Recuperatorio Parcialito N°4

Ivan Litteri - 106223

08/03/2021

Ejercicio 1

Consigna

a) Aplicar el algoritmo de Prim al siguiente grafo, empezando por el vértice E (grafo no dibujado por tiempo).

b) Si los pesos de un grafo son todos diferentes, ¿obtendríamos siempre el mismo árbol de tendido mínimo aplicando el Algoritmo de Prim?.

Resolución

El algoritmo de Prim resumido es:

1. Empezar por un vértice aleatorio.
2. Encolar todas las aristas (cuyo destino no esté visitado) en el heap.
3. Desencolar el heap (la función de comparación usa el peso de la arista).
4. Repetir paso 2).

a) Comenzando por el vértice E del grafo.

En el heap encolo sus adyacentes en el heap y actualizo visitados:

$$Heap = [(E-A,2),(E-G,3),(E-C,3),(E-I,3)]$$

$$Visitados = \{E\}$$

$$T = \{E\}$$

- Desencolo del heap a $(E-A,2)$. A no está visitado entones actualizo visitados y lo agrego a la solución:

$$Heap = [(E-G,3),(E-C,3),(E-I,3)]$$

$$Visitados = \{E,A\}$$

$$T = \{(E-A,2)\}$$

- Miro los adyacentes de A no visitados y los agrego al heap:

$$Heap = [(E-G,3),(E-C,3),(E-I,3),(A-B,5),(A-C,7),(A-F,8)]$$

$$Visitados = \{E,A\}$$

- Desencolo del heap, hay 3 posibilidades, supongo que sale $(E-G,3)$. G no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(E-C,3),(E-I,3),(A-B,5),(A-C,7),(A-F,8)]$$

$$Visitados = \{E,A,G\}$$

$$T = \{(E-A,2),(E-G,3)\}$$

- Miro los adyacentes de G, y como están todos visitados, no cambio nada.
- Desencolo del heap, hay 2 posibilidades, supongo que sale $(E-C,3)$. C no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(E-I,3),(A-B,5),(A-C,7),(A-F,8)]$$

$$Visitados = \{E,A,G,C\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3)\}$$

- Miro los adyacentes de C no visitados y los agrego al heap:

$$Heap = [(E-I,3),(C-H,3),(A-B,5),(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C\}$$

- Desencolo del heap, hay 2 posibilidades, supongo que sale $(E-I,3)$. I no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(C-H,3),(A-B,5),(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3)\}$$

- Miro los adyacentes de I, y como están todos visitados, no cambio nada.
- Desencolo del heap sale $(C-H,3)$. H no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(A-B,5),(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3)\}$$

- Miro los adyacentes de H, y como están todos visitados, no cambio nada.
- Desencolo del heap sale $(A-B,5)$. B no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5)\}$$

- Miro los adyacentes de B y agrego a los vértices cuyo destino no este visitado:

$$Heap = [(B-F,1),(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B\}$$

- Desencolo del heap sale $(B-F,1)$. F no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B,F\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5),(B-F,1)\}$$

- Miro los adyacentes de F y agrego a los vértices no visitados:

$$Heap = [(F-D,6),(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B,F\}$$

- Desencolo del heap sale $(F-D,6)$. D no está visitado entonces actualizo visitados y lo agrego a la solución:

$$Heap = [(A-C,7),(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B,F,D\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5),(B-F,1),(F-D,6)\}$$

- Miro los adyacentes de D, ya están todos visitados entonces no hago nada.
- Desencolo del heap $(A-C,7)$, ambos visitados entonces no hago nada.

$$Heap = [(A-F,8),(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B,F,D\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5),(B-F,1),(F-D,6)\}$$

- Desencolo del heap $(A-F,8)$, ambos visitados entonces no hago nada.

$$Heap = [(C-D,8)]$$

$$Visitados = \{E,A,G,C,I,H,B,F,D\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5),(B-F,1),(F-D,6)\}$$

- Desencolo del heap $(C-D,8)$, ambos visitados entonces no hago nada.

$$Heap = []$$

$$Visitados = \{E,A,G,C,I,H,B,F,D\}$$

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5),(B-F,1),(F-D,6)\}$$

- Destruyo el heap, y me quedó el MST:

$$T = \{(E-A,2),(E-G,3),(E-C,3),(E-I,3),(C-H,3),(A-B,5),(B-F,1),(F-D,6)\}$$

b) Si los pesos de un grafo son todos diferentes, obtendríamos siempre el mismo árbol de tendido mínimo aplicando el Algoritmo de Prim, porque se irían desencolando y agregando al MST siempre en el mismo orden, generando el mismo MST siempre.

