Clase 03 MA3705. 07 de septiembre 2020.

Tabla

- Generación de aristas
- Algoritmos de búsqueda
- Problema del árbol/bosque generador de peso mínimo (MST).

Generación de aristas

Generación (span) de aristas

Sea $G = (V, E)$ un grafo, $F \subseteq E$.

- Definimos el generado

$$\text{span}(F) = \{e = uv \in E \colon \exists u\text{-}v \text{ camino en } F\}$$

- Decimos que H (o $E(H)$) genera G si $E(G) \subseteq \text{span}(E(H))$.

Lemas

Si H genera a G entonces cada componentes de G está contenida en una componente de H.

Si $e = uv \in E(G)$ pertenece a algún ciclo de G entonces $G - e$ genera G.

Abuso de notación

Sea $F \subseteq \binom{V}{2}$.

Decimos que F es árbol, bosque, camino, ciclo, etc.

cuando (V, F) es árbol, bosque, camino, ciclo, etc.

Resultados útiles

Sea $G = (V, E)$ grafo $F \subseteq E$, $e \in E$.

Lemas

1. Si (V, F) tiene alguna arista, y todos los grados son pares, entonces tiene un ciclo.
2. Si F bosque entonces para cada $u, v \in V$ existe a lo más 1 u-v camino.
3. Si F bosque entonces $F + e$ tiene a lo más un ciclo $C(F, e)$
4. Si F bosque generador entonces (V, F) tiene las mismas componentes que G y cada una es un árbol.

Consecuencia importante

Intercambio

Sea F bosque generador de G.
Para todo $e \in E(G)$, existe $f \in F$ tal que $F - f + e$ es bosque generador de G.

De hecho, si $e \in F$ basta tomar $f =$
Si $e \notin f$ basta tomar ...

¿Cómo encontrar algún árbol generador en un grafo conexo?

(o un bosque generador en un grafo cualquiera)

Primera estrategia:
Partir de un vértice r y agregar aristas (sin crear ciclos) hasta que todos los vértices sean alcanzados.

Algoritmos de búsqueda

Algoritmo genérico de búsqueda / crecer un árbol desde un vértice:

Buscar árbol que cubra la CC de un vértice r dado:

Búsqueda genérico:

Entrada: $G = (V, E)$, $r \in V$
$U \leftarrow \{r\}$ `// Nodos visitados`
$F \leftarrow \emptyset$ `// Aristas de solución`
mientras $\delta(U) \neq \emptyset$ **hacer**
  Elegir $e = uv \in \delta(U)$, $u \in U$, $v \notin U$
  $U \leftarrow U + v$
  $F \leftarrow F + e$
fin
devolver (U, F)

Correctitud del algoritmo:

Búsqueda genérico:

Entrada: $G = (V, E)$, $r \in V$
$U \leftarrow \{r\}$, `// Nodos visitados`
$F \leftarrow \emptyset$ `// Aristas de solución`
mientras $\delta(U) \neq \emptyset$ **hacer**
    Elegir $e = uv \in \delta(U)$, $u \in U$, $v \notin U$
    $U \leftarrow U + v$
    $F \leftarrow F + e$
fin
devolver (U, F)

Casos especiales: Búsqueda en amplitud y búsqueda en profundidad:

Búsqueda en aplitud (BFS):

Entrada: $G = (V, E)$, $r \in V$
$U \leftarrow \{r\}$, $F \leftarrow \emptyset$
$\textsc{Cola} \leftarrow \emptyset$.
Insertar aristas de $\delta(r)$ en $\textsc{Cola}$.
mientras $\textsc{Cola} \neq \emptyset$. **hacer**
  Extraer primer e de $\textsc{Cola}$.
  si $e \in \delta(U)$, $u \in U$, $v \notin U$
   entonces
    $U \leftarrow U + v$
    $F \leftarrow F + e$
    Insertar aristas de $\delta(v)$ en $\textsc{Cola}$
  fin
fin
devolver (U, F)

Búsqueda en profundidad (DFS):

Entrada: $G = (V, E)$, $r \in V$
$U \leftarrow \{r\}$, $F \leftarrow \emptyset$
$\textsc{Pila} \leftarrow \emptyset$.
Insertar aristas de $\delta(r)$ en $\textsc{Pila}$.
mientras $\textsc{Pila} \neq \emptyset$. **hacer**
  Extraer última e de $\textsc{Pila}$.
  si $e \in \delta(U)$, $u \in U$, $v \notin U$
   entonces
    $U \leftarrow U + v$
    $F \leftarrow F + e$
    Insertar aristas de $\delta(v)$ en $\textsc{Pila}$
  fin
fin
devolver (U, F)

Ejemplo

Ejemplo

Problema del árbol/bosque generador de costo mínimo

Bosque generador de costo mínimo.

Dado G grafo, $c\colon E \to \mathbb{R}$.

Problema del bosque generador de costo mínimo

Encontrar $F \subseteq E$ bosque generador de costo $c(F) = \sum_{e \in F} c(e)$ mínimo

Si G es conexo, el problema coincide con el del árbol generador/cobertor de costo mínimo (MST).

