Tarea 8

May 14, 2023

1 Sea $d(x, y)$ la distancia más corta entre los vértices x y y. El diámetro de una gráfica $G = (V, A)$ se define como $\max d(x, y)\ \forall x, y \in V(G)$.

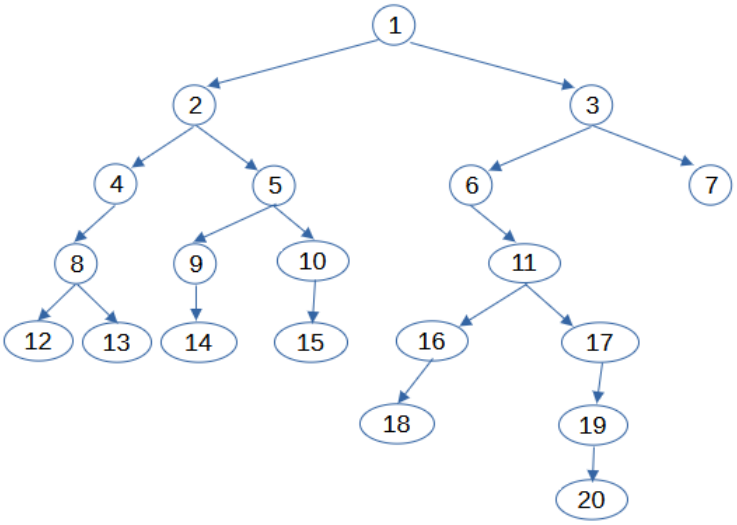
1.1 Diseñar un algoritmo que calcule en tiempo lineal el diámetro de un árbol. Justificar con sumo detalle la respuesta dada.

Podemos realizar una búsqueda en profundidad (DFS) desde cualquier vértice v del árbol y encontrar el vértice u más lejano de v. Luego, realizamos otra búsqueda en profundidad desde u para encontrar el vértice w más lejano de u. El diámetro del árbol será la distancia entre u y w.

Para implementar este algoritmo, podemos utilizar una sola función de búsqueda en profundidad. La idea es que, durante la búsqueda, mantengamos una variable $maxDist$ que almacene la distancia máxima encontrada hasta ese momento, así como otra variable $farthest$ que almacene el vértice más lejano del vértice actual. Cada vez que encontremos un vértice que tenga una distancia mayor a $maxDist$, actualizamos $maxDist$ y $farthest$.

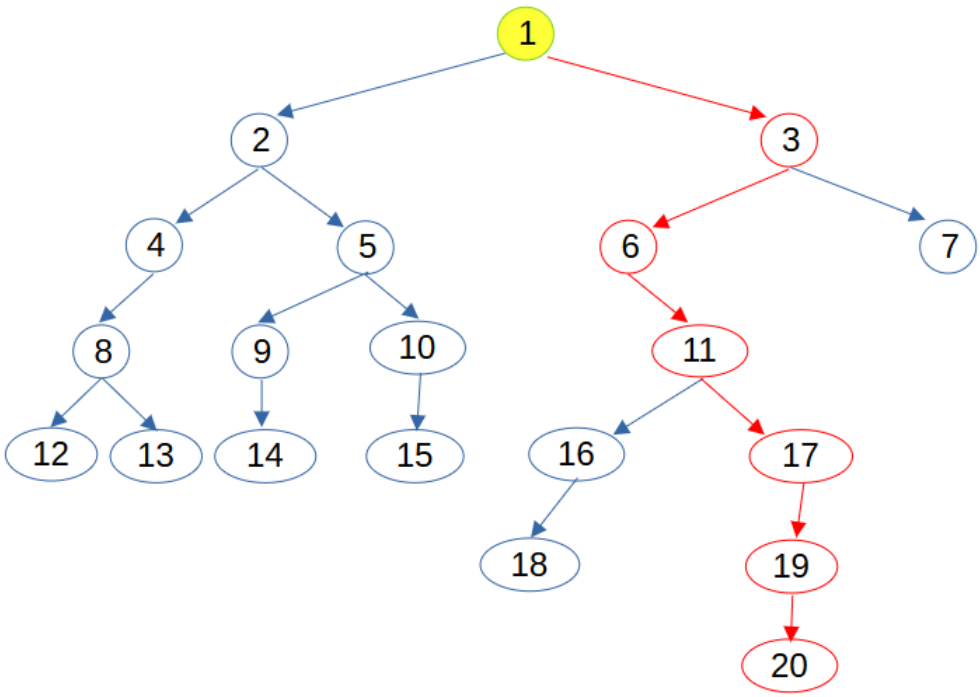
El algoritmo funciona en tiempo lineal ya que realiza dos recorridos en profundidad, cada uno de los cuales visita cada vértice del árbol exactamente una vez. Por lo tanto, el tiempo de ejecución es $O(n)$, donde n es el número de vértices del árbol.

