PROGRAMACION II

MATERIAL para TEORIA

Grafos

Código Asignatura 6A4

Año 2021

REPRESENTACION

↳ REPRESENTACION EN MEMORIA de GRAFOS

① Representación Matricial ⇨ Matriz de Adyacencia
② Representación por medio de Listas ⇨ Lista de Adyacencia
⇨ Multilista

↳ REPRESENTACION EN MEMORIA de DIGRAFOS

① Representación Matricial ⇨ Matriz de Adyacencia
② Representación por medio de Listas ⇨ Lista de Adyacencia

Se definen:

- ✓ Matriz de Clausura Transitiva

$$CT[i,j] = \begin{cases} 1 & \text{si existe un camino de i a j de longitud} > 0 \\ 0 & \text{en caso contrario} \end{cases}$$

- ✓ Matriz de Clausura Transitiva Reflexiva

$$CTR[i,j] = \begin{cases} 1 & \text{si existe un camino de i a j de longitud} \geq 0 \\ 0 & \text{en caso contrario} \end{cases}$$

GRAFOS

Subgrafos

Sea G = (V,E), un **subgrafo** G' de G es un grafo G' = (V',E') tal que:

- $V' \subseteq V$ y $E' \subseteq E$, tal que en E' hay aristas (v,w) ε E con v y w ε V'.

Si en E' están todas las aristas (v,w) de E con v y w en V', se dice que G' es un **subgrafo inducido.**

Recorridos

El objetivo es visitar todos los nodos del grafo exactamente una vez (se supone conexo)

- ✓ en *Amplitud (BFS)*
  - se selecciona un nodo inicial, se visita y se marca como visitado
  - luego todos los nodos no visitados adyacentes al nodo inicial se visitan y se marcan como visitados
  - luego se visitan los nodos no visitados adyacentes a los ya visitados, tomándolos *en el orden* en el que se han visitado
  - se prosigue de la misma manera hasta que no haya nodos sin visitar.
- ✓ en *Profundidad (DFS)*
  - se selecciona un nodo inicial, se visita y se marca como visitado
  - luego se visita un nodo adyacente al nodo inicial, se visita y se marca como visitado,
  - se prosigue de la misma manera tomando un nodo no visitado adyacente al *último nodo* visitado; en caso de no haber ninguno, se va intentando con los ya visitados en orden inverso al que se los visitó.
  - finaliza cuando no haya más nodos por visitar

En los recorridos antes mencionados, ¿qué pasaría si el grafo fuera no conexo?

Árboles Abarcadores de Costo Mínimo (AAM)

Sea G = (V,E) un grafo con aristas ponderadas.
Un árbol abarcador para G es un árbol libre (grafo conexo acíclico) que conecta todos los vértices de V, su costo es la suma de los costos de las aristas del árbol.
Un AAM para G es un árbol abarcador que conecta todos los nodos de V con mínimo costo.

Algoritmos para obtener AAM

- ✓ Algoritmo de **Kruskal**

Se considera la inclusión de las aristas en orden creciente por costo. Una arista se incluye sólo si no forma ciclo (un ciclo implicaría dos caminos entre un mismo par de vértices).

Paso 1: Generar un grafo T=(V,∅) constituido por todos los vértices de V y ninguna arista.

Paso 2: Para construir componentes conexas cada vez más grandes, se examinan las aristas de E en orden creciente de costo. Se agrega la arista si conecta dos vértices que se encuentran en componentes conexas distintas; en caso contrario, se descarta la arista.

Cuando todos los vértices están en una misma componente conexa, T será el AAM

- ✓ Algoritmo de **Prim**

Se inicia un conjunto U con un vértice cualquiera de V. Se consideran las aristas que conectan un vértice de U con un vértice de V-U, de ellas se elige la de menor costo.

Paso 1: Asignar a un conjunto U de vértices un vértice cualquiera, a partir de él se construirá el AAM

Paso 2: localizar la arista (u,v) con menor costo que conecta un vértice de U con un vértice de V-U, agregar v a U y la arista al árbol

El proceso termina cuando U=V

DIGRAFOS

Problemas de los caminos más cortos

Sea G = (V,E) un digrafo conexo con aristas ponderadas.

- ✓ Algoritmo de **Dijkstra** (caminos mínimos con un solo origen)

Se define S como el *conjunto de vértices cuya distancia más corta desde el origen ya se conoce*. En principio S contiene sólo el origen. En cada paso se agrega algún vértice a S, cuya distancia en la más corta posible pasando sólo a través de vértices de S

Paso 1: Incluir en S el origen.

Paso 2: Calcular distancias (D[w]) desde el origen a cada uno de los vértices que no se encuentran en S (w ε V-S).

Paso 3: Elegir el vértice w (ε V-S) tal que D[w] sea mínima. Agregar w a S

Paso 4: Ir a Paso 2 hasta que S=V

- ✓ Algoritmo de **Floyd** (caminos mínimos entre todos los pares de vértices)

```
Inicializar matriz A
for (k = 1; k <=N ; k++)
   for (i = 1; i<=N ; i++)
      for (j = 1; j <=N ; j++)
          if (A[i,k] + A[k,j] < A[i,j])
              A[i,j] = A[i,k] + A[k,j];
```

- ✓ Algoritmo de **Warshall** (existencia de caminos entre todos los pares de vértices)

```
Inicializar matriz A
for (k = 1; k <=N ; k++)
   for (i = 1; i<=N ; i++)
      for (j = 1; j <=N ; j++)
          if (A[i,k] + A[k,j] !=0 && A[i,k] + A[k,j] != infinito)
              A[i,j] = 1;
```