

Grafos y Backtracking

Técnicas de Diseño de Algoritmos

FCEyN UBA

Marzo, 2025

¿Qué vamos a ver hoy?

- ▶ Presentación de la materia
- ▶ Un breve repaso de grafos
- ▶ Repaso de backtracking
- ▶ Y algunos ejercicios de backtracking

Presentación de la materia

Días y horarios:

- ▶ En principio los lunes serán las clases teóricas y los miércoles las prácticas
- ▶ Ambas son importantes, no se va a dar repaso de la teórica en la práctica

Presentación de la materia

Días y horarios:

- ▶ En principio los lunes serán las clases teóricas y los miércoles las prácticas
- ▶ Ambas son importantes, no se va a dar repaso de la teórica en la práctica

Para aprobar la materia necesitan aprobar:

- ▶ Dos parciales o sus respectivos recuperatorios
- ▶ Un TP final que se realiza en grupos de 5 personas

Grafos

Este cuatrimestre la materia cambió un poco y los grafos van a ser transversales a todos los temas de la materia.



Este cuatrimestre la materia cambió un poco y los grafos van a ser transversales a todos los temas de la materia.

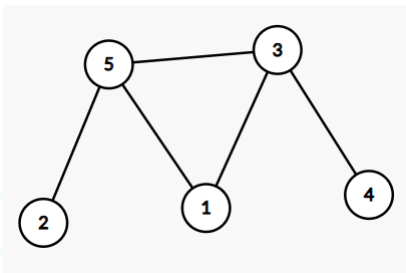
Entonces empezamos la materia definiéndolos y viendo cómo representarlos.

Definiciones

Grafo

Un grafo es un par (V, E) con V un conjunto de nodos (o vértices) y E conjunto de aristas (o ejes) de la forma (u, v) con $u, v \in V$.

Si no se aclara, se asume que los ejes no son dirigidos, es decir que $(u, v) = (v, u)$.



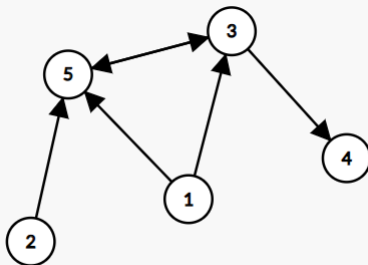
- En general vamos a notar n la cantidad de nodos y m a la cantidad de aristas.

Definiciones

Digrafo

Es un grafo, es decir, un par (V, E) como antes, pero cuyas aristas están orientadas. Es decir que en este caso $(u, v) \neq (v, u)$

Puedo tener dos ejes entre dos nodos, uno en cada sentido.



Definiciones

Otros:

- ▶ **Multigrafo:** grafo que permite tener múltiples aristas entre dos nodos.
- ▶ **Pseudografo:** grafo que permiten tener loops.
- ▶ **Grafo pesado:** grafo $G=(V, E, w)$ con $w(e)$ una "función de pesos" que asigna a cada arista $e=(u,v)$ un peso.

Por el momento nos vamos a concentrar en grafos y digrafos.

Más definiciones...

- ▶ **Recorrido:** una sucesión de vértices y aristas del grafo
- ▶ **Camino:** un recorrido que **no** pasa dos veces por el mismo vértice.
- ▶ **Circuito:** un recorrido que empieza y termina en el mismo nodo
- ▶ **Ciclo o circuito simple:** un circuito que **no** repite vértices. (Nota: para grafos, no consideramos como válido al ciclo de longitud 2)
- ▶ **Longitud:** la longitud de un recorrido se nota $l(P)$ y es la cantidad de *aristas* del mismo.
- ▶ **Distancia entre dos vértices:** longitud del camino más corto entre los vértices (si no existe se dice ∞).
- ▶ Un nodo es **adyacente** a otro si están conectados.
- ▶ Una arista es **incidente** a un nodo si conecta dicho nodo con algún otro.

Modelando con grafos

- ▶ Una ciudad?
- ▶ Una red social?
- ▶ Amistades?
- ▶ Operaciones aritméticas?
- ▶ Redes de información?
- ▶ Tareas?
- ▶ El cerebro humano?

Representando grafos

Por conveniencia, vamos a suponer que todos los nodos del grafo son números de $[1, n]$.

Principalmente vamos a utilizar las siguientes estructuras de representación:

- ▶ Lista de aristas
- ▶ Lista de adyacencia
- ▶ Matriz de adyacencia

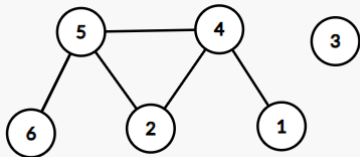
Veamos cada una de estas representaciones y sus diferencias.

