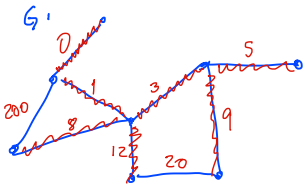
Clase 04 MA3705. 11 de septiembre 2020.

Tabla

- Problema del árbol/bosque generador de peso mínimo (MST).
- Algoritmos de Prim/Jarnik/Boruvka
- Complejidad de algoritmos
- BFS y DFS revisitado

Problema del árbol/bosque generador de peso mínimo (MST)

Bosque generador de costo mínimo.



árbol de peso mínimo

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

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 se llama 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.

óptimo

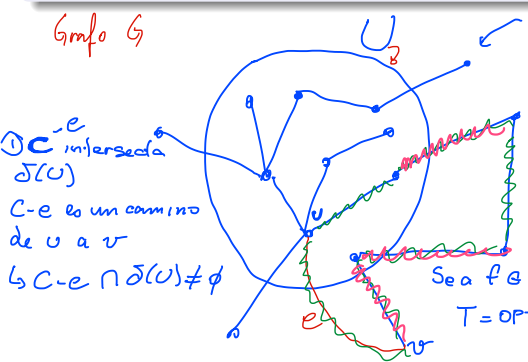
árbol generador de costo mínimo

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

Si e es una 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

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

OPT'



$e \in \delta(U)$, $e \in \operatorname{argmin}\{c(f): f \in \delta(U)\}$

Dem: Caso 1. $e \in \mathrm{OPT} \Rightarrow f = e \in \delta(U) \cap \mathrm{OPT}$

$\mathrm{OPT} - e + e = \mathrm{OPT}$

Caso 2) $e \notin \mathrm{OPT}$ $e = uv$, $u \in U$, $v \notin U$

$C(\mathrm{OPT}, e)$ ← único ciclo en $\mathrm{OPT} + e$

Sabemos $\mathrm{OPT} + e - f$ con $f \in C(\mathrm{OPT}, e) - e$, es árbol

① $C - e$ interseca $\delta(U)$

$C - e$ es un camino de u a v

↳ $C - e \cap \delta(U) \neq \emptyset$

Sea $f \in (C - e \cap \delta(U))$. ↑ $\mathrm{OPT} + e - f$ es árbol.

$T = \mathrm{OPT} + e - f$. $c(T) = c(\mathrm{OPT}) + c(e) - c(f)$

$\leq c(\mathrm{OPT})$. $\therefore$ T es un MST.

② $e, f \in \delta(U)$

$\therefore c(e) \leq c(f)$

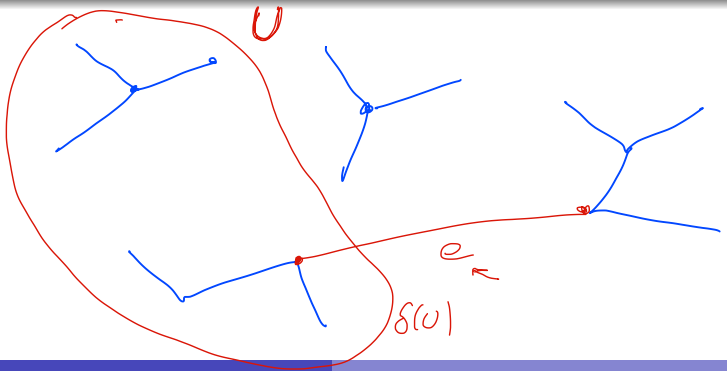
Corolario

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.



F óptimo parcial

F+e óptimo parcial

$F \subseteq OPT$

Si $e \notin OPT \Rightarrow \exists f \in \delta(U) \cap OPT$ tq $OPT + e - f$ es MST, $f \notin F$

F+e

Algoritmos basados en esta idea

Algoritmo de Prim (Prim 1957 - Jarník 1930):

Entrada: $G = (V, E)$ conexo, $r \in V$

Elegir $r \in V$;

