Representación de grafos y digrafos

Julián Braier

FCEN - UBA

2c 2022

Contexto

- Vieron las definiciones básicas de grafos.

Contexto

- Vieron las definiciones básicas de grafos.
- Demostraron algunas propiedades sobre grafos.

Contexto

- Vieron las definiciones básicas de grafos.
- Demostraron algunas propiedades sobre grafos.
- Queremos aprender a diseñar e implementar algoritmos sobre grafos.

Contexto

- Vieron las definiciones básicas de grafos.
- Demostraron algunas propiedades sobre grafos.
- Queremos aprender a diseñar e implementar algoritmos sobre grafos.
- Algún día hay que aprender a representar los grafos en la computadora.

Contexto

- Vieron las definiciones básicas de grafos.
- Demostraron algunas propiedades sobre grafos.
- Queremos aprender a diseñar e implementar algoritmos sobre grafos.
- Algún día hay que aprender a representar los grafos en la computadora. Hoy!

Menú del día

- Repasamos definiciones.

Menú del día

- Repasamos definiciones.
- Analizamos distintas maneras de representar grafos y digrafos según algunas operaciones que deseamos implementar.

Menú del día

- Repasamos definiciones.
- Analizamos distintas maneras de representar grafos y digrafos según algunas operaciones que deseamos implementar.
- Un poquito de implementación...

Menú del día

- Repasamos definiciones.
- Analizamos distintas maneras de representar grafos y digrafos según algunas operaciones que deseamos implementar.
- Un poquito de implementación...
- Un ejercicio de parcial viejo.

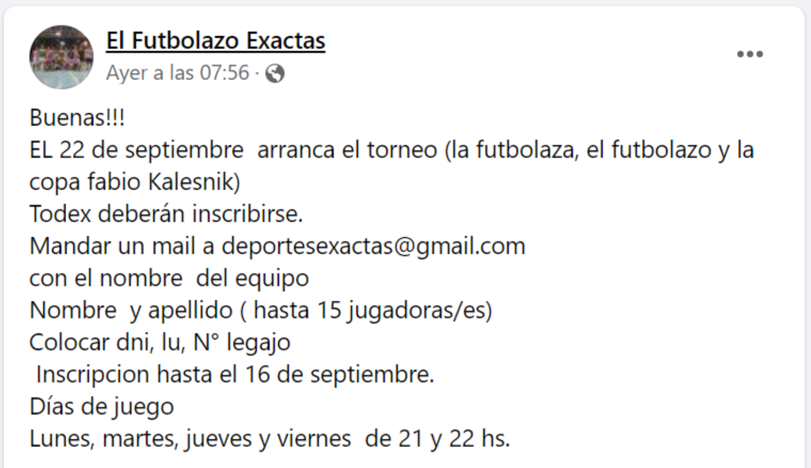
Menú del día

- Repasamos definiciones.
- Analizamos distintas maneras de representar grafos y digrafos según algunas operaciones que deseamos implementar.
- Un poquito de implementación...
- Un ejercicio de parcial viejo.
- Una representación para árboles.

Menú del día

- Repasamos definiciones.
- Analizamos distintas maneras de representar grafos y digrafos según algunas operaciones que deseamos implementar.
- Un poquito de implementación...
- Un ejercicio de parcial viejo.
- Una representación para árboles.
- Grafos implícitos.

Primer Espacio Publicitario: Deportes



El Futbolazo Exactas
Ayer a las 07:56 ·

Buenas!!!
EL 22 de septiembre arranca el torneo (la futbolaza, el futbolazo y la copa fabio Kalesnik)
Todex deberán inscribirse.
Mandar un mail a deportesexactas@gmail.com
con el nombre del equipo
Nombre y apellido (hasta 15 jugadoras/es)
Colocar dni, lu, N° legajo
Inscripcion hasta el 16 de septiembre.
Días de juego
Lunes, martes, jueves y viernes de 21 y 22 hs.

https://www.facebook.com/elfutbolazo
https://www.instagram.com/futbolazayfutbolazo.fcen/?hl=es-la

Primer Espacio Publicitario: Deportes

https://www.instagram.com/deportes_exactas/?hl=es-la

Fin del Espacio Publicitario

Definiciones

Definición (Grafo)

Un **grafo** $G = (V, E)$ es un par de conjuntos, donde V es un conjunto de puntos o nodos o vértices y E es un subconjunto del conjunto de pares **no ordenados** de elementos distintos de V. Los elementos de E se llaman aristas o ejes.

