Implementaciones Prim y Kruskal

Julián Braier

FCEN - UBA

1c 2023

Refrescando la memoria: AGM

Definición 7.

- Sea $T = (V, X)$ un árbol y $l : X \to \mathbb{R}$ una función que asigna costos a las aristas de T. Se define el ***costo*** de T como $l(T) = \sum_{e \in T} l(e)$.
- Dado un grafo $G = (V, X)$ un ***árbol generador mínimo*** de G, $AGM(G) = T$, es un árbol generador de G de mínimo costo, es decir
$$l(T) \leq l(T') \; \forall T' \text{ árbol generador de } G.$$
- Dado un grafo pesado en las aristas, $G = (V, X)$, el problema de ***árbol generador mínimo*** consiste en encontrar un AGM de G.

[1]

[1] Tomado de apuntes hechos por Paula Zabala en la virtualidad.

Cantidad de AGs

- La cantidad de AGs en un grafo pueden ser muchos (MUCHOS!).

Cantidad de AGs

- La cantidad de AGs en un grafo pueden ser muchos (MUCHOS!).
- En un grafo completo de n vértices hay n^{n-2} AGs diferentes. (Cayley's formula)

Cantidad de AGs

- La cantidad de AGs en un grafo pueden ser muchos (MUCHOS!).
- En un grafo completo de n vértices hay n^{n-2} AGs diferentes. (Cayley's formula)
- Por suerte es resoluble con algoritmos golosos.

Refrescando la memoria: Algoritmos

[2] Para ver los algoritmos animados: `https://visualgo.net/en/mst`

[3] Prim puede mejorarse a $O(m + n \; log \; n)$ usando Fibonacci Heap.

Refrescando la memoria: Algoritmos

Figure: Prim y Kruskal

[2]

- Complejidades: (ambos pueden implementarse en ambas complejidades)
  1. $O(m\ log\ n)$[3].
  2. $O(n^2)$

[2] Para ver los algoritmos animados: `https://visualgo.net/en/mst`

[3] Prim puede mejorarse a $O(m + n\ log\ n)$ usando Fibonacci Heap.

Refrescando la memoria: Algoritmos

Figure: Prim y Kruskal

[2]

- Complejidades: (ambos pueden implementarse en ambas complejidades)
  1. $O(m \ log \ n)$[3].
  2. $O(n^2)$ para grafos densos.

[2] Para ver los algoritmos animados: `https://visualgo.net/en/mst`

[3] Prim puede mejorarse a $O(m + n \ log \ n)$ usando Fibonacci Heap.

Prim

- Invariante: tenemos un árbol de i aristas que es subgrafo de algún AGM.

Prim

- Invariante: tenemos un árbol de i aristas que es subgrafo de algún AGM.
- Inicialización: empezamos con un sólo vértice v arbitrario.

Prim

- Invariante: tenemos un árbol de i aristas que es subgrafo de algún AGM.
- Inicialización: empezamos con un sólo vértice v arbitrario.
- Iteración: agregamos, de las aristas que podemos agregar y seguir teniendo un árbol, la más barata.

Implementamos Prim

- Versión $O(n^2)$.

Implementamos Prim

- Versión $O(n^2)$.
- Versión $O(m \; lg \; n)$. (hacemos en clase o no dependiendo del tiempo)

Primer Espacio Publicitario: Deportes

https://www.instagram.com/deportesexactasuba/

Segundo Espacio Publicitario: Programación Competitiva

¿Qué es el Training Camp?

Se trata de un entrenamiento intensivo de 2 semanas para competencias de programación con una parte teórica durante la mañana y sesiones de práctica durante la tarde.

El Training Camp se organiza todos los años desde 2010 durante el receso invernal en distintas universidades de la Argentina. La edición 2023 será la decimocuarta edición se realizara en La Matanza

El Training Camp dura dos semanas desde 31 de Julio al 11 de Agosto, de lunes a viernes, de 9 a 20hs.

Ver preguntas frecuentes

[4]

[4] https://www.pc-arg.com/tc-arg

Segundo Espacio Publicitario: Programación Competitiva

Tomen links:

- `https://icpc.global/` (página oficial de la ICPC)
- `https://codeforces.com/` (página de competencias online)
- `https://cses.fi/` (Code Submission Evaluation System, por Antti el del libro)

Fin del Espacio Publicitario

Kruskal

- Invariante: tenemos un bosque generador de i aristas que es subgrafo de algún AGM.

Kruskal

- Invariante: tenemos un bosque generador de i aristas que es subgrafo de algún AGM.
- (Invariante alternativo: tenemos un bosque generador de i aristas que es mínimo entre los bosques de i aristas).

Kruskal

- Invariante: tenemos un bosque generador de i aristas que es subgrafo de algún AGM.
- (Invariante alternativo: tenemos un bosque generador de i aristas que es mínimo entre los bosques de i aristas).
- Inicialización: empezamos con todos los vértices y ninguna arista.

Kruskal

- Invariante: tenemos un bosque generador de i aristas que es subgrafo de algún AGM.
- (Invariante alternativo: tenemos un bosque generador de i aristas que es mínimo entre los bosques de i aristas).
- Inicialización: empezamos con todos los vértices y ninguna arista.
- Iteración: agregamos, de las aristas que podemos agregar y seguir teniendo un bosque (no generan ciclos), la más barata.

Kruskal: más detalle

- Podemos empezar por ordenar las aristas, para considerarlas en orden creciente de costo.

Kruskal: más detalle

- Podemos empezar por ordenar las aristas, para considerarlas en orden creciente de costo.
- Para cada arista nos fijamos si al agregarla al bosque se genera un ciclo.

Kruskal: más detalle

- Podemos empezar por ordenar las aristas, para considerarlas en orden creciente de costo.
- Para cada arista nos fijamos si al agregarla al bosque se genera un ciclo.
- Esto sucede sii sus extremos ya están en la misma componente conexa.

Kruskal: más detalle

- Podemos empezar por ordenar las aristas, para considerarlas en orden creciente de costo.
- Para cada arista nos fijamos si al agregarla al bosque se genera un ciclo.
- Esto sucede sii sus extremos ya están en la misma componente conexa.
- Dada una arista $e = uv$, cómo sabemos si u y v están en la misma cc?

Disjoint Set Union

Necesitamos una estructura de datos eficiente que ofrezca dos funciones:

Disjoint Set Union

Necesitamos una estructura de datos eficiente que ofrezca dos funciones:

1. *find*(v): dado un vértice nos dice a qué cc pertenece.

Disjoint Set Union

Necesitamos una estructura de datos eficiente que ofrezca dos funciones:

1. *find*(v): dado un vértice nos dice a qué cc pertenece.
2. *unite*(u, v): une las componentes a las que pertenecen u y v.