$U \leftarrow \{r\}$

$F \leftarrow \varnothing$

mientras $\delta(U) \neq \varnothing$ **hacer**

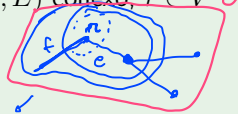
  Sea $e = uv \in \delta(U),\ u \in U,\ v \notin U$ tal que e es la arista de menor peso en $\delta(U)$

  $U \leftarrow U + v$

  $F \leftarrow F + e$

fin

devolver $T = (U, F)$



Algoritmo de Borůvka (Borůvka 1926):

Entrada: $G = (V, E)$ conexo, ~~$r \in V$~~

$F \leftarrow \varnothing$

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

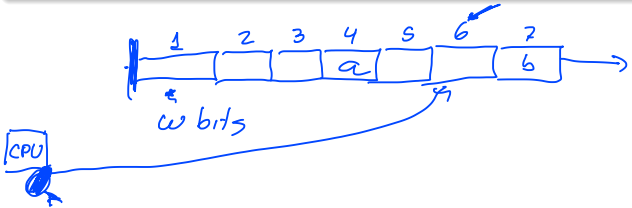
random access memory

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 $(>, <, =)$

división entera



9/4 = 2.

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.)
- lectura/escritura 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. ← Número de datos
- El número de bits total se denotará por B. ← # de bits.
- Además, si la entrada es un grafo G, usaremos $n = |V(G)|$, $m = |E(G)|$.

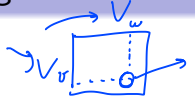
Normalmente estos valores están relacionados entre si.

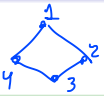
Codificación de un número:

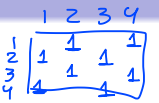
Un número $K \in \mathbb{N}$, en binario, se codifica usando $O(\log K)$ bits. Luego si un algoritmo recibe N números, y el número más grande tiene valor L, entonces $B = O(N \log L)$.

↳ cuando L es fijo $O(N \log L) = O(N)$

Formas de codificar un grafo







Costos de aristas

Matriz de adyacencia

Es una matriz $M \in \{0,1\}^{V \times V}$ con $M_{vw} = 1$ si $vw \in E$, $M_{vw} = 0$, si $vw \notin E$

Ventajas: Revisar si $e \in E$ toma $O(1)$

En este caso, $N = \Theta(n^2)$. $B = \Theta(n^2)$

$n = |V(G)|$

$m = |E(G)|$

Lista de incidencia

Es un arreglo de listas, una para cada vértice.

La lista de v contiene a un puntero (nombre) a cada vértice en $N(v)$.

Ventajas: Más compacta que matriz adyacencia

En este caso, $N = \Theta(n+m)$, $B = \Theta((n+m) \cdot \log n)$

$n+m$

n = largo maximo

Complejidad de un algoritmo

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 (peor-caso)

$$T(x) = \max\{\textsc{Tiempo}(\mathrm{ALG}, \mathrm{ENTRADA}) : \text{tamaño de } \mathrm{ENTRADA} = x\}.$$

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 $6mn^2$.

$O(mn^2)$ $T(N)$ $T(N) = \Theta(2^N)$ $= O(2^N)$

Notación asintótica y eficiencia

Nos interesa el crecimiento de la complejidad a medida que el tamaño de la entrada aumenta, así que todo lo hacemos en notación asintótica.

Desde un punto de vista teórico, un algoritmo se considera eficiente cuando su complejidad está acotada por un polinomio.

¿Por qué polinomios?

$T(N) = O(N^{10})$

$T(2N) \approx (2N)^{10} \sim \underbrace{2^{10}}_{\text{cte}} N^{10} = O(1) \cdot T(N)$

cte × complejidad en N

Esto no pasa para 2^N

$T(N) = 2^N$

$T(2N) = 2^{2N} = (2^N)^2$

Algoritmos polinomiales y fuertemente polinomiales

- Algoritmo polinomial (o débilmente polinomial) es uno con complejidad $O(B^k)$ para algún k fijo, es decir es polinomial en el número de bits de la entrada.
- Algoritmo fuertemente polinomial es uno con complejidad $O(N^k)$ para algún k fijo, es decir es polinomial en el número de datos de la entrada.

Obs:

Si no nos importa el exponente podemos escribir $B^{O(1)}$ o $N^{O(1)}$.