UNIVERSIDADE DA CORUÑA

FACULTAD DE INFORMÁTICA
GRADO EN CIENCIA E INGENIERÍA DE DATOS

Diseño y Análisis de Algoritmos
Práctica 3: Árboles expandidos mínimos

29 de octubre de 2024

Dado un grafo no dirigido y ponderado con pesos únicamente positivos, se pide *implementar los algoritmos de Kruskal y Prim* de manera que computen el árbol expandido mínimo a partir de un grafo completo. Por comodidad:

- Para el algoritmo de Kruskal trabajaremos sobre un grafo, G, representado como una tupla (V, E), donde $V = \{1, ..., n\}$ es un conjunto de números que identifica a los vértices del grafo y E es un conjunto de tuplas (u, v, w), donde $u, v \in V$ y $w > 0$ (representando el peso de la arista entre los dos vértices u y v).
- Para el algoritmo de Prim, trabajaremos con la correspondiente representación matricial del grafo G, G^M, donde $G^M_{[i][j]}$ indica el peso de la arista que conecta los vértices i y j. 0 indica que no hay conexión entre los nodos.

Kruskal Algorithm

```
function kruskal (V,E): T
    Sort the elements in E according to w;
    T := {};
    S := [1, 2, .., n];
    repeat
        a := select the tuple (u,v,w) that has not been yet analized and that minimizes w;
        u_S := find(u, S); # find is a function that finds the set of S where u is
        v_S := find(v, S);
        if u_S ≠ v_S:
            merge(v, u_S, S);
            T := T ∪ {a};
    until |T| = n − 1
    return T;

function find (u, S): u_S
    u_S := S[u];
    while u_S != S[u_S]
        u_S := S[u_S];
    return u_S

function merge (u, v_S, S):
    u_S := S[u];
    while u_S != S[u_S]
        aux := S[u_S];
        S[u_S] := v_S;
        u_S := aux;
    S[u_S] := v_S;
```

Prim Algorithm

```
function prim (M[1...n,1...n]): T
    mindist := [0...0]; #size n
    nearest := [1...1]; #size n
    T := {};
    for i := 2 to n:
        mindist[i] := M[i,1];
    repeat n-1 times
        min := ∞;
        for j := 2 to n:
            if 0 <mindist[j] < min;
                min := mindist[j]
                k := j
        a := (nearest[k], k, M[nearest[k],k]);
        T := T ∪ {a};
        mindist[k] := 0;
        for j := 2 to n:
            if 0 <M[j,k] <mindist[j]:
                mindist[j] := M[j,k];
                nearest[j] := k;
    return T
```

Se pide:

1. Implementar en Python los algoritmos de Kruskal y Prim utilizando como base los pseudocódigos explicados en clase e incluidos en este documento.
2. Validar el correcto funcionamiento de la implementaciones. En las figuras 1 y 2 se proponen dos casos de prueba. A mayores, la figura 3 muestra la representación en el correspondiente programa Python. Es posible considerar más ejemplos en caso de que se crea conveniente para la correcta depuración del código. Si se considera útil para ello, es posible utilizar la funciones `to_adjacency_matrix` o `to_vertices_and_edges` en la figura 4 para transformar de un determinado formato al otro.
3. Calcular empíricamente la complejidad computacional del algoritmo. Para generar *grafos completos no dirigidos* y de acuerdo al formato requerido, se dispone del código `create_graph` en la figura 4. Como referencia, puede tomar los valores [20, 40, 80, 160, 320, 640, 1280, 2560] para generar grafos completos de n nodos. Igualmente se realizará una comprobación empírica utilizando una cota subestimada y otra sobrestimada.

