

Estructuras de datos

Nombre: Alexandra Valeria Hernández Quintero

Matricula: 1663507

Unidad de aprendizaje: Matemáticas Computacionales

Fecha: 06/10/2017

Resumen: Este reporte trata sobre las estructuras de datos; Pila, Fila, Grafo búsquedas por amplitud y profundidad (BFS y DFS).

Pila

Explicar que es una fila en Python, es más sencillo si nos imaginamos primero gráficamente una pila, literalmente una pila es un contenedor de objetos, donde dichos objetos pueden ser diferentes, similares, del mismo tipo e incluso. En Python este concepto no cambia mucho, la diferencia es que nuestro “contenedor” son espacios de memoria y los “objetos” son datos.

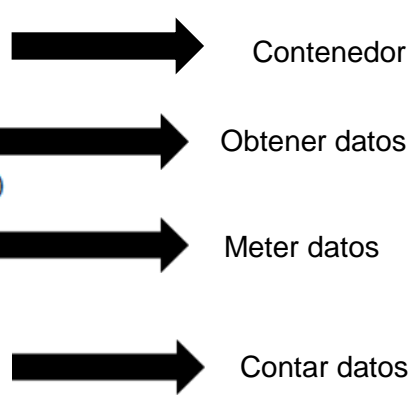
¿Cómo hacer una pila en Python?

Pues como se mencionó anteriormente, si los hacemos de forma literal, lo primero para crear nuestra pila sería hacer un contenedor, que esto en Python podríamos tomar un arreglo como nuestro “contenedor” e inicializarlo, es decir que un principio no tiene nada, está vacío. Después, para llenar nuestro “contenedor” podemos usar una función para obtener “objetos”, es decir, los datos para llenar el arreglo.

Una vez que ya tenemos lleno nuestro “contenedor”, necesitaremos los “objetos” que tiene dentro, y para esto se crea una función para obtener los datos que están dentro del arreglo para así poder manipularlos. Además de eso habrá ocasiones en que necesitemos saber cuántos “objetos” tiene nuestro “contenedor”, para esto creamos otra función que nos ayude a contar los datos que están dentro del arreglo.

Pseudocódigo

```
class Pila:
    definir __inicializar__(self):
        self.pila = []
    definir obtener(self):
        regresar self.pila.surgir()
    definir meter(self, e):
        self.pila.llenar(e)
        regresar tamaño(self.pila)
    @propiedad
    definir longitud(self):
        regresar tamaño(self.pila)
```



Fila

De la misma forma que en pila, antes de explicar que es una fila en Python, imaginemos una fila, puede ser de personas por ejemplo. En una fila todas las personas son diferentes, o pueden ser similares. En Python una fila claro esta no es

de personas, esta fila es de datos, que dichos datos al igual que en pila se guardan en espacios de memoria.

¿Cómo hacer una fila en Python?

Utilizando la analogía anterior de la fila de las personas. Primero necesitamos un “lugar” donde poner a las “personas”, es decir que necesitamos un arreglo para poner nuestros datos, al igual que en pila en un principio no hay nada en nuestro arreglo, es por eso que debemos inicializarlo.

Después iremos poniendo “personas” en el “lugar” señalado para hacer nuestra fila, esto quiere decir que debemos de crear una función para llenar el arreglo con datos, para después con otra función traer (obtener) datos de nuestro arreglo y manipular dichos datos.

En algunas ocasiones en la vida real, se requiere contar las personas que están en una fila, esto también pasa en nuestra fila de Python, y para eso se crea una función que nos regrese el número de elementos de nuestra fila.

Pseudocódigo

```
clase Fila:
    definir __inicializar__(self):
        self.fila = []
    definir obtener(self):
        regresar self.fila.surgir()
    definir meter(self, e):
        self.fila.insertar(0,e)
        regresar tamaño(self.fila)
    @propiedad
    definir longitud(self):
        regresar tamaño(self.fila)
```

→ Lugar

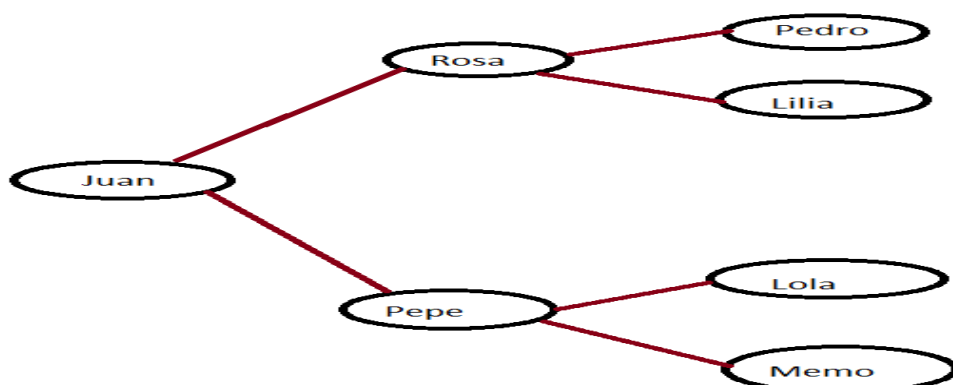
→ Obtener datos

→ Meter datos

→ Contar datos

Grafo

¿Qué es un grafo?, pues de la forma más sencilla, un grafo es un conjunto de vértices (nodos) y aristas (las que conectan a los vértices), esta definición es un poco ambigua y no nos ayuda a visualizar que es un grafo. Como se hizo anteriormente utilizaremos una analogía para comprender mejor y poder visualizar a un grafo. Imaginemos un árbol genealógico de la familia Pérez, donde Juan tiene dos hijos y tiene 4 nietos tal como se muestra en la siguiente imagen:



Consideremos el árbol genealógico de la familia Pérez, como si fuera un grafo, donde nuestros nodos serían los nombres de cada miembro de la familia y las aristas las líneas rojas que unen a cada padre con su respectivo hijo.

¿Cómo hacer un grafo en Python?

Primero debemos inicializar un lugar donde irán nuestros vértices y otro lugar donde irán nuestras futuras aristas (uniones entre un vértice y otro). Después tenemos que definir una función que agregue los vértices y otra función que conecte dichos vértices.

Pseudocódigo

```
clase Grafo:
    definir __inicializar__(self):
        self.V=conjunto()
        self.A=diccionario()
        self.hijos=diccionario()
    definir agrega(self, v):
        self.V.añade(v)
        si v no esta en self.hijos:
            self.hijos[v]=conjunto()
    definir conecta(self, v, u, peso=1):
        self.agrega(v)
        self.agrega(u)
        self.A[(v, u)] = self.A[(u, v)] = peso
        self.hijos[v].añade(u)
        self.hijos[u].añade(v)
    @propiedad
    definir complemento(self):
        comp = Grafo()
        para cada v en self.V:
            para cada w en self.V:
                si v != w y (v, w) no esta en self.A:
                    comp.conecta(v, w, 1)
```

} Inicializar los vértices y aristas.

} Agrega vértices y pregunta si no está en hijos de tal vértice

} Conecta los vértices e indica quien es hijo de tal vértice

Nota: complemento aún no se utiliza.

BFS (Búsqueda por amplitud).

Este algoritmo sirve para explorar un grafo, este tipo de búsqueda nos proporciona la ruta más corta, al usar una fila explorará la anchura de una profundidad de vértice antes de seguir adelante. Este comportamiento garantiza que la primera ruta ubicada sea una de las “distancias” más cortas presentes, en función del número de aristas que unen a los vértices.

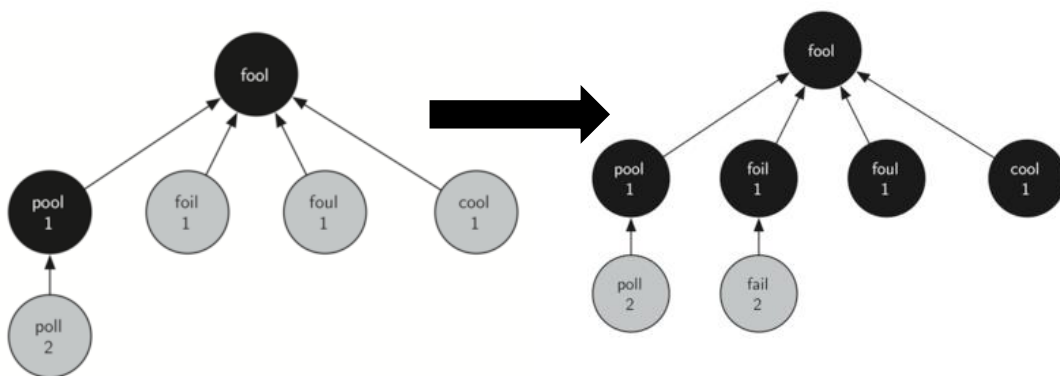
Una buena manera de visualizar lo que hace el BFS es imaginar que estamos construyendo un árbol, un nivel del árbol a la vez. Una primera búsqueda de amplitud

añade todos los hijos del vértice inicial antes de que empiece a descubrir a alguno de los nietos.

¿Cómo funciona BFS?

Podríamos decir que BFS colorea cada uno de los vértices blanco, gris o negro. Todos los vértices se inicializan en blanco cuando se construyen. Un vértice blanco es un vértice no descubierto. Cuando un vértice se descubre inicialmente es de color gris, y cuando BFS ha explorado por completo un vértice que es de color negro. Esto significa que una vez que un vértice es de color negro, no tiene vértices blancos adyacentes a él. Un nodo gris, por otro lado, puede tener algunos vértices blancos adyacentes a él, lo que indica que todavía hay vértices adicionales para explorar.

Como se muestra en la siguiente figura:



¿Cómo hacer una BFS en Python?

Se necesita una función que tenga como parámetros (datos de entrada) un grafo y un nodo inicial (el nodo con el cual queremos iniciar). Después hacemos uso de un arreglo para guardar nuestro nodo inicial y los nodos que se visitarán, también utilizaremos como herramienta auxiliar una Fila, así que mandamos llamar a Fila así podremos meter el nodo inicial dentro de nuestra Fila, usaremos un ciclo While para que repita las instrucciones de ir buscando nodos en el grafo, mientras que la longitud de nuestra fila sea mayor que cero, es decir que nuestra fila tenga elementos. Pero, ¿Qué instrucciones irán dentro del ciclo While? Pues bien, obtendremos de nuestra fila el nodo inicial, y vamos a meter a el nodo inicial dentro del arreglo de los nodos visitados, después usaremos como herramienta auxiliar un For para ir recorriendo cada nodo y dentro del For tendrá una condición que preguntara si el nodo ya está dentro de los visitados y si no agregara al nodo dentro de los visitados.

Pseudocódigo:

```

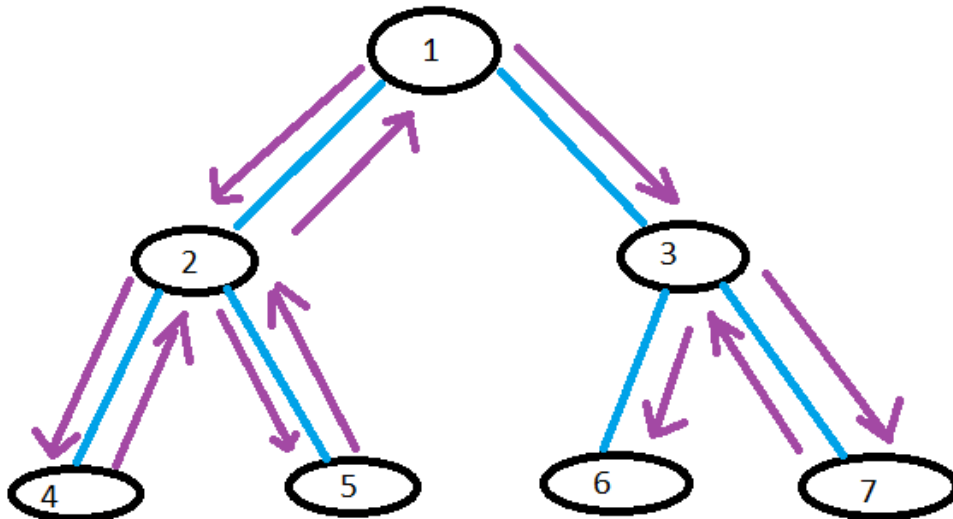
definir bfs(grafo, nodo_inicial):
    visitados=[nodo_inicial]
    f=Fila()
    f.meter(nodo_inicial)
    Mientras f.longitud>0:
        na= f.obtener()
        vecinos= grafo.vecinos[na]
        Para cada nodo en vecinos:
            si nodo no esta en visitados:
                visitados.llenar(nodo)
                f.meter(nodo)
    regresar visitados

```

DFS (Búsqueda por profundidad)

Al igual que la BFS es un algoritmo para explorar un grafo, pero a diferencia de BFS por lo general la DFS es la ruta menos corta, explora posibles vértices (de una raíz suministrada) por cada rama antes de retroceder. Se usa Pila para generar y devolver un conjunto de vértices que son accesibles dentro del componente conectado a los temas.

El DFS representada de forma gráfica sería:



¿Cómo hacer una DFS en Python?

Las instrucciones para realizar el código para la DFS, es muy parecido a las instrucciones de la BFS, se necesita una función que tenga como parámetros (datos de entrada) un grafo y un nodo inicial (el nodo con el cual queremos iniciar). Después hacemos uso de un arreglo para guardar los nodos que se visitaran, en vez de usar fila como lo hicimos en el BFS, usaremos como herramienta auxiliar pila.

Usaremos un ciclo While para que repita las instrucciones de ir buscando nodos en el grafo, mientras que la longitud de nuestra pila sea mayor que cero, es decir que nuestra pila tenga elementos. Dentro del ciclo While, obtendremos de nuestra pila el nodo inicial, habrá una condición que verifica si el nodo inicial esta en los visitados, si ya está en visitados continuara, luego llenaremos el arreglo, vamos a meter a el nodo inicial dentro del arreglo de los nodos visitados después usaremos como herramienta auxiliar un For para ir recorriendo cada nodo y dentro del For tendrá una condición que preguntará si el nodo ya está dentro de los visitados y si no agregará al nodo dentro de los visitados.

Pseudocódigo:

```
definir dfs(grafo, nodo_inicial):  
    visitados=[]  
    f=pila()  
    f.meter(nodo_inicial)  
    Mientras f.longitud>0:  
        na= f.obtener()  
        si na esta en visitados:  
            continuar  
        visitados.llenar(na)  
        vecinos= grafo.vecinos[na]  
        Para cada nodo en vecinos:  
            si nodo no esta en visitados:  
                f.meter(nodo)  
    regresar visitados
```