1.2 Presentar un árbol de 20 vértices y aplicar el algoritmo

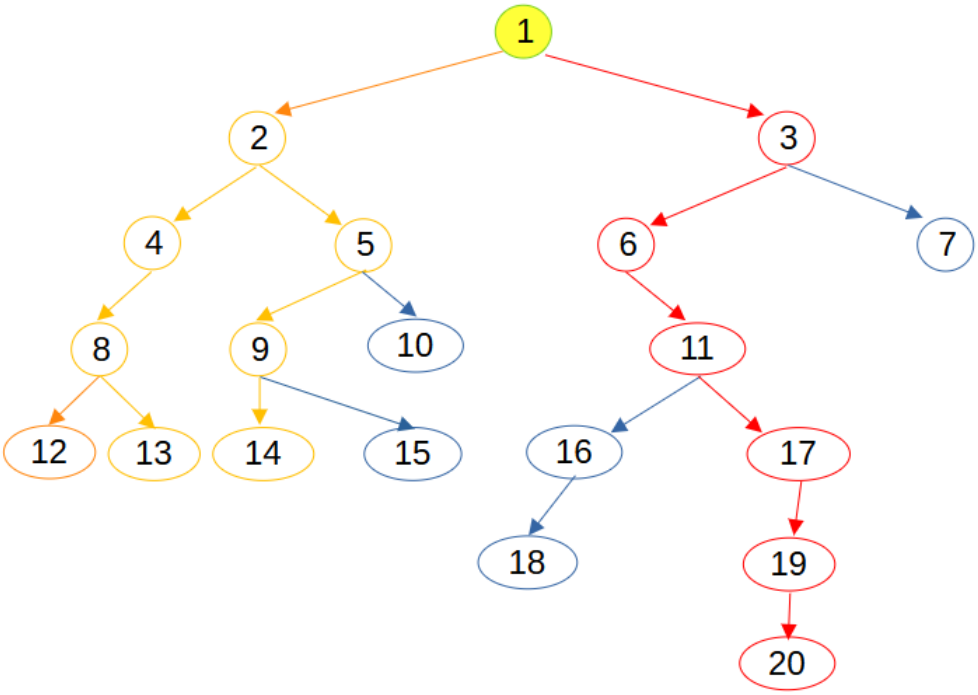


Arbol Inicial

Arbol aplicando DFS apartir del vértice inicial 1



Aqui tenemos 6 pasos.



Se aplica DFS desde el vértice 20, Como se llega al vértice inicial que fue el primero marcado, se sigue aplicando DFS, así se obtienen 4 niveles más.

Por lo tanto el diámetro del árbol es de $6 + 4 = 10$

2 Utilizar BFS o DFS para determinar las componentes conexas de una gráfica no conexa. Dada una gráfica G, su algoritmo debe ser capaz de indicar el conjunto de vértices de cada componente conexa.

Se puede usar el siguiente algoritmo:

