Grafos
Minimum Spanning Tree & Shortest Paths

MSc Edson Ticona Zegarra

Taller avanzado 2025

Contenido

Contenido

Grafo de expansión

- Dado un grafo $G = (V, E)$ y un subgrafo $G' = (V', E')$, tal que $G' \subset G$, se dice que G' es un subgrafo de expansión si todos los vértices de V' se expanden a todo G, es decir si $V' = V$

Grafo de expansión

- Dado un grafo $G = (V, E)$ y un subgrafo $G' = (V', E')$, tal que $G' \subset G$, se dice que G' es un subgrafo de expansión si todos los vértices de V' se expanden a todo G, es decir si $V' = V$
- Llamamos a un grafo G de *edge-maximal* con cierta propiedad si G cumple con la propiedad pero el grafo $G + xy$ no, para cualquier par de vértices $x, y \in G$

Grafo de expansión

- Dado un grafo $G = (V, E)$ y un subgrafo $G' = (V', E')$, tal que $G' \subset G$, se dice que G' es un subgrafo de expansión si todos los vértices de V' se expanden a todo G, es decir si $V' = V$
- Llamamos a un grafo G de *edge-maximal* con cierta propiedad si G cumple con la propiedad pero el grafo $G + xy$ no, para cualquier par de vértices $x, y \in G$
- En general hablamos de *maximal* o *minimal* con cierta propiedad, se refiere a la relación de subgrafo

Árbol de expansión mínimo

- Llamamos de árbol de expansión a todo subgrafo $T \in G$ tal que T es un árbol y T es un subgrafo de expansión

Árbol de expansión mínimo

- Llamamos de árbol de expansión a todo subgrafo $T \in G$ tal que T es un árbol y T es un subgrafo de expansión
- Para un grafo con pesos, entre todos los posibles árboles de expansión, se llama a T de árbol de expansión mínimo si la suma de los pesos de las aristas de T es la menor de todas

Árbol de expansión mínimo

- Llamamos de árbol de expansión a todo subgrafo $T \in G$ tal que T es un árbol y T es un subgrafo de expansión
- Para un grafo con pesos, entre todos los posibles árboles de expansión, se llama a T de árbol de expansión mínimo si la suma de los pesos de las aristas de T es la menor de todas
- Formalmente,

$$\min \sum_{e \in E} w(e)$$

Árbol de expansión mínimo

- Llamamos de árbol de expansión a todo subgrafo $T \in G$ tal que T es un árbol y T es un subgrafo de expansión
- Para un grafo con pesos, entre todos los posibles árboles de expansión, se llama a T de árbol de expansión mínimo si la suma de los pesos de las aristas de T es la menor de todas
- Formalmente,

$$\min \sum_{e \in E} w(e)$$

- donde $w(e)$ representa el peso de la arista e

Minimum Spanning Tree (MST)

- El problema del árbol de expansión mínimo, en inglés *minimum spanning tree (MST)* admite solución por un algoritmo greedy

Minimum Spanning Tree (MST)

- El problema del árbol de expansión mínimo, en inglés *minimum spanning tree (MST)* admite solución por un algoritmo greedy
- La idea es construir el MST agregando arista por arista, siempre que sea una arista *segura* de peso mínimo

Minimum Spanning Tree (MST)

- El problema del árbol de expansión mínimo, en inglés *minimum spanning tree (MST)* admite solución por un algoritmo greedy
- La idea es construir el MST agregando arista por arista, siempre que sea una arista *segura* de peso mínimo
- Una arista segura es aquella que mantiene la propiedad

Algoritmo de Prim

- El algoritmo de Prim inicia de un vértice arbitrario, dicho vértice provee diversas opciones en cuanto a cuál arista agregar a continuación

Algoritmo de Prim

- El algoritmo de Prim inicia de un vértice arbitrario, dicho vértice provee diversas opciones en cuanto a cuál arista agregar a continuación
- Siempre se agrega la menor disponible al componente creado hasta el momento

Prim

input : $G = (V, E)$ es el grafo
output: T es el MST
$Q.insert(e)$;   `// Q cola de prioridades en base al peso`
while $!Q.empty()$ **do**
  $e \leftarrow Q.top()$
  `// e es seguro si no crea ciclos`
  if $e.isSafe()$ **then**
    $T.push(e)$
    $Q.push(E(e))$
  end
end

Prim

- Complejidad $O(E \log E) = O(E \log V)$

Algoritmo de Kruskal

- El algoritmo de Kruskal inicia por la arista de menor peso

Algoritmo de Kruskal

- El algoritmo de Kruskal inicia por la arista de menor peso
- Se continue con la siguiente arista de menor peso, si dicha arista no pertenece al mismo componente se crea otro componente

Algoritmo de Kruskal

- El algoritmo de Kruskal inicia por la arista de menor peso
- Se continue con la siguiente arista de menor peso, si dicha arista no pertenece al mismo componente se crea otro componente
- Eventualmente una arista va a unir un par de componentes desconectados

Algoritmo de Kruskal

- El algoritmo de Kruskal inicia por la arista de menor peso
- Se continue con la siguiente arista de menor peso, si dicha arista no pertenece al mismo componente se crea otro componente
- Eventualmente una arista va a unir un par de componentes desconectados
- Si una arista crea un ciclo es ignorada

