

Programación Avanzada

IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán



Anuncios

1. Hoy es la última clase expositiva con el método *flipped-classroom*.
2. Hoy tenemos la última experiencia.
3. ¡Próxima semana tendremos una modalidad especial!

Tópicos Avanzados 1

Esta semana les presentamos muchas herramientas avanzadas

- Expresiones regulares
- Grafos
- Búsqueda en grafos
- Pandas
- Numpy

¿Por qué creen que son importantes?
Discutamos...

Expresiones Regulares (RegEx)



A veces necesitamos encontrar patrones en nuestros strings bastante específicos.

Para estos casos, podríamos tener funciones de varias líneas haciendo uso de métodos de string y procesando el string de forma progresiva...

O usar una Expresión Regular, también llamadas RegEx 😎.

Qué es una RegEx

Patrones de búsqueda, compuestos de secuencias de caracteres especializados, aplicables a *strings*. El objetivo es revisar si el patrón se encuentra una o más veces dentro del *string* a analizar.

Esto permite que la validación de un *string* para que siga un formato definido sea más concisa.

En **Python**, usaremos la librería **re** para trabajar con RegEx.

Caracteres básicos de una RegEx

| Sintaxis | Descripción |
|----------|-------------------------------|
| [] | Clases de caracteres |
| + | Puede estar 1 o más veces |
| * | Puede estar 0 o más veces |
| ? | Puede estar a lo más 1 vez |
| {m, n} | Puede estar entre m y n veces |
| . | Comodín (cualquier carácter) |

| Sintaxis | Descripción |
|----------|--|
| ^ | Inicio del <i>string</i> |
| \$ | Final del <i>string</i> |
| () | Agrupar |
| | OR |
| \ | Escapar caracteres especiales para usarlos |

Caracteres básicos de una RegEx

| Sintaxis | Descripción |
|-----------------------|---|
| <code>\s</code> | Cualquier espacio en blanco |
| <code>[a-z]</code> | Cualquier letra minúscula entre a y z |
| <code>[a-zA-Z]</code> | Cualquier letra entre A y Z, sin importar mayúscula o minúscula |
| <code>[0-9]</code> | Cualquier dígito entre 0 y 9 |
| <code>[^arn]</code> | Cualquiera EXCEPTO a, r, n |

Y muchos más...

Algunos ejemplos

Validar un rut:

`[0-9]{1,2}\.[0-9]{3}\.[0-9]{3}-([0-9kK])`

- **`[0-9]{1,2}`** Un número de 1 o 2 dígitos, que pueden ir entre 0 y 9 cada dígito.
- **`\.`** Seguido de un punto ‘.’.
- **`[0-9]{3}`** Número de 3 dígitos, que pueden ir entre 0 y 9 cada dígito
- **`\.`** Seguido de otro punto.
- **`[0-9]{3}`** Número de 3 dígitos, que pueden ir entre 0 y 9.
- **`-`** Seguido de un guión ‘-’.
- **`([0-9kK])`** Un dígito entre 0 y 9, o una K (mayúscula o minúscula).

Algunos ejemplos

Validar un correo con formato específico:

[a-zA-Z0-9_.]+@((seccion1|seccion2)\.)?(mimail|mail)\.cl

- **[a-zA-Z0-9_.]+** Tiene que haber una letra, dígito, guion bajo o punto por lo menos una vez.
- **@** Luego un arroba
- **((seccion1|seccion2)\.)?** Luego puede haber o no haber un “seccion1.” o “seccion2.” (no ambos).
- **(mimail|mail):** Luego debe haber un “mimail” o un “mail” (no ambos)
- **\.cl:** Finalizar con .cl

Algunos ejemplos

Validar un correo con formato específico:

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:2(5[0-5])|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])\.\.){3}(?:2(5[0-5])|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0-9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\\])
```

Fuente: [How can I validate an email address using a regular expression? - Stack Overflow](#)

Algunos

Validar un correo electrónico

```
(?:[a-z0-9!#$%&'*+
x08\x0b\x0c\x0e-\
")@(?:[a-z0-9!#$%&
(2(5[0-5]|[0-4][0-
0-9]|[1-9]?[0-9])?)
```