Definiciones

Definición (Grafo)

Un **grafo** $G = (V, E)$ es un par de conjuntos, donde V es un conjunto de puntos o nodos o vértices y E es un subconjunto del conjunto de pares **no ordenados** de elementos distintos de V. Los elementos de E se llaman aristas o ejes.

Definición (Digrafo)

Un **digrafo** $G = (V, E)$ es un par de conjuntos, donde V es un conjunto de vértices y E es un subconjunto del conjunto de pares **ordenados** de elementos distintos de V.

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.
- $N(v)$:

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.
- $N(v)$: vecindario del vértice v.
- $N[v]$:

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.
- $N(v)$: vecindario del vértice v.
- $N[v]$: vecindario cerrado de v, $N[v] = N(v) \cup \{v\}$.
- $d(v)$:

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.
- $N(v)$: vecindario del vértice v.
- $N[v]$: vecindario cerrado de v, $N[v] = N(v) \cup \{v\}$.
- $d(v)$: cantidad de vecinos (grado) de v, $d(v) = |N(v)|$.
- $N^{in}(v)$ y $N^{out}(v)$:

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.
- $N(v)$: vecindario del vértice v.
- $N[v]$: vecindario cerrado de v, $N[v] = N(v) \cup \{v\}$.
- $d(v)$: cantidad de vecinos (grado) de v, $d(v) = |N(v)|$.
- $N^{in}(v)$ y $N^{out}(v)$: vecindarios de entrada y de salida del vértice v.
- $d^{in}(v)$ y $d^{out}(v)$:

Notaciones

- Habitualmente n y m denotan la cantidad de vértices y aristas del grafo.
- $N(v)$: vecindario del vértice v.
- $N[v]$: vecindario cerrado de v, $N[v] = N(v) \cup \{v\}$.
- $d(v)$: cantidad de vecinos (grado) de v, $d(v) = |N(v)|$.
- $N^{in}(v)$ y $N^{out}(v)$: vecindarios de entrada y de salida del vértice v.
- $d^{in}(v)$ y $d^{out}(v)$: grados de entrada y de salida en un digrafo.

Representaciones (con poco nivel de detalle)

- Vamos a asumir que representamos los vértices del grafo con números de 0 a $n - 1$. Conveniente.

Representaciones (con poco nivel de detalle)

- Vamos a asumir que representamos los vértices del grafo con números de 0 a $n-1$. Conveniente.
- Podemos almacenar de dos maneras un grafo en memoria:

Representaciones (con poco nivel de detalle)

- Vamos a asumir que representamos los vértices del grafo con números de 0 a $n - 1$. Conveniente.
- Podemos almacenar de dos maneras un grafo en memoria:
  - Conjunto de aristas: almacenar el conjunto de aristas E en alguna estructura de datos que implemente conjunto de pares.

Representaciones (con poco nivel de detalle)

- Vamos a asumir que representamos los vértices del grafo con números de 0 a $n-1$. Conveniente.
- Podemos almacenar de dos maneras un grafo en memoria:
  - Conjunto de aristas: almacenar el conjunto de aristas E en alguna estructura de datos que implemente conjunto de pares.
  - Diccionario: tener un diccionario que, dado un vértice $v \in V$, nos dé $N(v)$, su vecindario. Esta opción se llama *conjunto de adyacencias*.

Representaciones (con poco nivel de detalle)

- Vamos a asumir que representamos los vértices del grafo con números de 0 a $n-1$. Conveniente.
- Podemos almacenar de dos maneras un grafo en memoria:
  - Conjunto de aristas: almacenar el conjunto de aristas E en alguna estructura de datos que implemente conjunto de pares.
  - Diccionario: tener un diccionario que, dado un vértice $v \in V$, nos dé $N(v)$, su vecindario. Esta opción se llama *conjunto de adyacencias*. Para representar a $N(v)$ podemos usar varias estructuras de datos (fuertemente basado en AED II).