Ejercicio 2

Consigna

Implementar un algoritmo que reciba un grafo, un vértice v y otro w, y determine (utilizando backtracking) la cantidad de caminos diferentes que hay desde v a w.

Resolución

```
def _todos_los_caminos(grafo, v, w, visitados, camino, caminos) -> None:
    # Si se visitan todos, termino la busqueda de caminos
    if len(visitados) == len(grafo):
        return

    visitados[v]= True
    camino.append(v)

    # Si el inicio es igual al fin, agrego un camino.
    if v == w:
        caminos.append(camino)
    # Sino visito los adyacentes.
    else:
        for x in grafo.adyacentes[v]:
            if x not in visitados[x]:
                _todos_los_caminos(grafo, x, w, visitados, camino, caminos)

    # Vuelvo para atras para ver si encuentro otro camino.
    camino.pop()
    visitados.pop(v)

def todos_los_caminos(grafo, v, w) -> int:
    caminos =  []
    _todos_los_caminos(grafo, v, w, {}, [], caminos)
    return len(caminos)
```

Ejercicio 3

Consigna

Dado un grafo no dirigido y pesado (todos pesos positivos), se desea averiguar el camino mínimo de un vértice v hacia todos los demás vértices del grafo. Se desea saber cuántos caminos mínimos hay hacia cada vértice (considerando que pueden haber varios caminos de mismo peso para llegar a un determinado vértice). Explicar cómo modificarías el algoritmo de Dijkstra para que, además de las representaciones de padres (para reconstruir caminos) y distancias, obtenga la cantidad de caminos mínimos de cada vértice (definimos que para el origen esa cantidad es 0). La complejidad del algoritmo debería quedar tal cual el algoritmo original. Justificar la complejidad del nuevo algoritmo. Para el siguiente grafo, la cantidad de caminos mínimos desde A son: $\{A:0, B:1, C:1, F:2, E:1, D:3\}$

Resolución

```
def camino_minimo(grafo, origen):
    dist = {}
    padre = {}
    cant_caminos_minimos = {} # Agregado
    for v in grafo:
        distancia[v] = infinito
        cant_caminos_minimos[v] = 0 # Agregado
    dist[origen] = 0
    padre[origen] = None
    q = Heap()
    q.encolar(origen, 0)
    while not q.esta_vacia():
        v = q.desencolar()
        for w in grafo.adyacentes(v):
            # Si empata la distancia incrementa en uno la cantidad.
```

```
            if dist[v] + grafo.peso_union(v, w) == dist[w]:
                cant_caminos_minimos[w] += 1 # Agregado
            # Cuando encuentro un mejor camino, simplemente actualizo la cantidad a 1.
            if dist[v] + grafo.peso_union(v, w) < dist[w]:
                dist[w] = dist[v] + grafo.peso_union(v, w)
                cant_caminos_minimos[w] = 1 # Agregado
                padre[w] = v
                q.encolar(w, dist[w])
    # Agrego que devuelta la cantidad de caminos minimos.
    return padre, distancia, cant_caminos_minimos # Agregado

''' Complejidad
    La complejidad se mantiene, siendo esta O(E log V) ya que solo agrego operaciones
    constantes.
'''
''' Modificacion
    Agrego un diccionario con cada vertice con camino minimo inicial en 0 (el origne va a
    quedar en 0). Cuando encuentro un camino que sea igual de bueno que el que ya tenia como
     minimo, incremento en 1 la cantidad de caminos minimos al vertice de esa iteracion. Si
    encuentra un camino mas largo no hace nada, y si encuentra un camino mejor pone en uno
    la cantidad.
'''
```