```
[+)*|"(?:[\x01-\x0c\x0e-\x7f])*
-z0-9])?|\\(?:[a-z
4[0-9])|1[0-9][0-
.f\x21-\x5a\x53]
```

Fuente: [How can I validate an email address?](#)

RegEx no es trivial

Por eso hay muchas páginas que ayudan a entender qué diablos significa una expresión o cómo diablos escribo una:

- <https://regexr.com/>
- <https://regex101.com/>
- <https://www.regextester.com/>

Usar RegEx en Python

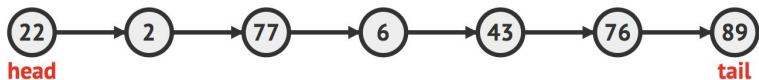
El módulo **re** nos permite aplicar RegEx para analizar *strings* en nuestro código:

```
re.match(patron, string) # Encontrar la primera ocurrencia desde el inicio  
re.fullmatch(patron, string) # Todo el string cumple el patrón  
re.search(patron, string) # Encontrar en cualquier lado el patrón  
re.sub(patron, reemplazar_por, string) # Reemplazar un patrón  
re.split(patron, string) # Separar según el patrón
```

Estructuras nodales: grafos

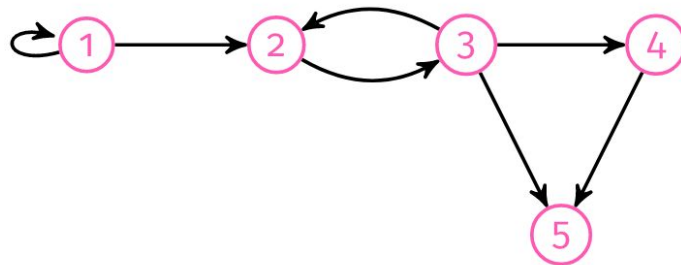
Estructuras nodales

Presentan un orden para navegar



```
self.siguiete = ...
```

No presentan un orden para navegar



```
self.vecinos = [...]
```


Representación de grafos

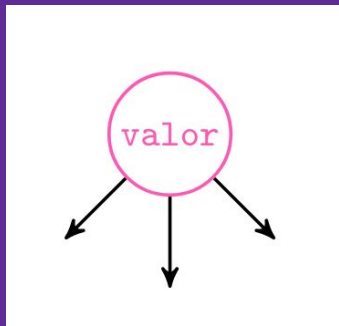
- Representación con nodos
- Lista de adyacencia
- Matriz de adyacencia

Representación con nodos

```
class Nodo:
```

```
    def __init__(self, valor=None):  
        self.valor = valor  
        self.vecinos = list()
```

```
    def agregar_vecino(self, vecino):  
        self.vecinos.append(vecino)
```



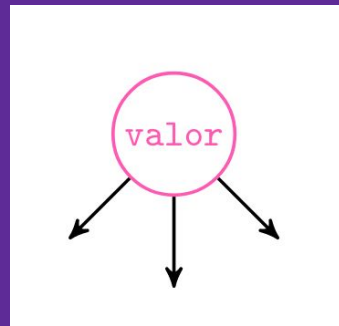
```
nodo_1 = Nodo(1)  
nodo_2 = Nodo(2)  
nodo_3 = Nodo(3)  
nodo_1.agregar_vecino(nodo_2)
```

Representación con nodos

```
class Nodo:

    def __init__(self, valor=None):
        self.valor = valor
        self.vecinos = list()

    def agregar_vecino(self, vecino):
        self.vecinos.append(vecino)
```



¿Cómo podemos representar el grafo completo en un solo objeto?

Representación con nodos

```
class Grafo:  
  
    def __init__(self):  
        self.nodos = list()
```

Representación con nodos

```
class Grafo:
```

```
    def __init__(self):  
        self.nodos = list()
```

*¿Cómo podemos representar el grafo
completo en un solo objeto?*

Lista de adyacencia

```
grafo = {  
    1: [2, 3],  
    2: [4, 5],  
    3: [],  
    4: [5],  
    5: [4],  
}
```

```
nodos = list(grafo.keys())
```

Lista de adyacencia

```
class Grafo:
    def __init__(self):
        self.listas_de_adyacencia = dict()

    def agregar_nodo(self, valor):
        if valor not in self.listas_de_adyacencia:
            self.listas_de_adyacencia[valor] = list()

    def agregar_conexion_dirigida(self, valor_1, valor_2):
        self.agregar_nodo(valor_1)
        self.agregar_nodo(valor_2)
        self.listas_de_adyacencia[valor_1].append(valor_2)
```

Lista de adyacencia