Refrescando opciones de estructuras de datos

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Stack	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Queue	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Singly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Doubly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Skip List	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n \log(n))$
Hash Table	N/A	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Binary Search Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Cartesian Tree	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
B-Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
Red-Black Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
Splay Tree	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	N/A	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
AVL Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$	$O(n)$
KD Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

https://www.bigocheatsheet.com/

¿Cuál representación es mejor?

¿Cuál representación es mejor?

Depende... qué hay que tener en cuenta:

¿Cuál representación es mejor?

Depende... qué hay que tener en cuenta:

- Características del grafo (ejemplos: ralo vs denso, es árbol?, es bipartito?).

¿Cuál representación es mejor?

Depende... qué hay que tener en cuenta:

- Características del grafo (ejemplos: ralo vs denso, es árbol?, es bipartito?).
- Operaciones que necesitamos para implementar el algoritmo. Cuántas veces usamos cada operación. Elegir opción menos costosa.

¿Cuál representación es mejor?

Depende... qué hay que tener en cuenta:

- Características del grafo (ejemplos: ralo vs denso, es árbol?, es bipartito?).
- Operaciones que necesitamos para implementar el algoritmo. Cuántas veces usamos cada operación. Elegir opción menos costosa.
- Costo de construcción vs costo del algoritmo.

Operaciones

a) Inicializar la estructura a partir de un conjunto de aristas de G.

b) Determinar si dos vértices v y w son adyacentes.

c) Recorrer y/o procesar el vecindario $N(v)$ de un vértice v dado.

d) Insertar un vértice v con su conjunto de vecinos $N(v)$.

e) Insertar una arista vw.

f) Remover un vértice v con todas sus adyacencias.

g) Remover una arista vw.

h) Mantener un orden de $N(v)$ de acuerdo a algún invariante que permita recorrer cada vecindario en un orden dado.

Implementaciones de conjunto de adyacencias

a) N se representa con una secuencia (vector o lista) que en cada posición v tiene el conjunto $N(v)$ implementado sobre una secuencia (lista o vector). Cada vértice es una estructura que tiene un índice para acceder en $O(1)$ a $N(v)$. Esta representación se conoce comúnmente como *lista de adyacencias*.

Implementaciones de conjunto de adyacencias

a) N se representa con una secuencia (vector o lista) que en cada posición v tiene el conjunto $N(v)$ implementado sobre una secuencia (lista o vector). Cada vértice es una estructura que tiene un índice para acceder en $O(1)$ a $N(v)$. Esta representación se conoce comúnmente como *lista de adyacencias*.

b) ídem anterior, pero cada $w \in N(v)$ se almacena junto con un índice a la posición que ocupa v en $N(w)$. Esta representación también se conoce como *lista de adyacencias*, pero tiene información para implementar operaciones dinámicas.

Implementaciones de conjunto de adyacencias

a) N se representa con una secuencia (vector o lista) que en cada posición v tiene el conjunto $N(v)$ implementado sobre una secuencia (lista o vector). Cada vértice es una estructura que tiene un índice para acceder en $O(1)$ a $N(v)$. Esta representación se conoce comúnmente como *lista de adyacencias*.

b) ídem anterior, pero cada $w \in N(v)$ se almacena junto con un índice a la posición que ocupa v en $N(w)$. Esta representación también se conoce como *lista de adyacencias*, pero tiene información para implementar operaciones dinámicas.

c) $N(v)$ se representa con un vector que en cada posición i tiene un vector booleano A_i con $|V(G)|$ posiciones tal que $A_i[j]$ es verdadero si y solo si i es adyacente a j.

Implementaciones de conjunto de adyacencias

a) N se representa con una secuencia (vector o lista) que en cada posición v tiene el conjunto $N(v)$ implementado sobre una secuencia (lista o vector). Cada vértice es una estructura que tiene un índice para acceder en $O(1)$ a $N(v)$. Esta representación se conoce comúnmente como *lista de adyacencias*.

b) ídem anterior, pero cada $w \in N(v)$ se almacena junto con un índice a la posición que ocupa v en $N(w)$. Esta representación también se conoce como *lista de adyacencias*, pero tiene información para implementar operaciones dinámicas.

c) $N(v)$ se representa con un vector que en cada posición i tiene un vector booleano A_i con $|V(G)|$ posiciones tal que $A_i[j]$ es verdadero si y solo si i es adyacente a j. Esta representación se conoce comúnmente como *matriz de adyacencias*.