Kruskal

input : $G = (V, E)$ es el grafo
output: T es el MST

```
Sort(E)
for e ∈ E do
    if FindSet(e.u)! = FindSet(e.v) then
        T.push(e)
        Union(e.u, e.v)
    end
end
```

Kruskal

- Complejidad $O(E \log V)$

Kruskal

- Complejidad $O(E \log V)$
- Igual que Prim

Conjuntos Disjuntos

- En un *conjunto* no hay elementos repetidos

Conjuntos Disjuntos

- En un *conjunto* no hay elementos repetidos
- *Conjuntos disjuntos* es un grupo de conjuntos sin elementos en común

Conjuntos Disjuntos

- En un *conjunto* no hay elementos repetidos
- *Conjuntos disjuntos* es un grupo de conjuntos sin elementos en común
- Estructura de datos para manejar conjuntos disjuntos de manera eficiente

Conjuntos Disjuntos

- En un *conjunto* no hay elementos repetidos
- *Conjuntos disjuntos* es un grupo de conjuntos sin elementos en común
- Estructura de datos para manejar conjuntos disjuntos de manera eficiente
- Para el propósito de Kruskal necesitamos dos operaciones básicas: `FindSet` y `Union`

Conjuntos Disjuntos

- En un *conjunto* no hay elementos repetidos
- *Conjuntos disjuntos* es un grupo de conjuntos sin elementos en común
- Estructura de datos para manejar conjuntos disjuntos de manera eficiente
- Para el propósito de Kruskal necesitamos dos operaciones básicas: `FindSet` y `Union`
- El objetivo es poder determinar, de manera eficiente, a que conjunto pertenece cierto elemento y unir dos conjuntos

Conjuntos Disjuntos

- En un *conjunto* no hay elementos repetidos
- *Conjuntos disjuntos* es un grupo de conjuntos sin elementos en común
- Estructura de datos para manejar conjuntos disjuntos de manera eficiente
- Para el propósito de Kruskal necesitamos dos operaciones básicas: `FindSet` y `Union`
- El objetivo es poder determinar, de manera eficiente, a que conjunto pertenece cierto elemento y unir dos conjuntos
- La implementación trivial se puede hacer con arreglos, una implementación más eficiente puede ser con árboles

Conjuntos Disjuntos: `FindSet`

- Cada conjunto tiene un elemento *representativo*

Conjuntos Disjuntos: `FindSet`

- Cada conjunto tiene un elemento *representativo*
- Dado un elemento x, la idea es saber a cuál conjunto pertenece, es decir, hallar su elemento representativo

Conjuntos Disjuntos: `Union`

- Al unir dos conjuntos, todos los elementos de ambos conjuntos terminan con el mismo elemento representativo

Conjuntos Disjuntos: `Union`

- Al unir dos conjuntos, todos los elementos de ambos conjuntos terminan con el mismo elemento representativo
- Para optimizar esta operación se utiliza el *rank* y solo tiene uso cuando se utiliza una representación de árbol

Conjuntos Disjuntos: `Union`

- Al unir dos conjuntos, todos los elementos de ambos conjuntos terminan con el mismo elemento representativo
- Para optimizar esta operación se utiliza el *rank* y solo tiene uso cuando se utiliza una representación de árbol
- El *rank* mide la altura (límite superior) del árbol de cada conjunto

Conjuntos Disjuntos: `Union`

- Al unir dos conjuntos, todos los elementos de ambos conjuntos terminan con el mismo elemento representativo
- Para optimizar esta operación se utiliza el *rank* y solo tiene uso cuando se utiliza una representación de árbol
- El *rank* mide la altura (límite superior) del árbol de cada conjunto
- Dos *heurísticas* son conocidas: *union by rank* y *path compression*

Conjuntos Disjuntos: heurísticas

- *union by rank* hace que al unir dos conjuntos, el de menor rango apunte a aquel con mayor rango

Conjuntos Disjuntos: heurísticas

- *union by rank* hace que al unir dos conjuntos, el de menor rango apunte a aquel con mayor rango
- *path compression* hace que cada nodo apunte directamente al nodo raíz

Comparación

- abd

Comparación

- abd
- cd

Caminos mínimos

- Dado un grafo con pesos, se busca minimizar la suma de los pesos de las aristas del camino entre s y t

Caminos mínimos

- Dado un grafo con pesos, se busca minimizar la suma de los pesos de las aristas del camino entre s y t
- Formalmente,

$$\min \sum_{e \in P} w(e)$$

Caminos mínimos

- Dado un grafo con pesos, se busca minimizar la suma de los pesos de las aristas del camino entre s y t
- Formalmente,

$$\min \sum_{e \in P} w(e)$$

- Tal que P es un camino entre s y t

Relajación

- Dadar la relajación

Algoritmo de Bellman-Ford

- Dado un grafo con pesos, se busca minimizar la suma de los pesos de las aristas del camino entre s y t

Algoritmo de Bellman-Ford

- Dado un grafo con pesos, se busca minimizar la suma de los pesos de las aristas del camino entre s y t
- Formalmente,

$$\min \sum_{e \in P} w(e)$$

Algoritmo de Bellman-Ford

- Dado un grafo con pesos, se busca minimizar la suma de los pesos de las aristas del camino entre s y t
- Formalmente,

$$\min \sum_{e \in P} w(e)$$

- Tal que P es un camino entre s y t