```
class Grafo:
    def __init__(self):
        self.listas_de_adyacencia = dict()

    def agregar_nodo(self, valor):
        if valor not in self.listas_de_adyacencia:
            self.listas_de_adyacencia[valor] = list()

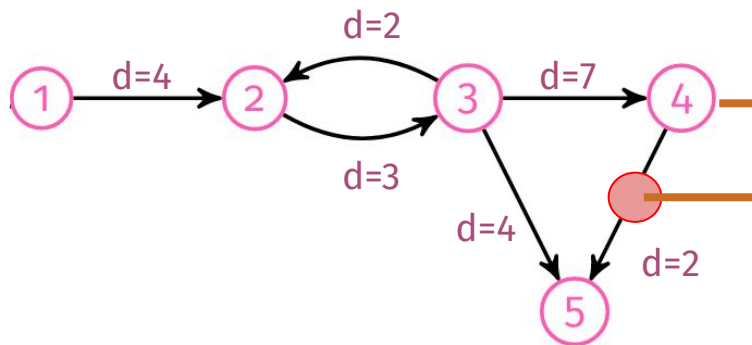
    def agregar_conexion_dirigida(self, valor_1, valor_2):
        self.agregar_nodo(valor_1)
        self.agregar_nodo(valor_2)
        self.listas_de_adyacencia[valor_1].append(valor_2)
```

¿Existe otra forma aparte de la lista de adyacencia?

Matriz de adyacencia

```
grafo = [  
    [0, 1, 1, 0, 0],  
    [0, 0, 0, 1, 1],  
    [0, 0, 0, 0, 0],  
    [0, 0, 0, 0, 1],  
    [1, 0, 0, 1, 0]  
]
```

Representación basada en nodos con pesos



```
class Lugar:
```

```
    def __repr__(self, nombre):
```

```
        self.nombre = nombre
```

```
        self.calles = list()
```

```
    def agregar_calle(self, distancia, destino):
```

```
        self.calles.append((distancia, destino))
```

```
class Ciudad:
```

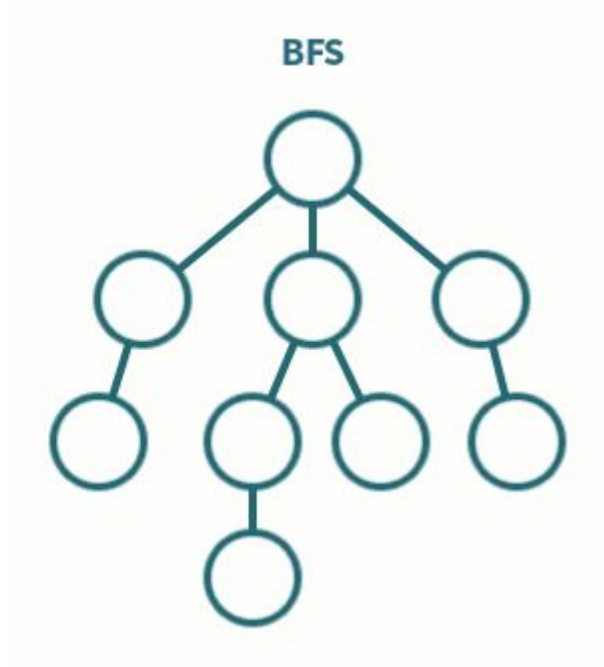
```
    def __repr__(self):
```

