

Ejercicios

Rodrigo Vazquez

01/01/01

Ejercicios

Prácticos

Dado un grafo (viene como : lista de listas, matriz de incidencia, matriz de adyacencia, dibujo). Encontrar el árbol de tendido mínimo.

PRIM

- Empieza de algun vertice y a partir de las aristas que va descubriendo de ese vertice
- Logra que sea minimo usando las aristas de menor valor dentro de las conocidas. Usa un heap
- 1. Elegir vertice y agregar al heap todas las aristas adyacentes al vertice.
- 2. Desencolar y ver el vertice que se descubrio (fué visitado?). Repetir con nuevo vertice
 - Si desencolamos algo ya visitado, no hay que agregar la arista hacia un vertice ya visitado
- 3. Repetir hasta que esten todos visitados.
- 4. Desencolar lo que queda en el heap.?

KRUSKAL

- Ordenar aristas por peso. -> siempre tratar de incluir las aristas de menor peso.
- Evita generar ciclos.
- 1. Ordenar las aristas de menor a mayor y desencolar arista agregada a la lista
- 2. *Union/Find*-> find de 2 vertices y preguntar si estan en el mismo arbolito. Sino **union**
- 3. Repetir hasta que la cantidad de aristas sea uno menos que la cantidad de vertices.

Backtracking

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

```
def caminos_posibles(grafo,v,w):
    visitados = {}
    return encontrar_caminos_posibles(grafo,v,w,visitados)

def encontrar_caminos_posibles(grafo,v,w,visitados):
    visitados[v] = True # paso para adelante
```

```

contador = 0
for ady in grafo.adyacentes(v):
    if ady not in visitados:
        cont += encontrar_caminos_posibles(grafo,v,w,visitados)

visitados.remove(v) # paso para atras
return contador

```

Implementar un algoritmo que reciba un grafo, un vertice v otro w y un valor k , y determine (utilizando backtracking) la cantidad de caminos simples diferentes que hay desde v a w , siempre y cuando el peso del camino sea menor a k .

```

def caminos_posibles(grafo,v,w,k, peso):

    visitados = {}
    return encontrar_caminos_posibles(grafo,v,w,visitados,k,peso)

def encontrar_caminos_posibles(grafo,v,w,visitados,k,peso):

    visitados[v] = True # paso para adelante
    contador = 0
    for ady in grafo.adyacentes(v):
        peso += obtener_peso(v,ady)
        if ady not in visitados and peso <= k:
            cont += encontrar_caminos_posibles(grafo,v,w,visitados,k,peso)
        peso -= obtener_peso(v,ady)
    visitados.remove(v) # paso para atras
    return contador

```

Teoricos

Al aplicar sobre un grafo el algoritmo de Dijkstra para encontrar caminos minimos desde un vertice v cualquiera, se obtiene un árbol definido por el diccionario de padres (que permite reconstruir dichos caminos minimos). Dicho árbol, es siempre de tendido minimo?

No, porque Dijkstra toma el costo minimo de manera loca, vertice a vertice, y no necesariamente

Al obtener un arbol de tendido minimo de un grafo, se asegura que la suma de los pesos de las aristas sean minimos. Es posibles utilizar el arbol de tendido minimo para encontrar el camino minimo entre dos pares de vertices cuales queira?

No, porque tal ves podes encontrar un camino minimo por aristas que no se encuentran en ese

Si un grafo no es pesado, se puede utilizar el algoritmo de Dijkstra para obtener los caminos minimos de dicho grafo?

Si, porque podes poner todos los pesos a 1 y ahi usar Dijkstra. Aunque la opcion optima seria

Maximizacion de Flujo

Qué algoritmo nos sirve para resolver el problema de Maximizacion de Flujo?

Ford Fulkerson

Qué condiciones debe cumplir el grafo para aplicar el algoritmo de Ford Fulkerson

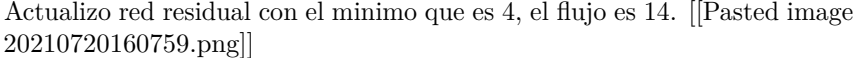
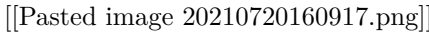
- Tenia que tener una sola fuente y un solo sumidero.
- No tiene que haber bucles.
- No tiene que haber ciclo de 2 vertices (aristas antiparalelas).

Algoritmo de Ford Fulkerson

1. Creo red residual
2. camino = bfs(fuente, sumidero) # en red residual
3. Si no hay camino termino.
4. Si hay camino, saco el valor minimo de todas las aristas del camino.
5. Actualizo las aristas del camino (sumando o restando segun el sentido del bfs). # en red residual
6. Vuelvo a paso 2.

Seguimiento

- Parto desde una red residual [[Pasted image 20210720155224.png]]
- Busco camino e indico flujo -> minimo costo [[Pasted image 20210720160035.png]]
- El flujo es 4 -> **uso**. Sacar arista del minimo y poner una que vuelve con el uso en todas los vertices del camino. Actualizo las aristas de ida con la resta entre el costo- uso [[Pasted image 20210720160202.png]]
- Busco otro camino minimo. [[Pasted image 20210720160405.png]]
- Repito proceso con el minimo que es 6, el flujo es 10. [[Pasted image 20210720160521.png]]
- Busco otro camino minimo. [[Pasted image 20210720160631.png]]

- Actualizo red residual con el minimo que es 4, el flujo es 14. 
- Con otro camino, actualizo la red con minimo que es 5, el flujo es 19. 
- Aca termina porque no hay mas camino minimo.
- Este ejemplo no usa listas de retorno.

Algoritmo de Dijkstra

1. Inicializar los pesos de todos los vertices con infinito
2. Encolar en el heap el origen (peso,clave).
3. Encolamos los adyacentes con costo del origen + costo de llegar a arista y el origen que
4. Desencolo arista (el mas chico de los pesos).
5. Encolo los adyacentes con costo actualizado desde origen. -> Usar diccionario de costos.
6. Repetir hasta que esten todos visitados.

Problemas Avanzados

Ya vistos - Lenguaje Alien - Cambio de sentido de aristas - Biblioteca - Arbitrage
 ### Arbitraje Queremos encontrar un camino ta que $C_1.C_2.C_3...C_K > 1$
