realizan 2 operaciones elementales (ie O(1) operaciones elementales).
2. En el peor caso se ejecutan $O(m)$ iteraciones del ciclo for.

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Algorithm Primo Básico(m)

Input: $m \in \mathbb{N},\ m \geq 3$

```
for k = 2 to m − 1 do
    r ← m mód k
    if r = 0 then
        return m no es primo
    end if
end for
return m es primo
```

Tiempo de ejecución:

1. En cada iteración del ciclo for se realizan 2 operaciones elementales (ie O(1) operaciones elementales).
2. En el peor caso se ejecutan $O(m)$ iteraciones del ciclo for.
3. Por lo tanto, el tiempo de ejecución es: $O(m) \cdot O(1) = O(m)$.

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

Algorithm Primo Básico(m)

Input: $m \in \mathbb{N},\ m \geq 3$

for $k = 2$ to $m - 1$ **do**
  $r \leftarrow m \mod k$
  if $r = 0$ **then**
    return m no es primo
  end if
end for
return m es primo

Tiempo de ejecución:

1. En cada iteración del ciclo for se realizan 2 operaciones elementales (ie O(1) operaciones elementales).
2. En el peor caso se ejecutan $O(m)$ iteraciones del ciclo for.
3. Por lo tanto, el tiempo de ejecución es: $O(m) \cdot O(1) = O(m)$.
4. Como el tamaño de la entrada es $size(m) = \log(m)$, entonces el algoritmo NO es polinomial ($O(m) \neq O(\log(m))^k$).

Algoritmo polinomial

El siguiente algoritmo determina si un número natural es o no primo.

Algoritmo de Euclides

```
Algorithm Primo Básico(m)
Input: m ∈ ℕ, m ≥ 3
for k = 2 to m − 1 do
    r ← m mód k
    if r = 0 then
        return m no es primo
    end if
end for
return m es primo
```

Tiempo de ejecución:

1. En cada iteración del ciclo for se realizan 2 operaciones elementales (ie O(1) operaciones elementales).
2. En el peor caso se ejecutan $O(m)$ iteraciones del ciclo for.
3. Por lo tanto, el tiempo de ejecución es: $O(m) \cdot O(1) = O(m)$.
4. Como el tamaño de la entrada es $size(m) = \log(m)$, entonces el algoritmo NO es polinomial $(O(m) \neq O(\log(m))^k)$.

Observación: Hay un algoritmo polinomial que determina si un entero $m \in \mathbb{N}$ es primo o no. (Agrawal et al. "Prime is P". Annals of Mathematics 160(2) (2004), pp 781-793).

Algoritmo base de exploración de grafos

Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algoritmo base de exploración de grafos

Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
1:  pred[s] ← Null
2:  Q ← {s},  M ← ∅
3:  while Q ≠ ∅ do
4:      Extraer v ∈ Q
5:      Marcar v (M ← M ∪ {v})
6:      for all u ∈ V sucesor de v do
7:          if u no está marcado then
8:              pred[u] ← v
9:              Q ← Q ∪ {u}
10:         end if
11:     end for
12: end while
13: return (M,pred)
```

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.

Algoritmo base de exploración de grafos

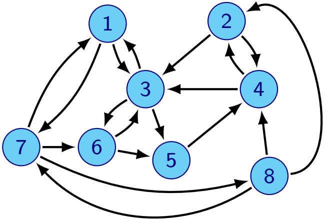
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- Inicio: $s = 1$; $pred[s] = Null$; $Q = \{1\}$

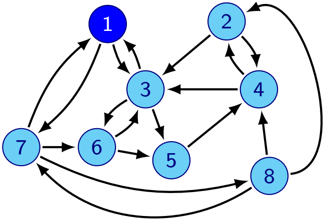
Algoritmo base de exploración de grafos

Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.
1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 1$: M:1;
  $pred[3] = pred[7] = 1$;
  $Q = \{3, 7\}$

Algoritmo base de exploración de grafos

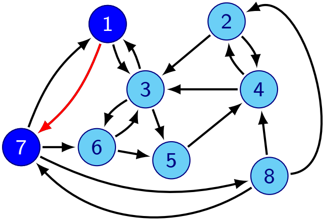
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}$, $M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v ($M \leftarrow M \cup \{v\}$)
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 2$: M:1, 7;
  $pred[6] = pred[8] = 7$,
  $Q = \{3, 6, 8\}$

Algoritmo base de exploración de grafos

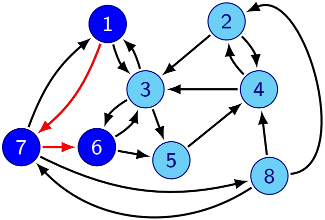
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 3$: M:$1, 7, 6$; $pred[5] = pred[3] = 6$; $Q = \{3, 5, 8\}$

Algoritmo base de exploración de grafos

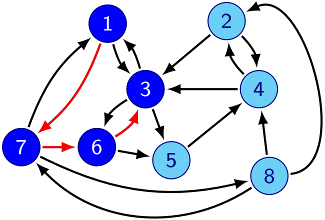
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}, \;\; M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 4$: M:$1, 7, 6, 3$; $pred[5] = 3$; $Q = \{5, 8\}$

Algoritmo base de exploración de grafos

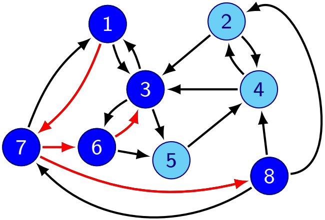
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}$, $M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4: Extraer $v \in Q$
5: Marcar v $(M \leftarrow M \cup \{v\})$
6: **for all** $u \in V$ sucesor de v **do**
7: **if** u no está marcado **then**
8: $pred[u] \leftarrow v$
9: $Q \leftarrow Q \cup \{u\}$
10: **end if**
11: **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 5$: M:$1, 7, 6, 3, 8$; $pred[2] = pred[4] = 8$; $Q = \{2, 4, 5\}$

Algoritmo base de exploración de grafos

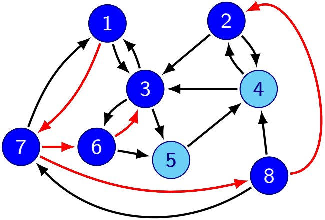
Sea $G=(V,E)$ un grafo (dirigido o no) con $|V|=n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G=(V,E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}$, $M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t=6$: M:$1,7,6,3,8,2$; $pred[4]=2$; $Q=\{4,5\}$

Algoritmo base de exploración de grafos

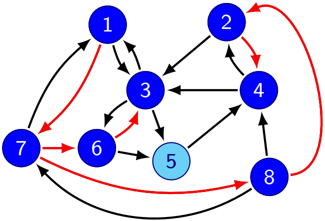
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}$, $M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 7$: M:1, 7, 6, 3, 8, 2, 4; $Q = \{5\}$

Algoritmo base de exploración de grafos

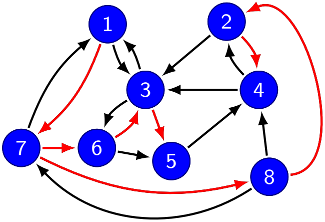
Sea $G = (V, E)$ un grafo (dirigido o no) con $|V| = n$ vértices y $|E|$ aristas. El siguiente es una algoritmo base de exploración o de búsqueda en G a partir de un vértice $s \in V$.

Algorithm Exploración Grafo (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

1: $pred[s] \leftarrow Null$
2: $Q \leftarrow \{s\}, \; M \leftarrow \emptyset$
3: **while** $Q \neq \emptyset$ **do**
4:     Extraer $v \in Q$
5:     Marcar v $(M \leftarrow M \cup \{v\})$
6:     **for all** $u \in V$ sucesor de v **do**
7:         **if** u no está marcado **then**
8:             $pred[u] \leftarrow v$
9:             $Q \leftarrow Q \cup \{u\}$
10:         **end if**
11:     **end for**
12: **end while**
13: **return** (M,pred)

Observación: Cuando ocurre $pred[u] \leftarrow v$ y $Q \leftarrow Q \cup \{u\}$ en las línea 8 y 9, se dice que u es "visitado" por v.



- $t = 8$: M:$1, 7, 6, 3, 8, 2, 4, 5$; $Q = \emptyset$.

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Corolario: *El algoritmo Exploración Grafo es polinomial. Consecuentemente, los siguientes problemas en grafos (dirigido o no dirigido) pueden ser resueltos por un algoritmo polinomial.*

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Corolario: *El algoritmo Exploración Grafo es polinomial. Consecuentemente, los siguientes problemas en grafos (dirigido o no dirigido) pueden ser resueltos por un algoritmo polinomial.*

- *Determinar si un grafo es conexo y sus componentes conexas.*

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Corolario: *El algoritmo Exploración Grafo es polinomial. Consecuentemente, los siguientes problemas en grafos (dirigido o no dirigido) pueden ser resueltos por un algoritmo polinomial.*

- *Determinar si un grafo es conexo y sus componentes conexas.*
- *Determinar si un grafo dirigido es fuertemente conexo y sus componentes fuertemente conexas.*

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Corolario: *El algoritmo Exploración Grafo es polinomial. Consecuentemente, los siguientes problemas en grafos (dirigido o no dirigido) pueden ser resueltos por un algoritmo polinomial.*

- *Determinar si un grafo es conexo y sus componentes conexas.*
- *Determinar si un grafo dirigido es fuertemente conexo y sus componentes fuertemente conexas.*
- *Determinar si existe un camino de un vértice a otro en un grafo dado.*

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Corolario: *El algoritmo Exploración Grafo es polinomial. Consecuentemente, los siguientes problemas en grafos (dirigido o no dirigido) pueden ser resueltos por un algoritmo polinomial.*

- *Determinar si un grafo es conexo y sus componentes conexas.*
- *Determinar si un grafo dirigido es fuertemente conexo y sus componentes fuertemente conexas.*
- *Determinar si existe un camino de un vértice a otro en un grafo dado.*
- *Determinar si un grafo tiene un ciclo (dirigido o no dirigido).*

Algoritmo base de exploración de grafos

Definición: Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Un nodo $v \in V$ se dice alcanzable desde s, o equivalentemente s alcanza v, si existe un camino en G de s a v.

Teorema: *Sea $G = (V, E)$ un grafo (dirigido o no dirigido) y $s \in V$. Entonces, en tiempo $O(|V| + |E|)$ el algoritmo Exploración Grafo entrega un conjunto de nodos marcados que corresponde exactamente a los nodos alcanzables desde s en G.*

Corolario: *El algoritmo Exploración Grafo es polinomial. Consecuentemente, los siguientes problemas en grafos (dirigido o no dirigido) pueden ser resueltos por un algoritmo polinomial.*

- *Determinar si un grafo es conexo y sus componentes conexas.*
- *Determinar si un grafo dirigido es fuertemente conexo y sus componentes fuertemente conexas.*
- *Determinar si existe un camino de un vértice a otro en un grafo dado.*
- *Determinar si un grafo tiene un ciclo (dirigido o no dirigido).*

Algoritmo base de exploración de grafos

Hay dos variantes importantes del algoritmo base de exploración de grafos, dependiendo de cómo se escoge $v \in Q$ para ser marcado:

Algoritmo base de exploración de grafos

Hay dos variantes importantes del algoritmo base de exploración de grafos, dependiendo de cómo se escoge $v \in Q$ para ser marcado:

Definición:

- Algorithmo de búsqueda en amplitud (o por capas) (**BFS**: Bread First Search)

Algoritmo base de exploración de grafos

Hay dos variantes importantes del algoritmo base de exploración de grafos, dependiendo de cómo se escoge $v \in Q$ para ser marcado:

Definición:

- Algorithmo de búsqueda en amplitud (o por capas) (**BFS**: Bread First Search)
- Algoritmo de búsqueda en profundidad (**DFS**:Depth First Search)

Algoritmo base de exploración de grafos

Hay dos variantes importantes del algoritmo base de exploración de grafos, dependiendo de cómo se escoge $v \in Q$ para ser marcado:

Definición:

- Algorithmo de búsqueda en amplitud (o por capas) (**BFS**: Bread First Search)
- Algoritmo de búsqueda en profundidad (**DFS**:Depth First Search)

En el caso de *BFS* el conjunto Q es implementado por una lista FIFO (First Input, First Output), ie. Q es una fila de elementos. En el caso de DFS, Q es una lista LIFO (Last Input, First Output), ie Q es una pila de elementos.

Algoritmo BFS

BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algoritmo BFS

BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V do
    π[v] ← Null, d[v] ← ∞
end for
d[s] ← 0, Q ← {s}
while Q ≠ ∅ do
    Extraer v ∈ Q
    Marcar v
    for all u ∈ V sucesor de v do
        if d[u] = ∞ then
            d[u] ← d[v] + 1, π[u] ← v
            Q ← Q ∪ {u}
        end if
    end for
end while
return (d,π)
```