```
        self.lugares = list()
```

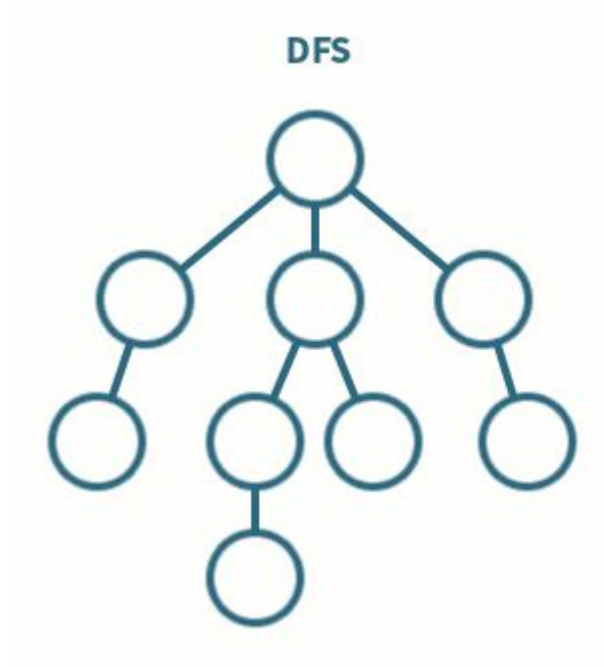
Recorrido de grafos

- BFS, búsqueda en amplitud
- DFS, búsqueda en profundidad

BFS vs DFS (en árboles)



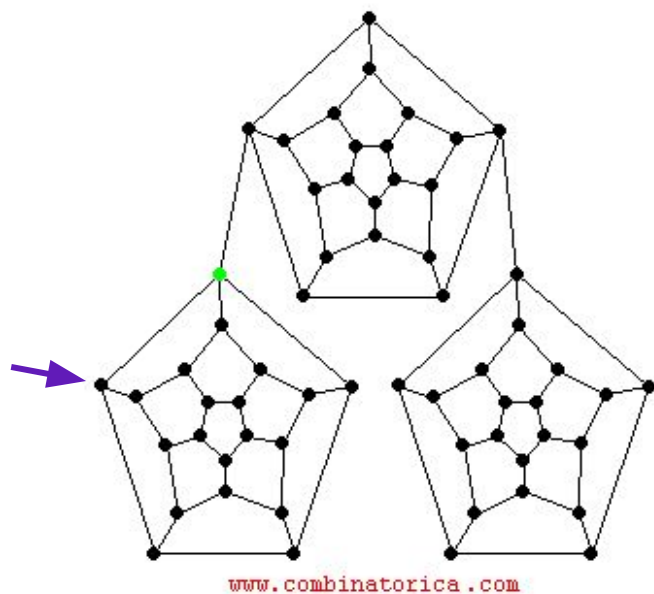
Búsqueda por amplitud



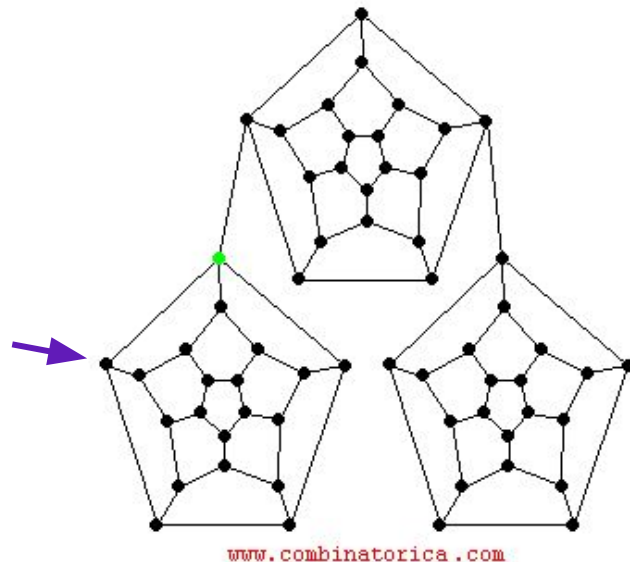
Búsqueda por profundidad

BFS vs DFS (en grafos)

Breadth-First Search

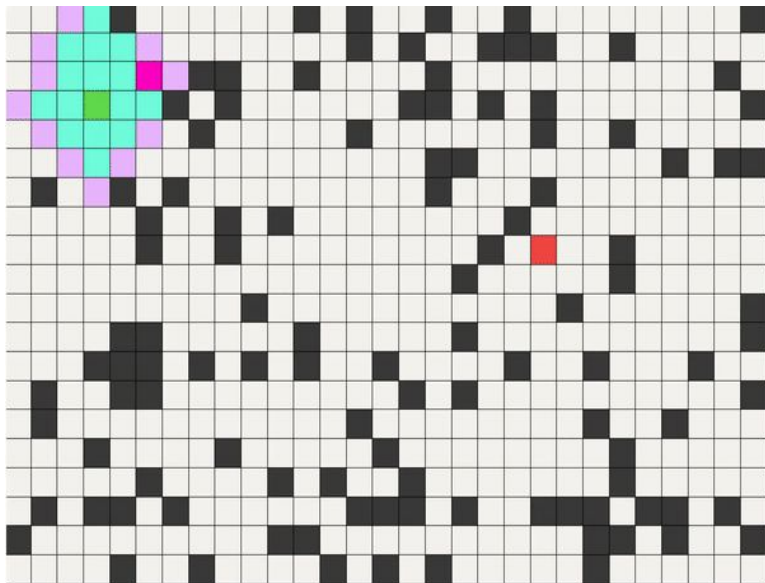


Depth-First Search

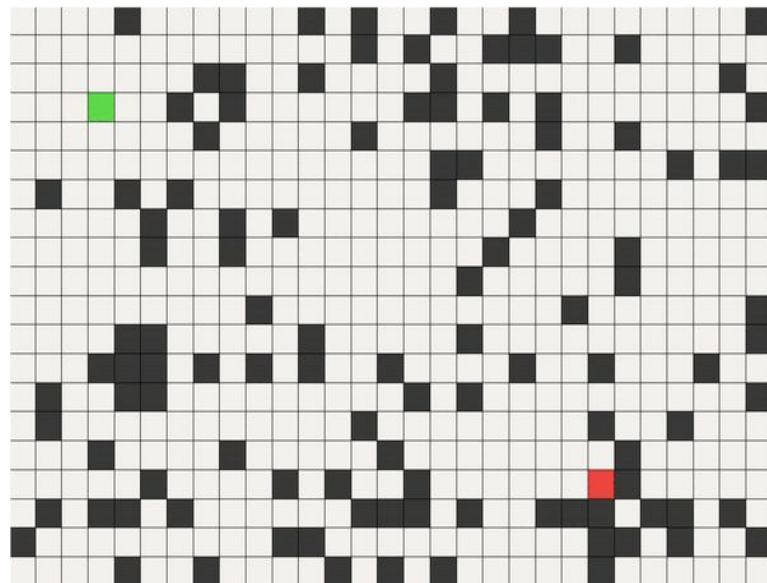


BFS vs DFS

Encontrar una ruta



BFS



DFS

El auto de Cristian Ruz

El auto de **Cristian Ruz** está malo, y se calienta si anda en calles de 5 km o más. Por lo tanto, solo puede transitar por calles cortas (**< 5 km**), descansa un rato y puede seguir transitando por otra calle corta.

Escribe un método que desde un **lugar inicial** entregue todos los **lugares alcanzables** con el auto de **Cristian Ruz**.

Representación en nodos con pesos