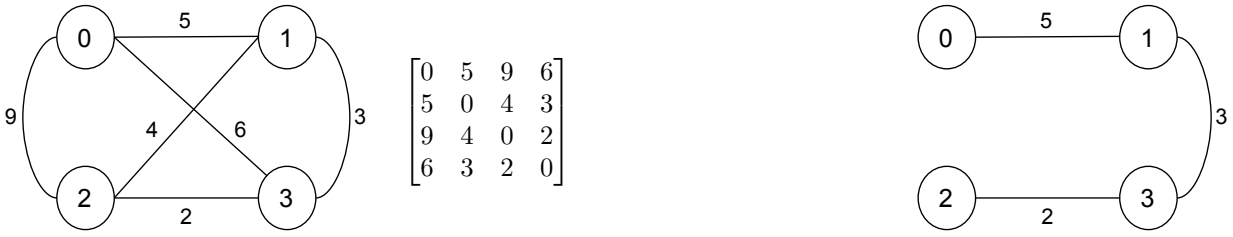
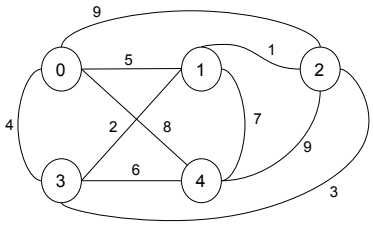


Figura 1: Ejemplo de un grafo completo no dirigido, su correspondiente representación matricial, y su árbol expandido mínimo.



$$\begin{bmatrix} 0 & 5 & 9 & 4 & 8 \\ 5 & 0 & 1 & 2 & 7 \\ 9 & 1 & 0 & 3 & 9 \\ 4 & 2 & 3 & 0 & 6 \\ 8 & 7 & 9 & 6 & 0 \end{bmatrix}$$

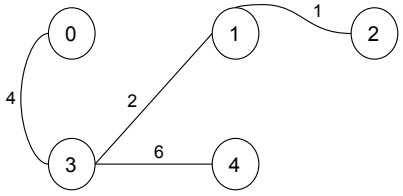


Figura 2: Otro ejemplo de un grafo completo no dirigido, su correspondiente representación matricial, y su árbol expandido mínimo.

```
V1 = {0,1,2,3}
E1 = {(0, 2, 9), (3, 2, 2), (0, 3, 6), (1, 2, 4), (0, 1, 5), (1, 3, 3)}
M1 = to_adjacency_matrix(V1,E1)
MST1_gold = {(3, 2, 2), (0, 1, 5), (1, 3, 3)}
MST1_prim = prim(M1)
MST1_kruskal = kruskal(V1,E1)

V2 = {0,1,2,3,4}
E2 = {(3, 4, 6), (1, 2, 1), (0, 2, 9), (1, 4, 7), (0, 3, 4), (3, 1, 2), (2, 3, 3), (2, 4, 9), (0, 4, 8), (0, 1, 5)}
M2 = to_adjacency_matrix(V2,E2)
MST2_gold = {(3, 4, 6), (1, 2, 1), (0, 3, 4), (3, 1, 2)}
MST2_prim = prim(M2)
MST2_kruskal = kruskal(V2,E2)
```

Figura 3: Definición en Python de los dos grafos ilustrados en las figuras 1 y 2, y ejemplo de cómo utilizar las funciones `kruskal` y `prim` que se piden en la práctica.

```python
import numpy as np

def to_adjacency_matrix(V, E):

    M =  np.zeros((len(V), len(V)), dtype=int)

    for i,j,w in E:
        M[i][j] = w
        M[j][i] = w

    return M

def to_vertices_and_edges(M):

    rows, cols =  M.shape
    E = set([])
    V = set([])
    for i in range(rows):
        V.add(i)
        for j in range(i+1, cols):
            E.add((i,j,M[i][j]))
    return (V,E)


def create_graph(n, max_distance=50, adjacency_matrix=False):

    a = np.random.randint(low=1, high=max_distance, size=(n,n))
    M = np.tril(a,-1) + np.tril(a, -1).T

    if adjacency_matrix:
        return M

    return to_vertices_and_edges(M)
```

Figura 4: Código de ejemplo para crear grafos completos aleatorios.

- La fecha de entrega límite es el día **15 de noviembre de 2024** a las 23:50 horas.
- Será necesario depositar en la página de la asignatura en el CampusVirtual, el fichero Python que contiene el código fuente de la práctica y el informe con el estudio de complejidad.
- Revise detenidamente el documento PlantillaPracticas para saber qué se va a evaluar.
- Es suficiente con que uno de los integrantes del equipo de práctica deposite el contenido en el CampusVirtual. El nombre de **todos** los integrantes debe figurar en el encabezado de ambos documentos a entregar.
- Es obligatorio realizar una defensa de la práctica en la clase de prácticas posterior a la entrega.