Obs: Un nodo puede ser visitado solo una vez.

Algoritmo BFS

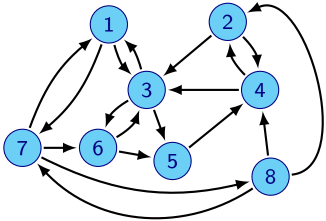
BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
    $\pi[v] \leftarrow Null,\ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0,\ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    Marcar v
    for all $u \in V$ sucesor de v **do**
        if $d[u] = \infty$ **then**
            $d[u] \leftarrow d[v] + 1,\ \pi[u] \leftarrow v$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return (d,π)

Obs: Un nodo puede ser visitado solo una vez.



- Inicio: $s = 1$; $d[1] = 0$; $\pi[1] = Null$; $Q = (1)$

Algoritmo BFS

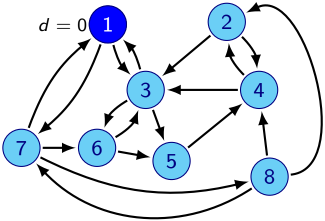
BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
    $\pi[v] \leftarrow Null, \ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0, \ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    Marcar v
    for all $u \in V$ sucesor de v **do**
        if $d[u] = \infty$ **then**
            $d[u] \leftarrow d[v] + 1, \ \pi[u] \leftarrow v$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return (d, π)

Obs: Un nodo puede ser visitado solo una vez.



- $t = 1$: M:1; $d[3] = d[7] = 1$; $\pi[3] = \pi[7] = 1$; $Q = (3, 7)$

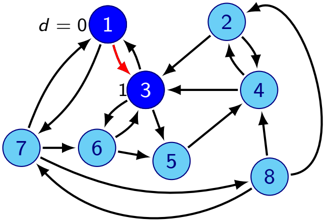
Algoritmo BFS

BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
    $\pi[v] \leftarrow Null,\ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0,\ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    Marcar v
    for all $u \in V$ sucesor de v **do**
        if $d[u] = \infty$ **then**
            $d[u] \leftarrow d[v] + 1,\ \pi[u] \leftarrow v$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return (d,π)



- $t = 2$: M:$1, 3$; $d[5] = d[6] = 2$; $\pi[5] = \pi[6] = 3$, $Q = (7, 5, 6)$

Obs: Un nodo puede ser visitado solo una vez.