```
class Lugar:
```

```
    def __init__(self, nombre):  
        self.nombre = nombre  
        self.calles = list()
```

```
    def agregar_calle(self, destino):  
        self.calles.append(destino)
```

```
class Ciudad:
```

```
    def __init__(self):  
        self.lugares = lugares
```


Representación en nodos con pesos

```
class Lugar:
```

```
    def __init__(self, nombre):  
        self.nombre = nombre  
        self.calles = list()
```

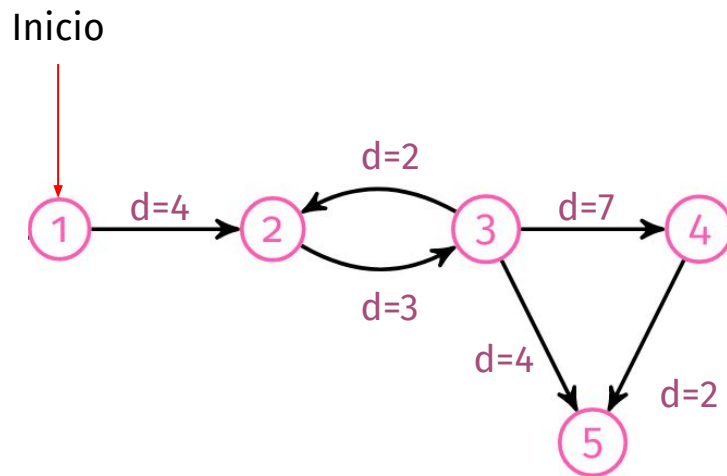
```
    def agregar_calle(self, distancia, destino):  
        self.calles.append(  
            (distancia, destino)  
        )
```

```
class Ciudad:
```

```
    def __init__(self):  
        self.lugares = lugares
```

Representación basada en nodos con pesos

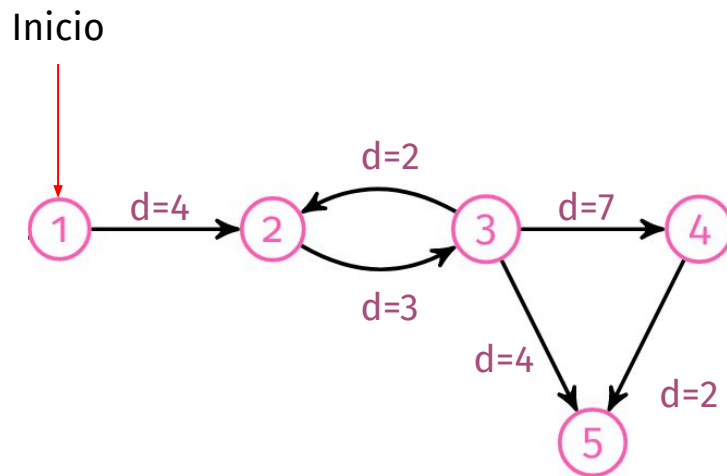
```
class Ciudad:  
    def cruz_alcanza(self, inicio):  
        pass  
  