Ejemplo: lista de aristas

lista de aristas

El conjunto de aristas como una secuencia (lista).

Tomemos el siguiente grafo como ejemplo:



Lista de aristas:

$\{(6, 5), (5, 2), (2, 4), (5, 4), (4, 1)\}$

Nota 1: normalmente esta va a ser la única representación con la que se van a expresar los inputs de grafos.

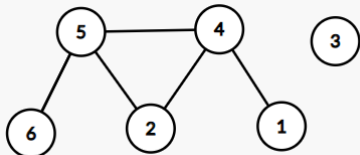
Nota 2: si junto con la lista se pasa el tamaño del grafo, como por convención las etiquetas de los vértices están numeradas 1...n podemos deducir qué vértices no tienen vecinos.

Ejemplo: lista de adyacencia

Lista de adyacencia

El diccionario es un vector y los vecindarios son listas de tamaño $d(v)$ conteniendo a los nodos vecinos.

Tomemos el siguiente grafo como ejemplo:



Lista de adyacencia:

Nodo : lista de vecinos

1 : 4

2 : 4 → 5

3 :

4 : 5 → 1 → 2

5 : 2 → 6 → 4

6 : 5

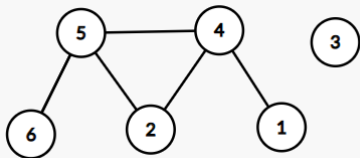
Nota: Se pueden hacer cosas como ordenar los vecindarios, pero es caro mantenerlo si el grafo cambia.

Ejemplo: matriz de adyacencia

Matriz de adyacencia

El diccionario y los vecindarios son vectores de tamaño n .
Resultando así en una matriz de $n \times n$ donde $M_{ij} = 1$ si los vértices i y j son adyacentes y $M_{ij} = 0$ si no.

Tomemos el siguiente grafo como ejemplo:



Matriz de adyacencia:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Operaciones

¿Qué operaciones podríamos llegar a querer hacer sobre un grafo?

- ▶ Inicializar un grafo.
- ▶ Agregar o sacar un nodo del grafo.
- ▶ Agregar o sacar una arista del grafo
- ▶ Obtener el vecindario de un nodo v .
- ▶ Evaluar si dos aristas u y v son adyacentes.

Complejidades

Las complejidades para las representaciones vistas quedarían:

	lista de aristas	matriz de ady.	listas de ady.
construcción	$O(m)$	$O(n^2)$	$O(n + m)$
adyacentes	$O(m)$	$O(1)$	$O(d(v))$
vecinos	$O(m)$	$O(n)$	$O(d(v))$
agregarArista	$O(m)$	$O(1)$	$O(d(u) + d(v))$
removerArista	$O(m)$	$O(1)$	$O(d(u) + d(v))$
agregarVértice	$O(1)$	$O(n^2)$	$O(n)$
removerVértice	$O(m)$	$O(n^2)$	$O(n + m)$

Nota: como el tamaño del grafo está dado por su cantidad de nodos y aristas, una complejidad $O(n+m)$ es lineal respecto al tamaño del grafo.

Eligiendo una representación

Qué representación conviene usar va a depender de:

- ▶ Las características del grafo (por ejemplo si es ralo o denso).
- ▶ Para qué lo vamos a querer usar y qué complejidades queremos para sus operaciones.

Backtracking

Idea: recorrer sistemáticamente todas las posibles configuraciones del espacio de soluciones de manera recursiva.

Tenemos:



Backtracking

Idea: recorrer sistemáticamente todas las posibles configuraciones del espacio de soluciones de manera recursiva.

Tenemos:

- ▶ Soluciones parciales
- ▶ Soluciones candidatas
- ▶ Soluciones válidas

Sudoku

Tomemos como ejemplo un Sudoku de 4x4 en el que la regla es que no se pueden repetir números en filas ni en columnas. Queremos hacer un algoritmo de backtracking que lo resuelva:



Sudoku

Tomemos como ejemplo un Sudoku de 4x4 en el que la regla es que no se pueden repetir números en filas ni en columnas.