Implementaciones de conjunto de adyacencias

a) N se representa con una secuencia (vector o lista) que en cada posición v tiene el conjunto $N(v)$ implementado sobre una secuencia (lista o vector). Cada vértice es una estructura que tiene un índice para acceder en $O(1)$ a $N(v)$. Esta representación se conoce comúnmente como *lista de adyacencias*.

b) ídem anterior, pero cada $w \in N(v)$ se almacena junto con un índice a la posición que ocupa v en $N(w)$. Esta representación también se conoce como *lista de adyacencias*, pero tiene información para implementar operaciones dinámicas.

c) $N(v)$ se representa con un vector que en cada posición i tiene un vector booleano A_i con $|V(G)|$ posiciones tal que $A_i[j]$ es verdadero si y solo si i es adyacente a j. Esta representación se conoce comúnmente como *matriz de adyacencias*.

d) $N(v)$ se representa con un vector que en cada posición tiene el conjunto $N(v)$ implementado con una tabla de hash. Esta representación es un mix entre las representaciones clásicas de matriz de adyacencias y lista de adyacencias.

Implementamos la opción 2 (lista de adyacencias con índices)

- Ahora tenemos un vector de vectores de pares (vecino,índice). Llamado *ady*.

Implementamos la opción 2 (lista de adyacencias con índices)

- Ahora tenemos un vector de vectores de pares (vecino,índice). Llamado *ady*.
- Escribo formalmente el invariante:

$$ady[u][i] = (v, j) \iff ady[v][j] = (u, i)$$

Implementamos la opción 2 (lista de adyacencias con índices)

- Ahora tenemos un vector de vectores de pares (vecino,índice). Llamado *ady*.
- Escribo formalmente el invariante:

$$ady[u][i] = (v, j) \iff ady[v][j] = (u, i)$$

- Ejemplo en pizarrón.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n + m)$.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.
  c) $O(n^2)$.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.
  c) $O(n^2)$.
  d) $O(n+m)$ (esperado).

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n + m)$.
  b) $O(n + m)$.
  c) $O(n^2)$.
  d) $O(n + m)$ (esperado).
- ¿Cuánto cuesta borrar un vértice v?

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.
  c) $O(n^2)$.
  d) $O(n+m)$ (esperado).
- ¿Cuánto cuesta borrar un vértice v?
  a) $O(\sum_{u \in N[v]} d(u)) \in O(m)$.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.
  c) $O(n^2)$.
  d) $O(n+m)$ (esperado).
- ¿Cuánto cuesta borrar un vértice v?
  a) $O(\sum_{u \in N[v]} d(u)) \in O(m)$.
  b) $O(d(v))$.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.
  c) $O(n^2)$.
  d) $O(n+m)$ (esperado).
- ¿Cuánto cuesta borrar un vértice v?
  a) $O(\sum_{u \in N[v]} d(u)) \in O(m)$.
  b) $O(d(v))$.
  c) $O(n)$.

Análisis de complejidad

- ¿Cuánto cuesta inicializar cada estructura a partir de un conjunto de aristas?
  a) $O(n+m)$.
  b) $O(n+m)$.
  c) $O(n^2)$.
  d) $O(n+m)$ (esperado).
- ¿Cuánto cuesta borrar un vértice v?
  a) $O(\sum_{u \in N[v]} d(u)) \in O(m)$.
  b) $O(d(v))$.
  c) $O(n)$.
  d) $O(d(v))$ (esperado).

Ejercicio de Parcial[1]

Queremos determinar aquellas aristas $v \to w$ de un digrafo D tales que $w \to v$ no es arista de D. Para ello, podemos usar las siguientes ideas sobre las estructuras de datos vistas en clase.

a) Describir un algoritmo lineal que, dado un multigrafo G representado con un conjunto de aristas, determine las aristas (v, w) que no están repetidas en G.

b) Describir un algoritmo lineal que, dado un digrafo D representado con un conjunto de aristas, determine las aristas $v \to w$ tales que $w \to v$ no es arista de D.

[1] Tomado en el primer recuperatorio del 1C 2022

Solución

¿Se acuerdan de *radix sort*?

Algorithm 3 RADIX-SORT(A, d)

```
for i ← [1..d] // del menos significativo al más significativo do
    Ordenar el arreglo A según el dígito i, en forma estable
end for
```

Solución

a) (aristas no repetidas en multigrafo)

- Asumo que tengo G representado con un vector de pares de enteros.