    return visitados
```



Representación basada en nodos con pesos

Creamos nuestro *stack* para guardar los nodos a visitar, y un *set* visitados para guardar los lugares ya visitados.

```
class Ciudad:  
  
    def cruz_alcanza(self, inicio):  
        visitados = set()  
        stack = [inicio]  
  
  
        return visitados
```

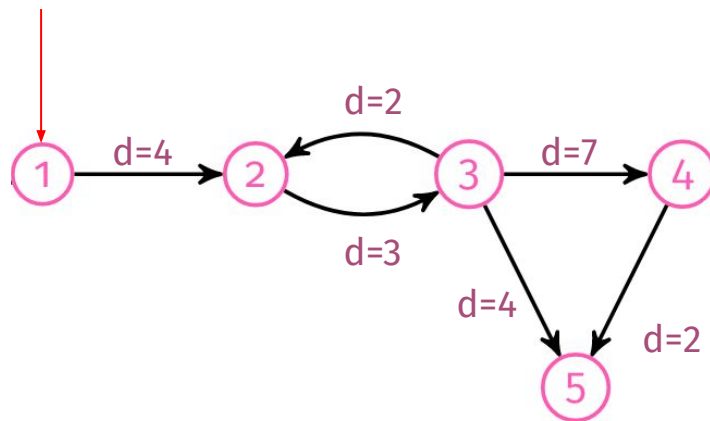


Representación basada en nodos con pesos

Mientras existan lugares por visitar en el *stack*, sacamos dicho lugar del *stack* para analizarlo.

```
class Ciudad:  
  
    def cruz_alcanza(self, inicio):  
        visitados = set()  
        stack = [inicio]  
        while len(stack) > 0:  
            lugar = stack.pop()  
  
        return visitados
```

Inicio



Representación basada en nodos con pesos

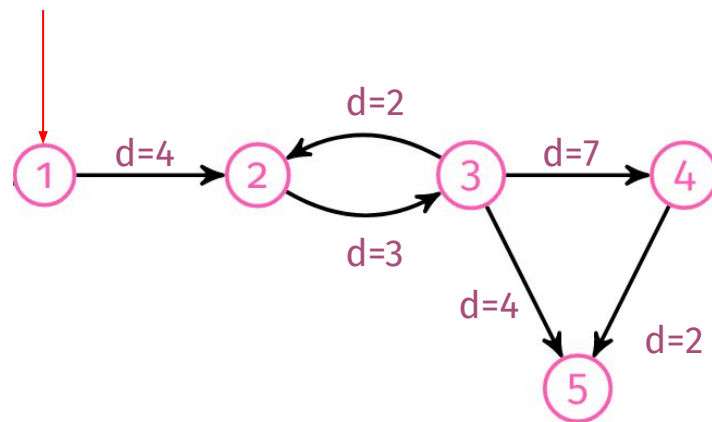
Si el lugar no ha sido visitado, lo agregamos a nuestro set de visitados.

```
class Ciudad:
```

```
    def cruz_alcanza(self, inicio):  
        visitados = set()  
        stack = [inicio]  
        while len(stack) > 0:  
            lugar = stack.pop()  
            if lugar not in visitados:  
                visitados.add(lugar)
```

```
        return visitados
```

Inicio



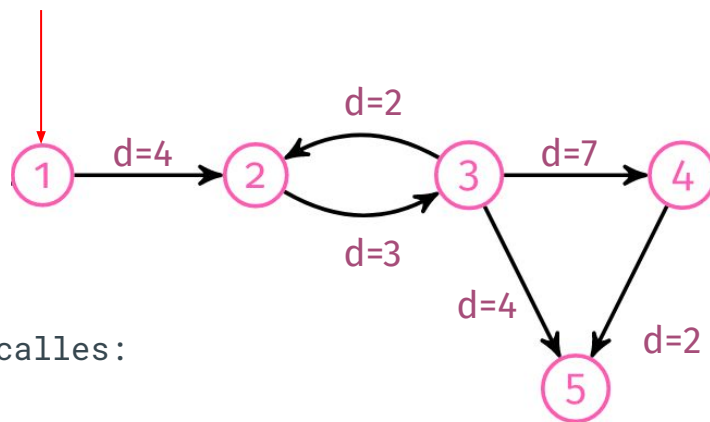
Representación basada en nodos con pesos

Ahora vamos a analizar los vecinos de ese lugar, si los agregamos a nuestro *stack* de lugares a visitar o no.