Teorema de arista mínima de un corte

Sea $G = (V, E)$ conexo, $c \colon E \to \mathbb{R}$. $\mathrm{OPT} \subseteq E$ un MST.

Teorema: Sea $\emptyset \subsetneq U \subsetneq V$.

Si e es la arista de menor costo de $\delta(U)$ entonces e se puede introducir a OPT intercambiando una arista de $\delta(U) \cap \mathrm{OPT}$. Es decir

$$\exists f \in \delta(U) \cap \mathrm{OPT} \colon \quad \mathrm{OPT} - f + e \text{ es un MST}.$$

Corolario

Sea $G = (V, E)$ conexo, $c\colon E \to \mathbb{R}$. Queremos resolver MST.

$F \subseteq E$ se dice óptimo parcial si existe MST óptimo OPT con $F \subseteq \mathrm{OPT}$.

Corolario: Sean $F \subseteq E$ óptimo parcial y $U \subseteq V$ tal que $\delta(U) \cap F = \emptyset$.

Si e es la arista de menor costo de $\delta(U)$ entonces $F + e$ es óptimo parcial.

Algoritmos basados en esta idea

Algoritmo de Prim (Prim 1957 - Jarník 1930):

Elegir $r \in V$;
$U \leftarrow \{r\}$
$F \leftarrow \varnothing$
mientras $\delta(U) \neq \varnothing$ **hacer**
  Sea $e = uv \in \delta(U),\ u \in U,\ v \notin U$ tal que e es la arista de menor peso en $\delta(U)$
  $U \leftarrow U + w$
  $F \leftarrow F + e$
fin
devolver $T = (U, F)$

Algoritmo de Borůvka (Borůvka 1926):

$F \leftarrow \varnothing$
repetir
  Calcular componentes conexas de (V, F).
  Calcular para cada componente U la arista $e_U \in \delta(U)$ de menor peso[a]
  Agregar todas las aristas e_U a F.
hasta que $cc(V, F) = 1$
devolver $T = (V, F)$

[a] rompiendo empates de manera consistente

Complejidad temporal de un Algoritmo

Modelo de computación

Dado un problema: ¿Qué algoritmo es más rápido? ¿Cuál realiza menor cantidad de operaciones?
Esta pregunta depende del modelo computacional usado. Nosotros usaremos el modelo RAM que básicamente asume lo siguiente (muy simplificado)

Modelo RAM

1. Conjunto de registros (numeros naturales de exactamente w bits c/u) indexados por naturales.
2. Un procesador capaz de hacer operaciones básicas:
    1. Leer/escribir/copiar el valor de un registro (a un valor fijo o al de otro registro).
    2. Tomar dos registros a y b y escribir en un tercero $a+b, a-b, a \cdot b, a/b$.
    3. Comparar dos registros $(>, <, =)$

Modelo de computación (continuación)

Combinando estas operaciones básicas podemos crear operaciones intermedias como

- estructuras de control (if-else-then; while; for; etc.)
- operaciones sobre variables y arreglos (crear variables, acceder a vectores, etc.)
- operaciones en estructuras de datos simples (listas enlazadas, colas, pilas, etc).

Suponemos que estas operaciones intermedias toman un número acotado por una constante universal de operaciones básicas. Estas toman $O(1)$ operaciones básicas.

Formas de medir el tamaño de la entrada

En optimización combinatorial, todo algoritmo toma como entrada una secuencia de números.

- La cantidad de números se denotará por N.
- El número de bits total se denotará por B.
- Además, si la entrada es un grafo G, usaremos $n = |V(G)|$, $m = |E(G)|$.

Sea ALG un algoritmo. Escribimos:

$\textsc{Tiempo}(\mathrm{ALG}, \mathrm{ENTRADA})$:

Número de operaciones básicas que realiza ALG en $\mathrm{ENTRADA}$, antes de terminar.

La función de complejidad del algoritmo ALG es la función

$\text{máx}\{\textsc{Tiempo}(\mathrm{ALG}, \mathrm{ENTRADA})\colon \text{tamaño de } \mathrm{ENTRADA} = x\}$.

Por ejemplo, la complejidad de un algoritmo podría ser $50B^2$, o $20N^5 + 2^N$; y la complejidad de un algoritmo en grafos podría ser $6mn^2$.

Notación asintótica

1. No nos interesa hacer diferencia entre un algoritmo de complejidad $5N^2$ y uno que tome tiempo $5N^2 + N$. Nos interesa en realidad determinar el comportamiento a medida que N crece (comportamiento asintótico).
2. Más aún, lo que importa acá es que el comportamiento es cuadrático (asintóticamente, no importa si es $500N^2$ o si es $N^2/100$).
3. Introduciremos notación para poder decir que dos funciones son asintóticamente parecidas, o que una domina a otra. Por ejemplo, $5N^2 + N \in \Theta(N^2)$.