Algoritmo BFS

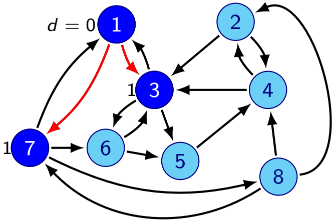
BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
    $\pi[v] \leftarrow Null,\ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0,\ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    Marcar v
    for all $u \in V$ sucesor de v **do**
        if $d[u] = \infty$ **then**
            $d[u] \leftarrow d[v] + 1,\ \pi[u] \leftarrow v$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return (d,π)

Obs: Un nodo puede ser visitado solo una vez.



- $t = 3$: M:1, 3, 7; $d[8] = 2$; $\pi[8] = 7$; $Q = (5, 6, 8)$

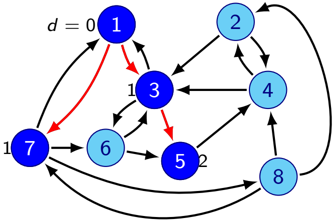
Algoritmo BFS

BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V do
    π[v] ← Null, d[v] ← ∞
end for
d[s] ← 0, Q ← {s}
while Q ≠ ∅ do
    Extraer v ∈ Q
    Marcar v
    for all u ∈ V sucesor de v do
        if d[u] = ∞ then
            d[u] ← d[v] + 1, π[u] ← v
            Q ← Q ∪ {u}
        end if
    end for
end while
return (d,π)
```



- $t = 4$: M:1, 3, 7, 5; $d[4] = 3$; $\pi[4] = 5$; $Q = (6, 8, 4)$

Obs: Un nodo puede ser visitado solo una vez.

Algoritmo BFS

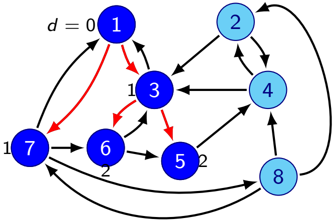
BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
    $\pi[v] \leftarrow Null,\ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0,\ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    Marcar v
    for all $u \in V$ sucesor de v **do**
        if $d[u] = \infty$ **then**
            $d[u] \leftarrow d[v] + 1,\ \pi[u] \leftarrow v$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return (d,π)

Obs: Un nodo puede ser visitado solo una vez.



- $t = 5$: M:1, 3, 7, 5, 6; $Q = (8, 4)$

Algoritmo BFS

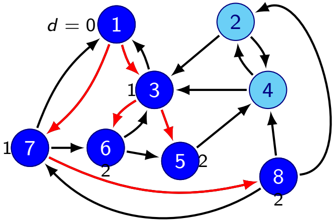
BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V do
    π[v] ← Null, d[v] ← ∞
end for
d[s] ← 0, Q ← {s}
while Q ≠ ∅ do
    Extraer v ∈ Q
    Marcar v
    for all u ∈ V sucesor de v do
        if d[u] = ∞ then
            d[u] ← d[v] + 1, π[u] ← v
            Q ← Q ∪ {u}
        end if
    end for
end while
return (d,π)
```

Obs: Un nodo puede ser visitado solo una vez.



- $t = 6$: M:$1, 3, 7, 5, 6, 8$; $d[2] = 3$; $\pi[2] = 8$; $Q = (4, 2)$

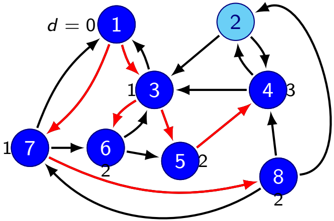
Algoritmo BFS

BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
  $\pi[v] \leftarrow Null,\ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0,\ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
  Extraer $v \in Q$
  Marcar v
  for all $u \in V$ sucesor de v **do**
    if $d[u] = \infty$ **then**
      $d[u] \leftarrow d[v] + 1,\ \pi[u] \leftarrow v$
      $Q \leftarrow Q \cup \{u\}$
    end if
  end for
end while
return (d,π)



- $t = 7$: M:$1, 3, 7, 5, 6, 8, 4$; $Q = (2)$

Obs: Un nodo puede ser visitado solo una vez.

Algoritmo BFS

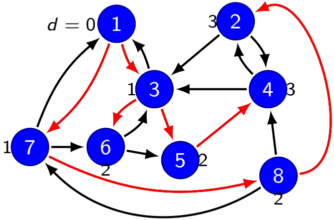
BFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de BFS es dos arreglos: d y π, donde $\forall v \in V$, si v es descendiente de s, $d[v]$ contiene la distancia (número de aristas de un camino más corto) de s a v y $\pi[v]$ el predecesor marcado de v. En caso contrario, $d[v] = \infty$ y $\pi[v] = Null$.

Algorithm BFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V$ **do**
    $\pi[v] \leftarrow Null,\ d[v] \leftarrow \infty$
end for
$d[s] \leftarrow 0,\ Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    Marcar v
    for all $u \in V$ sucesor de v **do**
        if $d[u] = \infty$ **then**
            $d[u] \leftarrow d[v] + 1,\ \pi[u] \leftarrow v$
            $Q \leftarrow Q \cup \{u\}$
        end if
    end for
end while
return (d,π)

Obs: Un nodo puede ser visitado solo una vez.



- $t = 8$: M:$1, 3, 7, 5, 6, 8, 4, 2$; $Q = \emptyset$.

Algoritmo BFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega BFS con entrada G y $s \in V$. Se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

Algoritmo BFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega BFS con entrada G y $s \in V$. Se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

- $V_\pi = \{v \in V : \ \pi[v] \neq Null\} \cup \{s\}$ y

Algoritmo BFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega BFS con entrada G y $s \in V$. Se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

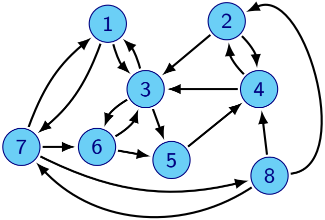
- $V_\pi = \{v \in V : \ \pi[v] \neq \mathit{Null}\} \cup \{s\}$ y
- $E_\pi = \{(\pi(v), v) \in E : \ v \in V_\pi \setminus \{s\}\}$.

Algoritmo BFS

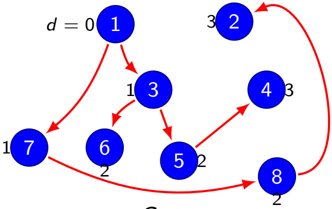
Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega BFS con entrada G y $s \in V$. Se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

- $V_\pi = \{v \in V : \pi[v] \neq Null\} \cup \{s\}$ y
- $E_\pi = \{(\pi(v), v) \in E : v \in V_\pi \setminus \{s\}\}$.

Ejemplo:



G



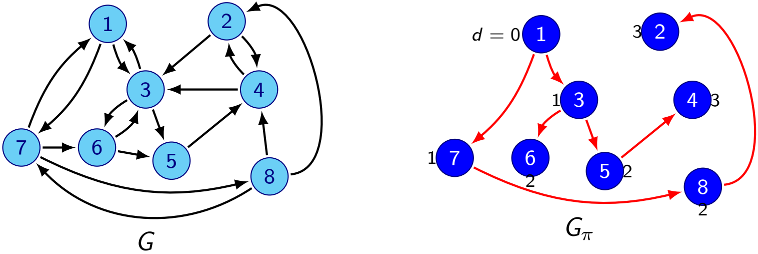
G_π

Algoritmo BFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega BFS con entrada G y $s \in V$. Se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

- $V_\pi = \{v \in V : \ \pi[v] \neq Null\} \cup \{s\}$ y
- $E_\pi = \{(\pi(v), v) \in E : \ v \in V_\pi \setminus \{s\}\}$.

Ejemplo:



Obs: Para G dado, G_π no es necesariamente único (depende de π).

Algoritmo BFS

Teorema: *Sea* $G = (V, E)$ *un grafo dirigido (no dirigido) conexo,* $s \in V$ *y* (d, π) *la salida de BFS con entrada* G *y* $s \in V$*. Entonces se tiene que:*

1. G_π *es un árbol dirigido (no dirigido) cubridor de* G *(ie* $V(G_\pi) = V$*).*
2. $\forall v \in V,\ d[v]$ *es la distancia de* s *a* v *en* G*.*

Algoritmo BFS

Teorema: *Sea $G = (V, E)$ un grafo dirigido (no dirigido) conexo, $s \in V$ y (d, π) la salida de BFS con entrada G y $s \in V$. Entonces se tiene que:*

1. *G_π es un árbol dirigido (no dirigido) cubridor de G (ie $V(G_\pi) = V$).*
2. *$\forall v \in V,\ d[v]$ es la distancia de s a v en G.*

Demo (idea): Se puede probar por inducción, sobre el número de iteración t del ciclo While de BFS, que para todo $v \in V$ marcado en una iteración anterior o igual a $t = k$:

Algoritmo BFS

Teorema: *Sea $G = (V, E)$ un grafo dirigido (no dirigido) conexo, $s \in V$ y (d, π) la salida de BFS con entrada G y $s \in V$. Entonces se tiene que:*

1. *G_π es un árbol dirigido (no dirigido) cubridor de G (ie $V(G_\pi) = V$).*
2. *$\forall v \in V,\ d[v]$ es la distancia de s a v en G.*

Demo (idea): Se puede probar por inducción, sobre el número de iteración t del ciclo While de BFS, que para todo $v \in V$ marcado en una iteración anterior o igual a $t = k$:

1. Hay un camino en G_π de s a v.

Algoritmo BFS

Teorema: *Sea $G = (V, E)$ un grafo dirigido (no dirigido) conexo, $s \in V$ y (d, π) la salida de BFS con entrada G y $s \in V$. Entonces se tiene que:*

1. *G_π es un árbol dirigido (no dirigido) cubridor de G (ie $V(G_\pi) = V$).*
2. *$\forall v \in V,\ d[v]$ es la distancia de s a v en G.*

Demo (idea): Se puede probar por inducción, sobre el número de iteración t del ciclo While de BFS, que para todo $v \in V$ marcado en una iteración anterior o igual a $t = k$:

1. Hay un camino en G_π de s a v.
2. $d_G(s, v) = d[v]$.

Algoritmo BFS

Teorema: *Sea $G = (V, E)$ un grafo dirigido (no dirigido) conexo, $s \in V$ y (d, π) la salida de BFS con entrada G y $s \in V$. Entonces se tiene que:*

1. *G_π es un árbol dirigido (no dirigido) cubridor de G (ie $V(G_\pi) = V$).*
2. *$\forall v \in V,\ d[v]$ es la distancia de s a v en G.*

Demo (idea): Se puede probar por inducción, sobre el número de iteración t del ciclo While de BFS, que para todo $v \in V$ marcado en una iteración anterior o igual a $t = k$:

1. Hay un camino en G_π de s a v.
2. $d_G(s, v) = d[v]$.

Sabemos que BFS marca todos los nodos alcanzables de s en G. Luego, de i) se tiene que G_π es conexo cubridor. Si además se prueba que G_π no tiene ciclos, entonces G_π es árbol (dirigido o no dirigido) cubridor.

Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\},\ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v ($M \leftarrow M \cup \{v\}$)
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)

Obs: Un nodo puede ser visitado varias veces.

Algoritmo DFS

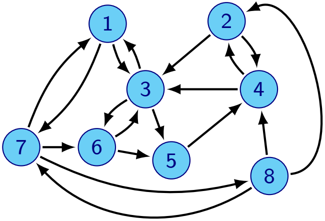
DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v $(M \leftarrow M \cup \{v\})$
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)

Obs: Un nodo puede ser visitado varias veces.



- Inicio: $s = 1$; $\pi[1] = Null$; $Q = (1)$

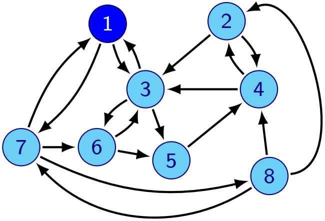
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V − {s} do
    π[v] ← Null
end for
Q ← {s},  M ← ∅
while Q ≠ ∅ do
    Extraer v ∈ Q
    if v no está marcado then
        Marcar v (M ← M ∪ {v})
        for all u ∈ V sucesor de v do
            if u no está marcado then
                π[u] ← v
                Q ← Q ∪ {u}
            end if
        end for
    end if
end while
return (M,π)
```



- $t = 1$: M:1; $\pi[3] = \pi[7] = 1$; $Q = (3, 7)$

Obs: Un nodo puede ser visitado varias veces.

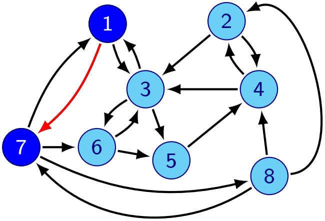
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V − {s} do
    π[v] ← Null
end for
Q ← {s},  M ← ∅
while Q ≠ ∅ do
    Extraer v ∈ Q
    if v no está marcado then
        Marcar v (M ← M ∪ {v})
        for all u ∈ V sucesor de v do
            if u no está marcado then
                π[u] ← v
                Q ← Q ∪ {u}
            end if
        end for
    end if
end while
return (M,π)
```



- $t = 2$: M:$1, 7$; $\pi[6] = \pi[8] = 7$, $Q = (3, 6, 8)$

Obs: Un nodo puede ser visitado varias veces.

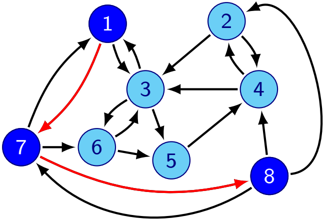
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}$, $M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v $(M \leftarrow M \cup \{v\})$
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)



- $t = 3$: M:$1, 7, 8$;
  $\pi[2] = \pi[4] = 8$;
  $Q = (3, 6, 2, 4)$

Obs: Un nodo puede ser visitado varias veces.

Algoritmo DFS

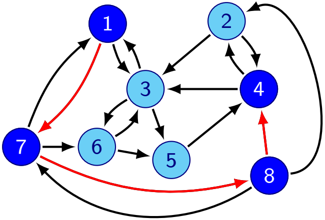
DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \; M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v $(M \leftarrow M \cup \{v\})$
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)

Obs: Un nodo puede ser visitado varias veces.



- $t = 4$: M:$1, 7, 8, 4$;
  $\pi[2] = \pi[3] = 4$;
  $Q = (3, 6, 2, 2, 3)$

Algoritmo DFS

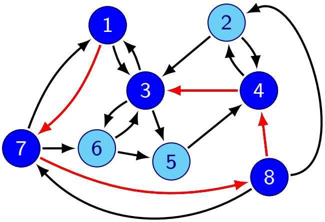
DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v $(M \leftarrow M \cup \{v\})$
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)

Obs: Un nodo puede ser visitado varias veces.



- $t = 5$: M:$1, 7, 8, 4, 3$;
  $\pi[5] = \pi[6] = 3$;
  $Q = (3, 6, 2, 2, 5, 6)$

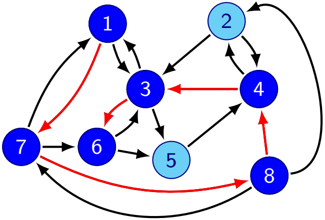
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V − {s} do
    π[v] ← Null
end for
Q ← {s},  M ← ∅
while Q ≠ ∅ do
    Extraer v ∈ Q
    if v no está marcado then
        Marcar v (M ← M ∪ {v})
        for all u ∈ V sucesor de v do
            if u no está marcado then
                π[u] ← v
                Q ← Q ∪ {u}
            end if
        end for
    end if
end while
return (M,π)
```



- $t = 6$: M:$1, 7, 8, 4, 3, 6$; $\pi[5] = 6$; $Q = (3, 6, 2, 2, 5, 3, 5)$

Obs: Un nodo puede ser visitado varias veces.

Algoritmo DFS

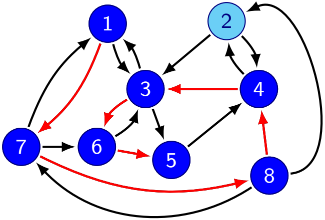
DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v $(M \leftarrow M \cup \{v\})$
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)

Obs: Un nodo puede ser visitado varias veces.



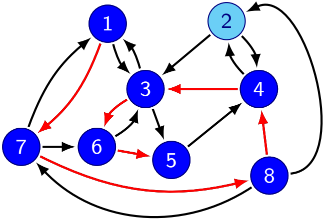
- $t = 7$: M:1, 7, 8, 4, 3, 6, 5; $Q = (3, 6, 2, 2, 5, 3)$

Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.
for all $v \in V - \{s\}$ **do**
  $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  Extraer $v \in Q$
  if v no está marcado **then**
    Marcar v $(M \leftarrow M \cup \{v\})$
    for all $u \in V$ sucesor de v **do**
      if u no está marcado **then**
        $\pi[u] \leftarrow v$
        $Q \leftarrow Q \cup \{u\}$
      end if
    end for
  end if
end while
return (M,π)



- $t = 8$: M:$1, 7, 8, 4, 3, 6, 5$; $Q = (3, 6, 2, 2, 5)$.

Obs: Un nodo puede ser visitado varias veces.

Algoritmo DFS

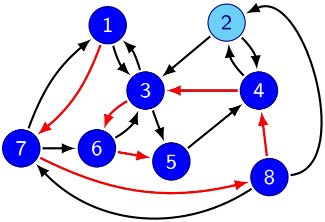
DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v ($M \leftarrow M \cup \{v\}$)
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)

Obs: Un nodo puede ser visitado varias veces.



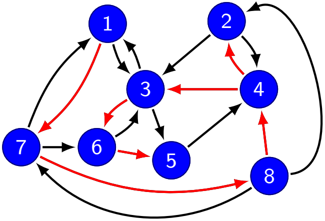
- $t = 9$: M:$1, 7, 8, 4, 3, 6, 5$; $Q = (3, 6, 2, 2)$.

Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

```
Input: G = (V, E) un grafo y s ∈ V.
for all v ∈ V − {s} do
    π[v] ← Null
end for
Q ← {s},  M ← ∅
while Q ≠ ∅ do
    Extraer v ∈ Q
    if v no está marcado then
        Marcar v (M ← M ∪ {v})
        for all u ∈ V sucesor de v do
            if u no está marcado then
                π[u] ← v
                Q ← Q ∪ {u}
            end if
        end for
    end if
end while
return (M,π)
```



- $t = 10$: M:$1, 7, 8, 4, 3, 6, 5, 2$; $Q = (3, 6, 2)$.

Obs: Un nodo puede ser visitado varias veces.

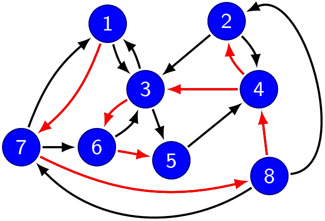
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
    $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\},\ M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
    Extraer $v \in Q$
    if v no está marcado **then**
        Marcar v $(M \leftarrow M \cup \{v\})$
        for all $u \in V$ sucesor de v **do**
            if u no está marcado **then**
                $\pi[u] \leftarrow v$
                $Q \leftarrow Q \cup \{u\}$
            end if
        end for
    end if
end while
return (M,π)



- $t = 11$: M:$1, 7, 8, 4, 3, 6, 5, 2$; $Q = (3, 6)$.

Obs: Un nodo puede ser visitado varias veces.

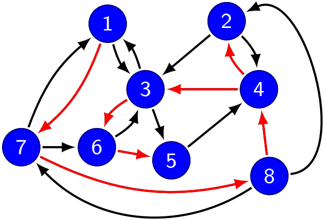
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