```
class Ciudad:
```

```
    def cruz_alcanza(self, inicio):  
        visitados = set()  
        stack = [inicio]  
        while len(stack) > 0:  
            lugar = stack.pop()  
            if lugar not in visitados:  
                visitados.add(lugar)  
                for (distancia, vecino) in lugar.calles:  
                    pass  
  
        return visitados
```

Inicio



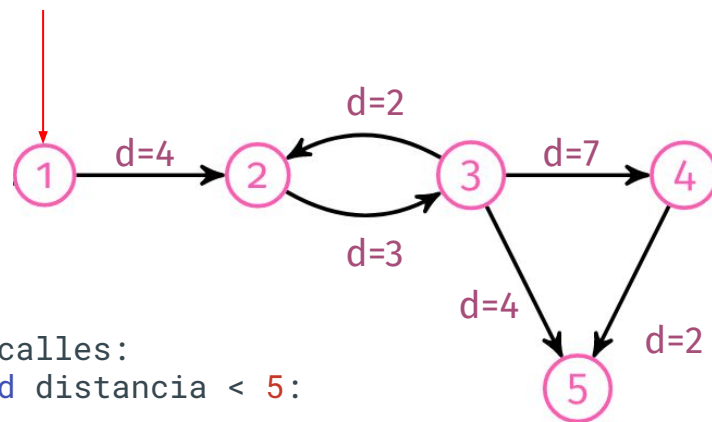
Representación basada en nodos con pesos

Si el vecino no ha sido visitado y está a menos de 5 KM, entonces es un lugar visitable y lo agregamos al *stack*.

```
class Ciudad:
```








```
def cruz_alcanza(self, inicio):  
    visitados = set()  
    stack = [inicio]  
    while len(stack) > 0:  
        lugar = stack.pop()  
        if lugar not in visitados:  
            visitados.add(lugar)  
            for (distancia, vecino) in lugar.calles:  
                if vecino not in visitados and distancia < 5:  
                    stack.append(vecino)  
    return visitados
```

Inicio



Pandas

PANDAS VS ALTERNATIVES

| | What it is | Good for | Not best fit for |
|--|---|--|---|
|  pandas | Go-to Python library for data analysis | Data manipulation and further analysis in different domains | Very large datasets, unstructured data |
|  NumPy | Python library for numerical computing | Mathematical operations on arrays and matrices | Non-numerical data types, data manipulation tasks |
|  PySpark | Python API for Apache Spark | Big data processing in distributed environment | Small-scale data tasks |
|  dask | Python library for parallel and distributed computing | Processing of larger-than-memory datasets | Data manipulation tasks |
|  MODIN | Python library for distributed computing of Pandas DataFrames | Manipulating datasets from 1MB to 1TB+ | Small-scale data tasks |
|  vaex.io | Python library for larger-than-memory Pandas DataFrames | Visualizing and exploring big tabular datasets | Data manipulation tasks |
|  R | Statistical programming language | Data mining, data wrangling, data visualization, machine learning operations | Basic data manipulation tasks, big data projects |

Pandas

- Pandas es una herramienta ampliamente utilizada en análisis de datos. Nos ayuda a organizar y simplificar operaciones grandes sobre datos tabulados. Eso sí, está diseñado para ser usado en Python.
- Trabaja con filas de datos heterogéneos, y columnas que se caracterizan por tener una **etiqueta** denominadora.
- Debe instalarse ya que no viene por defecto con la instalación de Python.
- Usualmente se usa el alias pd.

```
pip install pandas
```

```
import pandas as pd
```

Pandas

- Podemos crear Dataframes desde diccionarios.
- Por convención las variables resultantes de crear un Dataframe se denominan **df**.

