Chapter 10

Grafos de Precedencia

Definición 29 *Grafo de Precedencia Sea $G = (V, E)$ un digrafo, G es un grafo de precedencia sii G no posee circuitos y E representa una relación de orden parcial sobre los elementos en V.*

10.1 Usos

- Planificación de actividades
- Pensum de estudio (Grafo de la carrera)
- Expresión aritmética

Definición 30 *Vértice Fuente Sea $G = V, E)$ un grafo de precedencia y $v \in V$, v es un vértice fuente sii $d^-(v) = 0$ $\boxed{y\ d^+(v) > 0}$[1].*

Definición 31 *Vértice Sumidero Sea $G = V, E)$ un grafo de precedencia y $v \in V$, v es un vértice sumidero sii $d^+(v) = 0$ $\boxed{y\ d^-(v) > 0}$[2].*

En los casos que las condiciones $\boxed{d^+(v) > 0}$ y $\boxed{d^-(v) > 0}$ sean relajadas de las definiciones 30 y 31 respectivamente, también se usa la siguiente definición:

Definición 32 *Vértice Aislado Sea $G = V, E)$ un grafo de precedencia y $v \in V$, v es un vértice sumidero sii $d^+(v) = 0$ y $d^-(v) = 0$.*

[1] En ciertos casos esta condición tiende a ser relajada
[2] En ciertos casos esta condición tiende a ser relajada

Es decir, un vértice aislado se puede ver como un vértice sumidero y fuente al mismo tiempo.

10.2 Propiedades de grafos de precedencia:

- Un digrafo $G = (V, E)$, G es de precedencia si G^{-1} es de precedencia.
- Un digrafo G no posee circuito sii todo subgrafo de G no posee circuitos.

Definición 33 *Nivel y Altura Sea* $G = (V, E)$ *un digrafo:*

$$\begin{array}{rrl} \textit{nivel de } v: & \eta(v) = & max\{l(c) \in C(G) | \exists s \in V, c =< s, ..., v >\} \\ \textit{altura de } v: & h(v) = & max\{l(c) \in C(G) | \exists s \in V, c =< v, ..., s >\} \end{array}$$

$\eta(v)$ representa la longitud del camino elemental más largo terminado en v.

$h(v)$ representa la longitud del camino elemental más largo comenzado en v.

Proposición 8 *Sea* $G = (V, E)$ *un grafo de precedencia,* $\forall v \in V$*, el vértice inicial de un camino de largo máximo, terminado en* v*, es un vértice fuente.*

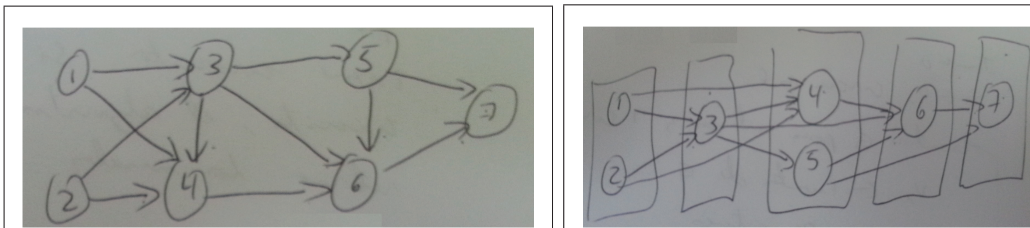
Dem:

Proposición 9 *Sea* $G = (V, E)$ *un digrafo,* G *es de precedencia si y sólo si* V *puede particionarse en los conjuntos* $V_0, V_1, ..., V_p$*, tal que,* $\forall i \in [0, p]$*,* $(\forall v \in V_i,\ \eta(v) = i)$*.*

Dem:

10.3 Relación de orden

Note que en base a la proposición anterior, los vértices de un grafo de precedencia pueden particionarse en conjuntos V_i, tal que, si $v \in V_i$, entonces $\eta(v) = i$. Por consiguiente, un grafo de precedencia puede ser particionado por niveles.