```
for all v ∈ V − {s} do
    π[v] ← Null
end for
Q ← {s},  M ← ∅
while Q ≠ ∅ do
    Extraer v ∈ Q
    if v no está marcado then
        Marcar v (M ← M ∪ {v})
        for all u ∈ V sucesor de v do
            if u no está marcado then
                π[u] ← v
                Q ← Q ∪ {u}
            end if
        end for
    end if
end while
return (M,π)
```



- $t = 12$: M:$1, 7, 8, 4, 3, 6, 5, 3, 2$; $Q = (3)$.

Obs: Un nodo puede ser visitado varias veces.

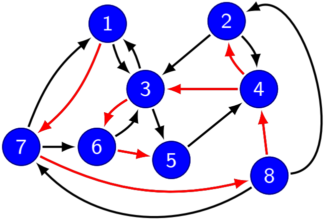
Algoritmo DFS

DFS recibe como entrada un grafo (dirigido o no) $G = (V, E)$ y $s \in V$. La salida de DFS es un arreglo π, donde $\forall v \in V$, $\pi[v]$ es el último predecesor marcado de v.

Algorithm DFS (G,s)

Input: $G = (V, E)$ un grafo y $s \in V$.

for all $v \in V - \{s\}$ **do**
  $\pi[v] \leftarrow Null$
end for
$Q \leftarrow \{s\}, \;\; M \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  Extraer $v \in Q$
  if v no está marcado **then**
    Marcar v $(M \leftarrow M \cup \{v\})$
    for all $u \in V$ sucesor de v **do**
      if u no está marcado **then**
        $\pi[u] \leftarrow v$
        $Q \leftarrow Q \cup \{u\}$
      end if
    end for
  end if
end while
return (M,π)



- $t = 13$: M:1, 7, 8, 4, 3, 6, 5, 3, 2; $Q = \emptyset$.

Obs: Un nodo puede ser visitado varias veces.

Algoritmo DFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega DFS con entrada G y $s \in V$. De igual forma que lo definido para BFS, se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

Algoritmo DFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega DFS con entrada G y $s \in V$. De igual forma que lo definido para BFS, se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

- $V_\pi = \{v \in V : \pi[v] \neq Null\} \cup \{s\}$ y

Algoritmo DFS

Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega DFS con entrada G y $s \in V$. De igual forma que lo definido para BFS, se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

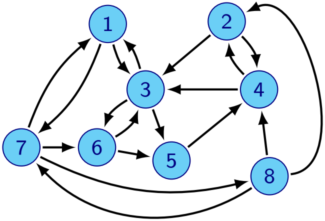
- $V_\pi = \{v \in V : \pi[v] \neq Null\} \cup \{s\}$ y
- $E_\pi = \{(\pi(v), v) \in E : v \in V_\pi \setminus \{s\}\}$.

Algoritmo DFS

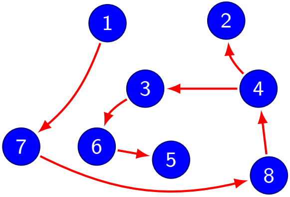
Definición: Sea $G = (V, E)$ un grafo dirigido (no dirigido) y π el arreglo de predecesores que entrega DFS con entrada G y $s \in V$. De igual forma que lo definido para BFS, se define el grafo dirigido (no dirigido) predecesor $G_\pi = (V_\pi, E_\pi)$ por:

- $V_\pi = \{v \in V : \ \pi[v] \neq Null\} \cup \{s\}$ y
- $E_\pi = \{(\pi(v), v) \in E : \ v \in V_\pi \setminus \{s\}\}$.

Ejemplo:



G



G_π

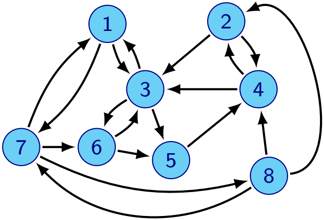
Obs: Para G dado, G_π no es necesariamente único (depende de π).

Ejemplo de BFS y DFS

Ejemplo:

Ejemplo de BFS y DFS

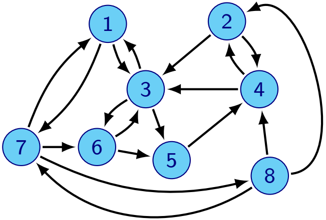
Ejemplo:



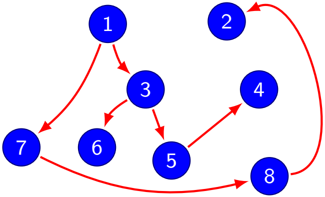
G

Ejemplo de BFS y DFS

Ejemplo:



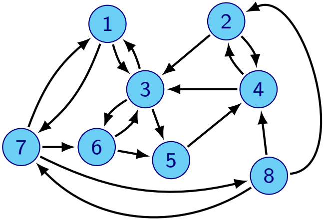
G



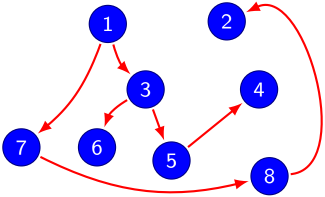
G_π obtenido con **BFS**

Ejemplo de BFS y DFS

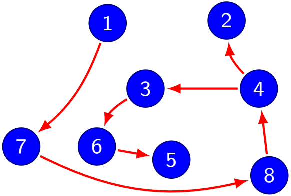
Ejemplo:



G



G_π obtenido con **BFS**



G_π obtenido con **DFS**