Solución

a) (aristas no repetidas en multigrafo)

- Asumo que tengo G representado con un vector de pares de enteros.
- Primero "normalizo" las aristas. Pongo siempre al vértice de menor número adelante. Es decir, cambio (v, w) por $(min\{v, w\}, max\{v, w\})$.

Solución

a) (aristas no repetidas en multigrafo)

- Asumo que tengo G representado con un vector de pares de enteros.
- Primero "normalizo" las aristas. Pongo siempre al vértice de menor número adelante. Es decir, cambio (v, w) por $(min\{v, w\}, max\{v, w\})$.
- Ordeno las aristas lexicográficamente (ordeno por primer vértice, desempato por el segundo). Con *radix sort* + *counting sort* sale en $O(2 * (n + m)) = O(n + m)$.

Solución

a) (aristas no repetidas en multigrafo)

- Asumo que tengo G representado con un vector de pares de enteros.
- Primero "normalizo" las aristas. Pongo siempre al vértice de menor número adelante. Es decir, cambio (v, w) por $(min\{v, w\}, max\{v, w\})$.
- Ordeno las aristas lexicográficamente (ordeno por primer vértice, desempato por el segundo). Con *radix sort* + *counting sort* sale en $O(2 * (n + m)) = O(n + m)$.
- En una sola pasada por el conjunto de aristas ordenado obtengo la respuesta en tiempo lineal. Selecciono a una arista sii:

Solución

a) (aristas no repetidas en multigrafo)

- Asumo que tengo G representado con un vector de pares de enteros.
- Primero "normalizo" las aristas. Pongo siempre al vértice de menor número adelante. Es decir, cambio (v, w) por $(min\{v, w\}, max\{v, w\})$.
- Ordeno las aristas lexicográficamente (ordeno por primer vértice, desempato por el segundo). Con *radix sort* + *counting sort* sale en $O(2 * (n + m)) = O(n + m)$.
- En una sola pasada por el conjunto de aristas ordenado obtengo la respuesta en tiempo lineal. Selecciono a una arista sii:
  - Es distinta a la arista anterior (o es la primera) y
  - Es distinta a la arista siguiente (o es la última).

Solución

b) (aristas tales que no existe la arista reversa en un digrafo)

Solución

b) (aristas tales que no existe la arista reversa en un digrafo)

- Obtengo D^r el digrafo reverso de D copiando al conjunto de aristas y revirtiendo el sentido de cada una. $O(m)$.

Solución

b) (aristas tales que no existe la arista reversa en un digrafo)

- Obtengo D^r el digrafo reverso de D copiando al conjunto de aristas y revirtiendo el sentido de cada una. $O(m)$.
- Ordeno las aristas de D y de D^r en $O(n+m)$ como en el ítem *a*).

Solución

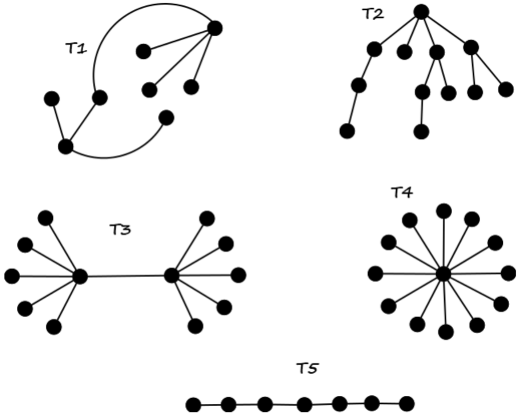
b) (aristas tales que no existe la arista reversa en un digrafo)

- Obtengo D^r el digrafo reverso de D copiando al conjunto de aristas y revirtiendo el sentido de cada una. $O(m)$.
- Ordeno las aristas de D y de D^r en $O(n+m)$ como en el ítem a).
- Con las aristas así ordenadas puedo ver cuáles de D no aparecen en D^r en tiempo lineal avanzando dos punteros. Un algoritmo tipo *merge*.

Árboles

Definición 1. Un *árbol* es un grafo conexo sin circuitos simples.

Ejemplo 1.



2

[2] Robado de apuntes de Paula Zabala de cuatrimestres anteriores.

Árboles

Teorema 2. *Dado un grafo G son equivalentes:*

1. *G es un árbol.*
2. *G es un grafo sin circuitos simples y $m = n - 1$.*
3. *G es conexo y $m = n - 1$.*

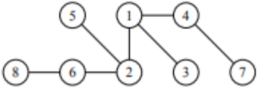
[3]

[3] Esto también.

Representando árboles

Aprovechando que un árbol tiene $O(n)$ aristas podemos pensar una representación más compacta.

Fig. 10.1 Tree that consists of 8 nodes and 7 edges



[4]

[4] Robado de un libro de programación competitiva. *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests. Antti Laaksonen.*

Segundo Espacio Publicitario: Programación Competitiva

TAP 2022 Inscripción Abierta!

August 31, 2022 in Uncategorized by fedepousa

____

2do TORNEO ARGENTINO DE PROGRAMACIÓN
8 de octubre 2022
Inscripción abierta en https://icpc.global/regionals/finder/TAP-2022

____

Sedes: Bahía Blanca (UNS), Buenos Aires (UBA), Chilecito (UNdeC),
Córdoba (UNC), Jujuy(UNJu), La Plata (UNLP), La Rioja (UNLaR), Orán (UNS), Rosario (UNR), Santa Fe(UTN) y Tucumán (UNT)

_____________ [5]

---

[5] `http://torneoprogramacion.com.ar/2022/08/31/tap-2022-inscripcion-abierta/`

Segundo Espacio Publicitario: Programación Competitiva

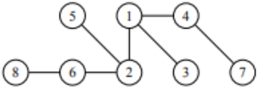
Tomen links:

- `https://icpc.global/` (página oficial de la ICPC)
- `https://codeforces.com/` (página de competencias online)
- `https://codingcompetitions.withgoogle.com/codejam/` (competencia anual organizada por *Google*)
- `https://codingcompetitions.withgoogle.com/kickstart` (otra organizada por *Google*, apunta más a principiantes)
- `https://cses.fi/` (Code Submission Evaluation System, por Antti el del libro)

Fin del espacio publicitario

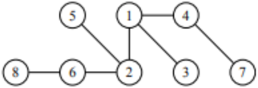
Representando árboles

Fig. 10.1 Tree that consists of 8 nodes and 7 edges



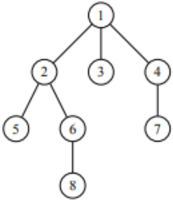
Representando árboles

Fig. 10.1 Tree that consists of 8 nodes and 7 edges



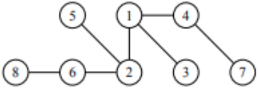
Enraizamos el árbol.

Fig. 10.2 Rooted tree where node 1 is the root node



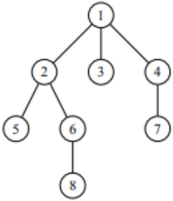
Representando árboles

Fig. 10.1 Tree that consists of 8 nodes and 7 edges



Enraizamos el árbol.

Fig. 10.2 Rooted tree where node 1 is the root node



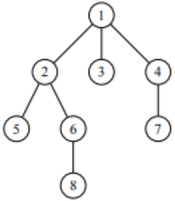
Podemos tener una función *padre* tal que $padre[v]$ es el (único) padre de v para todo v que no es la raíz. Para la raíz r podríamos poner $padre[r] = r$.

v	1	2	3	4	5	6	7	8
$padre[v]$	1	1	1	1	2	2	4	6

Utilizando esta representación

Asumamos que, además de la función *padre*, tenemos una función *profundidad* que nos dice para cada vértice v cuál es su profundidad en el árbol.

Fig. 10.2 Rooted tree where node 1 is the root node



v	1	2	3	4	5	6	7	8
$profundidad[v]$	0	1	1	1	2	2	2	3

Utilizando esta representación

Definición (Ancestro)

Un vértice v es ancestro de sí mismo. Si w es ancestro de v entonces $padre[w]$ también es ancestro de v.

Utilizando esta representación

Definición (Ancestro)

Un vértice v es ancestro de sí mismo. Si w es ancestro de v entonces $padre[w]$ también es ancestro de v.

- **Problema:** dados dos vértices v, w, ¿es w ancestro de v?

Utilizando esta representación

Definición (Ancestro)

Un vértice v es ancestro de sí mismo. Si w es ancestro de v entonces $padre[w]$ también es ancestro de v.

- **Problema:** dados dos vértices v, w, ¿es w ancestro de v?

Definición (Ancestro común más bajo)

Dados dos vértices v, w. El ancestro común más bajo (LCA) de v y w es, de los ancestros comunes de v y w (siempre tienen alguno en común), el más profundo.

Utilizando esta representación

Definición (Ancestro)

Un vértice v es ancestro de sí mismo. Si w es ancestro de v entonces $padre[w]$ también es ancestro de v.

- **Problema:** dados dos vértices v, w, ¿es w ancestro de v?

Definición (Ancestro común más bajo)

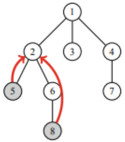
Dados dos vértices v, w. El ancestro común más bajo (LCA) de v y w es, de los ancestros comunes de v y w (siempre tienen alguno en común), el más profundo.

- **Problema:** dados dos vértices v, w obtener $lca(v, w)$.

Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener $lca(v, w)$.

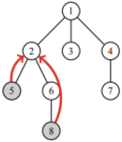
Fig. 10.19 Lowest common ancestor of nodes 5 and 8 is node 2



Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener $lca(v, w)$.

Fig. 10.19 Lowest common ancestor of nodes 5 and 8 is node 2



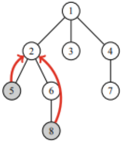
Solución:

1. Empezamos con dos punteros, uno en v y otro en w.

Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener $lca(v, w)$.

Fig. 10.19 Lowest common ancestor of nodes 5 and 8 is node 2



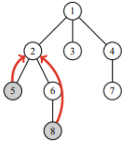
Solución:

1. Empezamos con dos punteros, uno en v y otro en w.
2. Ascendemos el más profundo de los punteros hasta que estén en el mismo nivel.

Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener $lca(v, w)$.

Fig. 10.19 Lowest common ancestor of nodes 5 and 8 is node 2



Solución:

1. Empezamos con dos punteros, uno en v y otro en w.
2. Ascendemos el más profundo de los punteros hasta que estén en el mismo nivel.
3. Ascendemos de a un pasito ambos punteros a la vez hasta que coincidan. En donde coincidan tenemos el LCA.

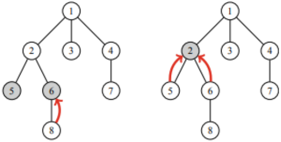


Fig. 10.20 Two steps to find the lowest common ancestor of nodes 5 and 8

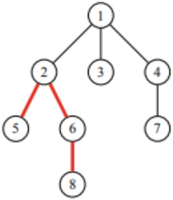
Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener la distancia entre v y w.

Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener la distancia entre v y w.

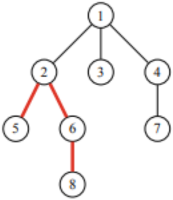
Fig. 10.23 Calculating the distance between nodes 5 and 8



Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener la distancia entre v y w.

Fig. 10.23 Calculating the distance between nodes 5 and 8

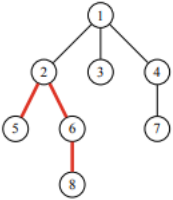


1. El (único) camino de v a w es la concatenación de los caminos de v a u y de u a w, donde u es el $lca(v, w)$.

Utilizando esta representación

- **Problema:** dados dos vértices v, w obtener la distancia entre v y w.

Fig. 10.23 Calculating the distance between nodes 5 and 8



1. El (único) camino de v a w es la concatenación de los caminos de v a u y de u a w, donde u es el $lca(v, w)$.
2. La longitud de este camino es $prof[v] - prof[u] + prof[w] - prof[u] = prof[v] + prof[w] - 2 * prof[u]$.

Bonus: una representación con redundancia

Definición (k-ésimo ancestro)

El k-ésimo ancestro de v es el vértice al que llego luego de subir k veces al padre por el árbol.
Más formalmente: sea $ancestro(v, k)$ el k-ésimo ancestro de v.

- $ancestro(v, 0) = v$.
- $ancestro(v, k) = padre[ancestro(v, k - 1)]$ si $k > 0$.

---

[6] si quieren buscar la respuesta pueden ver en el libro de Antti, o en `https://cp-algorithms.com/graph/lca_binary_lifting.html` o preguntarme

Bonus: una representación con redundancia

Definición (k-ésimo ancestro)

El k-ésimo ancestro de v es el vértice al que llego luego de subir k veces al padre por el árbol.
Más formalmente: sea $ancestro(v, k)$ el k-ésimo ancestro de v.

- $ancestro(v, 0) = v$.
- $ancestro(v, k) = padre[ancestro(v, k - 1)]$ si $k > 0$.

- Podemos precomputar $ancestro(v, k)$ para todo k que sea potencia de $2 \leq n$.

---

[6] si quieren buscar la respuesta pueden ver en el libro de Antti, o en `https://cp-algorithms.com/graph/lca_binary_lifting.html` o preguntarme

Bonus: una representación con redundancia

Definición (k-ésimo ancestro)

El k-ésimo ancestro de v es el vértice al que llego luego de subir k veces al padre por el árbol.
Más formalmente: sea $ancestro(v, k)$ el k-ésimo ancestro de v.

- $ancestro(v, 0) = v$.
- $ancestro(v, k) = padre[ancestro(v, k - 1)]$ si $k > 0$.

- Podemos precomputar $ancestro(v, k)$ para todo k que sea potencia de $2 \leq n$.
- ¿Cuánto cuesta? $O(n \ lg \ n)$ tiempo y memoria. Pensar cómo (si quieren).

[6] si quieren buscar la respuesta pueden ver en el libro de Antti, o en `https://cp-algorithms.com/graph/lca_binary_lifting.html` o preguntarme

Bonus: una representación con redundancia

Definición (k-ésimo ancestro)

El k-ésimo ancestro de v es el vértice al que llego luego de subir k veces al padre por el árbol.
Más formalmente: sea $ancestro(v, k)$ el k-ésimo ancestro de v.

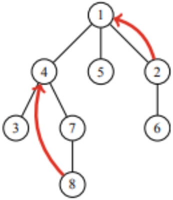
- $ancestro(v, 0) = v$.
- $ancestro(v, k) = padre[ancestro(v, k-1)]$ si $k > 0$.

- Podemos precomputar $ancestro(v, k)$ para todo k que sea potencia de $2 \leq n$.
- ¿Cuánto cuesta? $O(n\ lg\ n)$ tiempo y memoria. Pensar cómo (si quieren).
- ¿Para qué sirve? Con esta estructura los tres problemas que acabamos de ver se resuelven en $O(lg\ n)$. Pensar cómo (si quieren)[6].

[6] si quieren buscar la respuesta pueden ver en el libro de Antti, o en `https://cp-algorithms.com/graph/lca_binary_lifting.html` o preguntarme

Bonus: una representación con redundancia

Fig. 10.12 Finding ancestors of nodes



x	1	2	3	4	5	6	7	8
ancestor(x, 1)	0	1	4	1	1	2	4	7
ancestor(x, 2)	0	0	1	0	0	1	1	4
ancestor(x, 4)	0	0	0	0	0	0	0	0
...								

Grafos Implícitos

- Podemos resolver problemas usando teoría de grafos sin representar explícitamente el grafo en memoria.

Grafos Implícitos

- Podemos resolver problemas usando teoría de grafos sin representar explícitamente el grafo en memoria.
- **Ejemplo:** El trabado.

Grafos Implícitos

- Podemos resolver problemas usando teoría de grafos sin representar explícitamente el grafo en memoria.
- **Ejemplo:** El trabado.
  - V representa al conjunto de configuraciones posibles.

Grafos Implícitos

- Podemos resolver problemas usando teoría de grafos sin representar explícitamente el grafo en memoria.
- **Ejemplo:** El trabado.
  - V representa al conjunto de configuraciones posibles.
  - $uv \in E \iff$ se puede pasar de la configuración u a la configuración v en un movimiento.

Grafos Implícitos

- Podemos resolver problemas usando teoría de grafos sin representar explícitamente el grafo en memoria.
- **Ejemplo:** El trabado.
  - V representa al conjunto de configuraciones posibles.
  - $uv \in E \iff$ se puede pasar de la configuración u a la configuración v en un movimiento.
  - El problema se reduce a encontrar un camino en $G = (V, E)$ desde el vértice que representa a la configuración inicial a una desde la cual puedo sacar el cuadrado rojo.