(a) Un grafo de precedencia

(b) Las niveles/capas del grafo

Figure 10.1: Grafo de Precedencia

Note que un vértice puede preceder inmediatamente a otro ($V_1 \in predecesores(V_4)$ y no estar en niveles no consecutivos ($V_1 \in V_0$ y $V_4 \in V_2$).

10.3.1 Particionamiento por capas y reconocimiento de grafos de precedencia

Algoritmo 6: Particionamiento por capas y Reconocimiento de Grafos de precedencia

Entrada: $G = (V, E)$: Un grafo dirigido simple
Salida: $circuito : Boolean$, indica si G tiene algún circuito
Salida: si $circuito$ es falso, $\{V_i | v \in V_i,\ \eta(v) = i\}$
Comienzo
  $i \leftarrow 0$
  $circuito \leftarrow falso$
  while $|V| = 0 \wedge$ *No circuito* **do**
    $V_i \leftarrow \{v \in V | v$ es fuente en $G\}$
    Si $V_i = \emptyset$ **Entonces**
      $circuito \leftarrow cierto$
    sino
      Eliminar de G los vértices en V_i
      $i \leftarrow i + 1$
  Si *circuito* **Entonces**
    Print "El grafo tiene circuitos"
Fin

Orden: $O(max(|V|, |E|))$

10.4 Orden Topológico

Definición 34 *Orden Topológico Sea $G = (V, E)$ digrafo y $f : V \to \mathbb{N}$ una función inyectiva, f es un orden topológico si $\forall (v, u) \in E$, entonces $f(v) < f(u)$.*

Proposición 10 *Sea $G = (V, E)$ un grafo, G es un grafo de precedencia sii admite un orden topológico.*

Dem:

Algoritmo 7: Orden Topológico

Entrada: $G = (V, E)$: Un grafo dirigido simple sin circuitos.
Salida: $f : V \to \mathbb{N}$, f un orden topológico.
Variable: f y *visitado* arreglos indizados por $v \in V$.
Variable: *contador* :
Comienzo
  contador $\leftarrow |V| + 1$
  Para todo $v \in V$ **hacer**
    Si *No visitado*[v] **Entonces**
      `DFS_Recursivo` (v)
  Retornar f
Fin

Funcion `DFS_Recursivo` (v)
Comienzo
  visitado[v] $\leftarrow$ *cierto*
  Para todo $w \in sucesores(v)$ **hacer**
    Si *No visitado*[w] **Entonces**
      `DFS_Recursivo` (w)
  contador $\leftarrow$ *contador* $- 1$
  $f[v] \leftarrow$ *contador*
Fin

Orden: $O(max(|V|, |E|))$

10.5 Algoritmo de costo mínimo

10.5.1 Bellman Ford

Cáculo de caminos de costo mínimo desde un nodo fuente s. Retorna sólo un camino de costo mínimo por cada alcanzable. Al igual que Dijkstra, se usa este principio.

$$costo[v] = min\{costo[w] + c((w, v))|(w, v) \in E\}$$

donde: costo[v] es el costo del camino $< s, .., v >$ y $c(e)$ es el costo del arco $e = (w, v)$.

Sin embargo, recorren el grafo en forma diferente. Dijkstra recorre seleccionado el camino de menor costo. Mientras que Bellman va recorriendo el grafo en amplitud.

Algoritmo 8: Bellman-Ford : Costo mínimo (Cormen)