```
datos_libros = {  
    'Título': ['The Martian', 'Red Rising', 'Mockingjay'],  
    'Año_publicacion': [2011, 2016, 2010],  
    'Autor': ['Andy Weir', 'Pierce Brown', 'Suzanne Collins']  
}  
df_libros = pd.DataFrame(datos_libros)
```

| | Título | Año_publicacion | Autor |
|---|-------------|-----------------|-----------------|
| 0 | The Martian | 2011 | Andy Weir |
| 1 | Red Rising | 2016 | Pierce Brown |
| 2 | Mockingjay | 2010 | Suzanne Collins |

Pandas

| | Título | Año_publicacion | Autor |
|---|-------------|-----------------|-----------------|
| 0 | The Martian | 2011 | Andy Weir |
| 1 | Red Rising | 2016 | Pierce Brown |
| 2 | Mockingjay | 2010 | Suzanne Collins |

- Los métodos más usados son **iloc**, que me permite acceder según el índice de la fila o la columna; y **loc** según la etiqueta de la fila o la columna.

```
df_libros.info()           # obtenemos información general
df_libros.head(N)          # Vemos las primeras N filas
df_libros.describe()       # Estadísticas descriptivas
```

```
df_libros.iloc[fila, columna] # Accede a todos los datos según su índice
df_libros.iloc[2,0]           # 'Mockingjay'
```

```
df_libros.loc[fila, columna] # Accede a todos los datos según su etiqueta
df_libros.loc[1, 'Autor']    # 'Pierce Brown'
```

```
df_libros['Título']          # Accede a toda columna, usando su etiqueta
```

Pandas

- También podemos agregar datos u ordenarlos.

```
# agregar nueva columna
```

```
df_libros.loc[:, 'Actual'] = df_libros['Año_publicacion'] > 2000  
df_libros
```

| | Título | Año_publicacion | Autor | Actual |
|---|-------------|-----------------|-----------------|--------|
| 0 | The Martian | 2011 | Andy Weir | True |
| 1 | Red Rising | 2016 | Pierce Brown | True |
| 2 | Mockingjay | 2010 | Suzanne Collins | True |

```
# ordenar datos según columnas
```

```
df_libros = df_libros[['Autor', 'Actual', 'Año_publicacion', 'Título']]  
df_libros
```

| | Autor | Actual | Año_publicacion | Título |
|---|-----------------|--------|-----------------|-------------|
| 0 | Andy Weir | True | 2011 | The Martian |
| 1 | Pierce Brown | True | 2016 | Red Rising |
| 2 | Suzanne Collins | True | 2010 | Mockingjay |

NumPy

NumPy

- Numpy es altamente usado para operaciones en vectores o matrices, y grandes conjuntos de datos numéricos.
- Además cuenta con una colección de funciones matemáticas de alto nivel para operar sobre estos datos.
- Un **array** de NumPy es una matriz multidimensional de elementos homogéneos, es decir, todos los elementos son del mismo tipo de datos.
- Viene por defecto con la instalación de Python.
- Usualmente se usa el alias np.

```
import numpy as np
```

NumPy

- Existen muchas formas de crear arrays.

```
np.array((1,2,3,4))          # creamos un array desde una tupla.  
np.array([1,2,3,4])          # creamos un array desde una lista.  
np.array([[1, 2, 3], [4, 5, 6]]) # matriz desde lista de listas.
```

```
np.arange(inicio,fin,paso) # Crea un arrange desde un rango.  
np.linspace(0, 1, 5)      # 5 números equidistantes entre 0 y 1  
np.zeros((3, 4))          # Crea array de 3x4 de ceros.  
np.ones((2, 3))           # Crea array de 2x3 de unos.  
np.eye(4)                 # Crea una matriz identidad de 4x4.  
np.random.random((2, 3))  # Arrays 2x3 con números aleatorios entre 0 y 1.
```


NumPy

- Finalmente, una vez obtenido el array puedes realizar diversas operaciones con él o con otros arrays.
- Funciones clásicas que nos sirven por ejemplo para matrices de adyacencia es el producto punto. Entre dos vectores A y B se calcula como

$$A \cdot B = A_1 \times B_1 + A_2 \times B_2 + \dots + A_n \times B_n$$

en NumPy

```
matriz1 = np.array([[1, 2], [3, 4]])
```

```
matriz2 = np.array([[5, 6], [7, 8]])
```

```
producto_matrices = matriz1.dot(matriz2) # array([[19, 22], [43, 50]])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \longrightarrow \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Programación Avanzada

IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattán