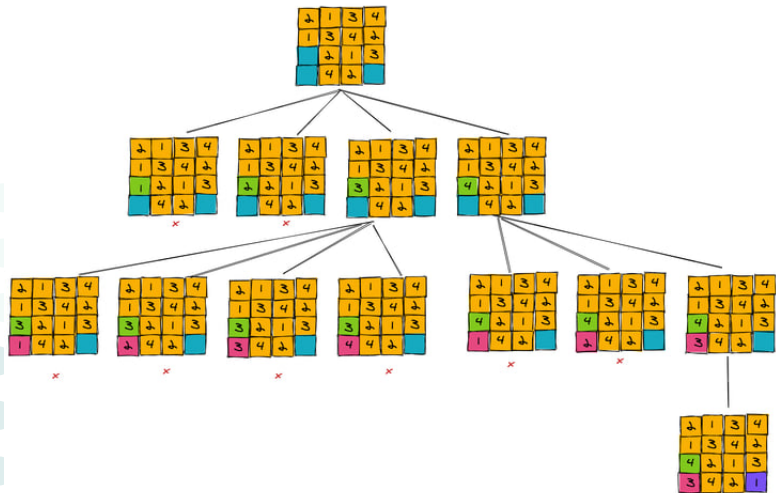
Queremos hacer un algoritmo de backtracking que lo resuelva:

- ▶ En cada casillero libre (azules) puede poner números del 1 al 4
- ▶ Como queremos evaluar todas las opciones posibles probamos con todas las combinaciones de numeros

Sudoku

2	1	3	4
1	3	4	2
	2	1	3
	4	2	

Sudoku 4x4
Fill in the blue cells
Rule:
No number is repeated
in a row or column



Sudoku

- ▶ Soluciones **parciales**: todos los tableros posibles a medida que los voy completando
- ▶ Soluciones **candidatas**: los tableros llenos, sean o no válidos (en este caso solo tenemos el válido)
- ▶ Soluciones **válidas**: el sudoku completado correctamente que cumple todas las reglas

Sudoku

En el dibujo lo que está marcado con cruces rojas lo llamamos podas. Una poda es cortar ramas del árbol que no nos llevan a soluciones válidas. En este caso "podamos" cuando el número que intentamos poner ya está ubicado en otro lugar de la misma fila o columna

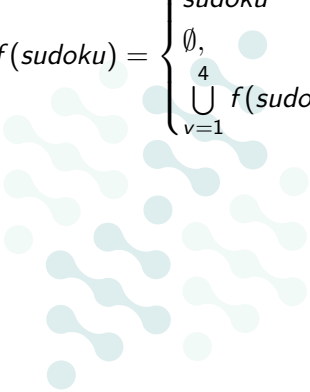
Sudoku

Hagamos la función recursiva:



Sudoku

Hagamos la función recursiva:


$$f(sudoku) = \begin{cases} sudoku & \text{si no hay espacios vacíos y es válido} \\ \emptyset, & \text{si no hay espacios vacíos pero es inválido} \\ \bigcup_{v=1}^4 f(sudoku \cup \{(i,j) \rightarrow v\}) & \text{si } (i,j) \text{ es la primera casilla vacía} \end{cases}$$

Sudoku

Hagamos la función recursiva:

$$f(sudoku) = \begin{cases} sudoku & \text{si no hay espacios vacíos y es válido} \\ \emptyset, & \text{si no hay espacios vacíos pero es inválido} \\ \bigcup_{v=1}^4 f(sudoku \cup \{(i,j) \rightarrow v\}) & \text{si } (i,j) \text{ es la primera casilla vacía} \end{cases}$$

Las podas no se suelen poner en la funciones recursivas (muchas veces no se puede), éstas se agregan en el código.

Sudoku

Sea n la cantidad de casilleros libres, ¿cuál es la complejidad de esta idea? ¿Podría ser mejor si agregamos más podas?



Tenemos un CD que soporta P minutos de música. Dado un conjunto con N canciones de duración p_i ($1 \leq i \leq N$) queremos encontrar la mayor cantidad de minutos de música que podemos poner en el CD.

Tenemos una cadena I de caracteres que son letras en mayúscula o comodines. Queremos saber cuántas cadenas podemos formar si reemplazamos comodines por mayúsculas teniendo en cuenta que:

- ▶ No queremos 3 vocales ni 3 consonantes seguidas.
- ▶ Tiene que haber por lo menos una L en la palabra.

Asumimos dada una función que resuelve en $O(1)$:

$$\text{validar}(I) = \begin{cases} 1 & \text{si la palabra es válida} \\ 0 & \text{c.c.} \end{cases}$$

Viaje en bondi

Pancho se mudó a una nueva ciudad y quiere saber si es posible llegar desde su casa a la facultad solo usando combinaciones de colectivos, sin caminar entre dos paradas. Tiene un mapa de todas las paradas de colectivo de la ciudad y los trayectos de todos los colectivos.