Entrada: $G = (V, E)$: Un grafo dirigido simple
Entrada: $s \in V$ un nodo incial
Entrada: $c : E \rightarrow \mathbb{R}$ función de costos
Salida: $\{< s, ..., a >\in C(G)|cmin(s, a) =< s, ..., a >\}$:
Comienzo
 Para todo $n \in V$ **hacer**
  Si $n = s$ **Entonces**
   $costo[n] \leftarrow 0$
  sino
   $costo[n] \leftarrow +\infty$
  $predecesor \leftarrow NULL$
 $i \leftarrow 1$
 $cambio \leftarrow cierto$
1 **while** $i < |V| \wedge cambio$ **do**
  $cambio \leftarrow falso$
  Para todo $(n, m) \in E$ **hacer**
   Si $costo[m] > costo[n] + c((n, m))$ **Entonces**
    $costo[m] \leftarrow costo[n] + c((n, m))$
    $predecesor[m] \leftarrow n$
    $cambio \leftarrow cierto$
  $i \leftarrow i + 1$
 Para todo $(n, m) \in E$ **hacer**
2   **Si** $costo[m] > costo[n] + c((n, m))$ **Entonces**
   Retornar "Error, hay un circuito de peso negativo"
Fin

LEER: Note la condición de la línea 2 para verificar circuitos de costo negativo. Recuerde que al final del algoritmo $costo[m]$ debe ser el costo del $cmin(s, m)$, es decir, $\forall (n, m) \in E$ se debe cumplir que $costo[m] \leq costo[n] + c(n, m)$. Por consiguiente si existe un $(n, m) \in E$ y $costo[m] > costo[n] + c(n, m)$ se está diciendo que el valor en $costo[m]$ no corresponde con el $cmin(s, m)$. Y por la forma cómo el ciclo 1 calcula los costos, esto no puede pasar a menos que haya un circuito de costo negativo.

Asimismo, note que en este caso, el ciclo en la línea 1 termina cuando $i = |V|$, debido a que siempre habrá un cambio. Note que en el caso que no haya circuitos, este ciclo se llevará a cabo $|V| - 1$ veces. Esto es debido a que se sabe que el camino elemental más largo que puede tener un grafo tiene a lo sumo esa longitud. En base a este argumento se puede decir, que si se conoce el largo L de la cadena elemental más larga posible en el grafo que se éste usando, entonces sólo es necesario iterar L veces. El problema aquí radica en que no siempre se conoce este valor.

Orden: $O(|V| * |E|)$

10.5.2 Bellman para grafos sin circuitos

Esta versión del algoritmo de Bellman permite calcular un camino de costo mínimo desde un vértice s hasta cada alcanzable en un grafo sin circuitos. Usando el hecho que el grafo no tiene circuitos, esta versión se va moviendo por los vértices siguiendo un orden topológico. Esto es debido a que considera los vértices en forma similar a como éstos están dispuestos en las capas del grafo: un vértice no es agregado a T hasta que todos sus predecesores hayan sido procesados.

Algoritmo 9: Bellman : Costo mínimo (Ortega&Meza)

Entrada: $G = (V, E)$: Un grafo dirigido simple sin circuitos.
Entrada: $s \in V$ un nodo incial
Entrada: $c : E \to \mathbb{R}$ función de costos
Salida: $\{< s, ..., a >\in C(G) | cmin(s, a) =< s, ..., a >\}$:
Variable: T : Conjunto de vértices
Variable: *predecesor*, *grado* y *costo* arreglos indizados por $v \in V$.
Comienzo

```
    T ← {s}
    Para todo v ∈ V hacer
        Si v = s Entonces
            costo[v] ← 0
            grado[v] ← 0
        sino
            costo[v] ← +∞
            grado[v] ← d⁻(v)
        predecesor[v] ← NULL
    while T ≠ ∅ do
        Tomar un vértice n ∈ T
        T ← T − {n}
        Para todo m ∈ sucesores(n) hacer
            grado[m] ← grado[m] − 1
            Si grado[m] = 0 Entonces
                T ← T ⋃{m}
            Si costo[m] > costo[n] + c((n, m)) Entonces
                costo[m] ← costo[n] + c((n, m))
                predecesor[m] ← n
```

Fin

Orden: $O(max(|V|, |E|))$

10.5.3 Bellman: Programación Dinámica Regresiva

Este algoritmo se usa para calcular el camino de costo mínimo entre el vértice s y el vértice t.

El criterio de Bellman nos dice:

$$cmin(s,t) = min\{cmin(s,w) + c(w,t)|(w,t) \in E\}$$

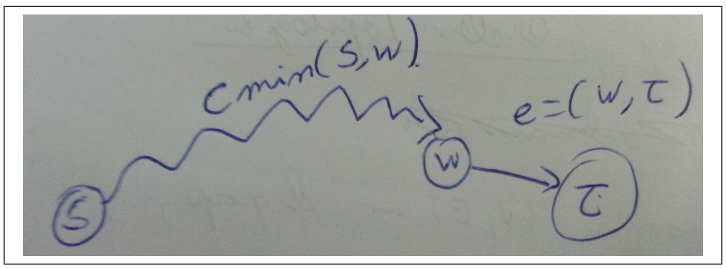


Figure 10.2: Belman: Criterio de costo mínimo de s a t

El principio dual de Bellman

$$cmin(s,t) = min\{c(s,w) + cmin(w,t)|(s,w) \in E\}$$

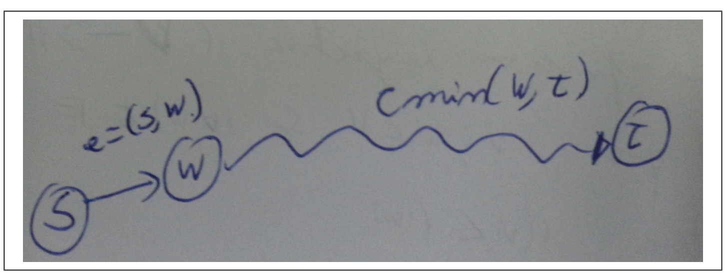


Figure 10.3: Belman: Criterio de dualidad de costo mínimo de s a t

En base a esto, la versión de Bellman presentada en esta seción va construyendo el camino del vértice s al vértice t, desde el final hacia adelante. Esto se hace siguiendo la estrategia ya estudiada que permite identificar un orden topológico en un grafo de precedencias. La modificación clave se agrega en la parte del algoritmo DFS Recursivo correspondiente al *finishing time*, es decir, cuando un vértice n no será tomando más en cuenta: cuando todos sus descendientes ya fueron procesados y por consiguiente el camino de costo mínimo desde los sucesores m de n hasta t ya ha sido identificado, es decir, los $cmin(m, t)$.

Algoritmo 10: Bellman: Programación Dinámica Regresiva

Entrada: $G = (V, E)$: Un grafo simple de precedencia.
Entrada: $s, t \in V$, vértice inicial y final respectivamente
Entrada: $c : E \to \mathbb{R}$ una función de costos
Salida: $< s, ..., t >$ un camino de costo mínimo.
Comienzo
  Para todo $v \in V$ **hacer**
    $visitado[v] \leftarrow falso$
    $costo[v] \leftarrow +\infty$
    $siguiente[v] \leftarrow NULL$
  `DFS_Recursivo` (s)
Fin

Funcion `DFS_Recursivo` (n)
Comienzo
  $visitado[n] \leftarrow cierto$
  Si $x = t$ **Entonces**
    $costo[n] \leftarrow 0$
  sino
    Para todo $m \in sucesores(n)$ **hacer**
      Si *No visitado*$[m]$ **Entonces**
        `DFS_Recursivo` (m)
    Para todo $m \in sucesores(n)$ **hacer**
      Si $(costo[m] \neq +\infty) \wedge (costo[n] > costo[m] + c((n, m)))$
      Entonces
        $costo[n] \leftarrow costo[m] + c((n, m))$
        $siguiente(n) \leftarrow m$
Fin

En los algotimos estudiados anteriormente, los caminos se construyen desde el nodo de inicio s hacia sus alcanzables. Al recuperar los caminos se hace siguiendo los apuntadores (en el arreglo *predecesores*) desde el final hacia adelante, es decir, desde cada alcanzable hasta el nodo inicial s. Sin embargo, en este caso note que el camino de s a t se construye desde el final hacia adelante y la recuperación se realiza desde s hacia t usando la información en el arreglo *siguiente*.

Orden $O(max(|V|, |E|))$