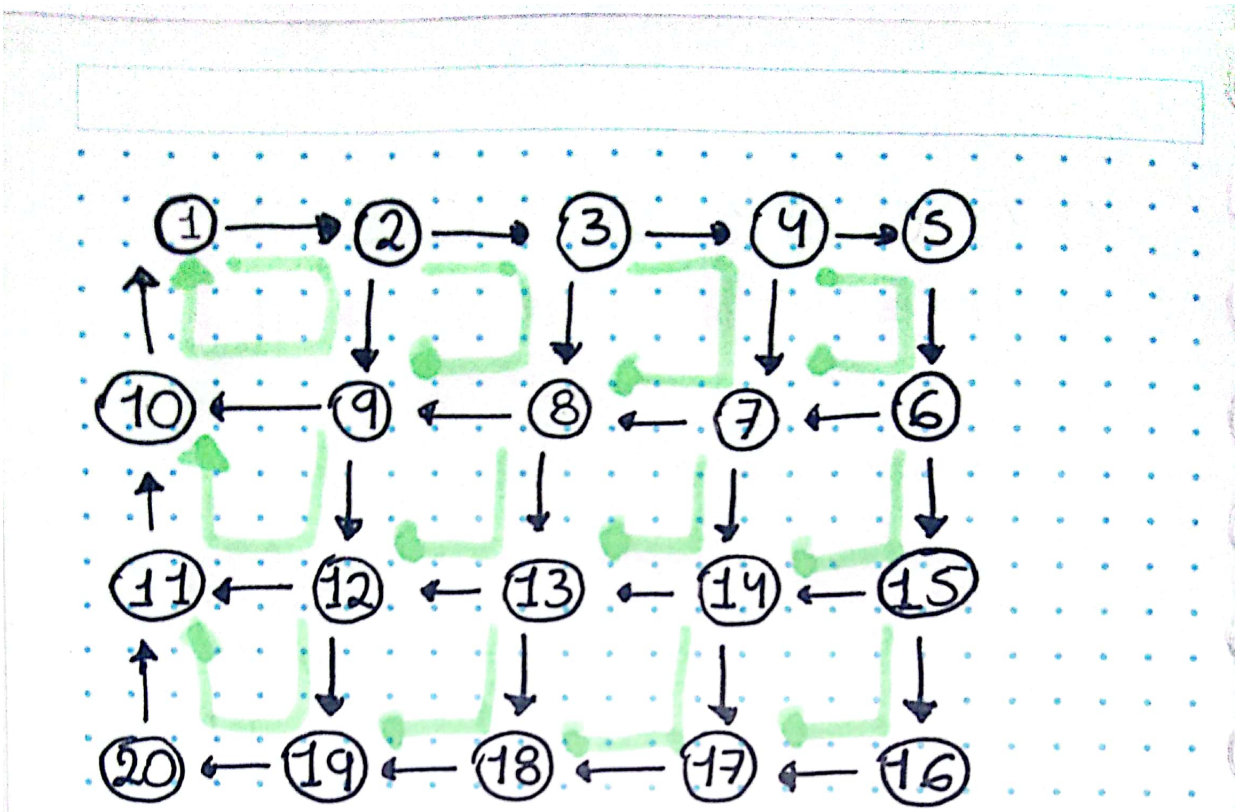
1. Inicializar un conjunto vacío para almacenar las componentes conexas.
2. Inicializar una lista vacía para almacenar los nodos visitados.
3. Para cada nodo no visitado en la gráfica, realizar lo siguiente:
   (a) Inicializar una lista vacía para almacenar los nodos de la componente conexa actual.
   (b) Realizar una búsqueda en anchura (BFS) desde el nodo actual, marcando cada nodo visitado y agregándolo a la lista de nodos de la componente conexa actual.
   (c) Agregar la lista de nodos de la componente conexa actual al conjunto de componentes conexas.
   (d) Agregar los nodos visitados a la lista de nodos visitados.
4. Devolver el conjunto de componentes conexas.

Pseudocodigo:

```
BFS_componentes_conexas(G):
  visitados = set()
  componentes_conexas = set()
  nodos_visitados = []

  for cada nodo n en G:
      if n no esta en visitados:
          componente_conexa_actual = []
          visitados.add(n)
          componente_conexa_actual.agregar(n)
          nodos_visitados.agregar(n)

          cola = Cola()
          cola.encolar(n)

          while cola no este vacia:
              nodo_actual = cola.desencolar()

              for cada vecino v de nodo_actual:
                  if v no esta en visitados:
                      visitados.add(v)
                      componente_conexa_actual.agregar(v)
                      nodos_visitados.agregar(v)
                      cola.encolar(v)

          componentes_conexas.agregar(conjunto_congelado(componente_conexa_actual))

  return componentes_conexas
```

3

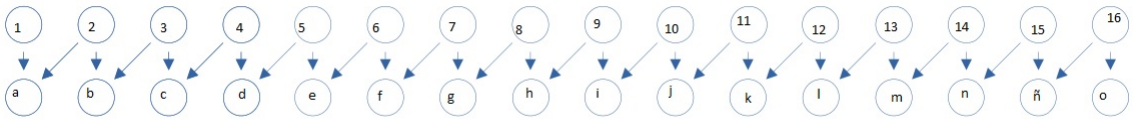
4 Construir una di-gáfica de al menos 20 vétices y 30 arcos donde no pueda aplicarse el Algoritmo Topological Sort e indique porque no puede aplicarlo.



No es posible dado a que la di-gráfica contiene ciclos.

Los ciclos se encuentran anotados con color.

5 Construir una di-gráfica de al menos 20 vértices y 30 arcos donde no donde si pueda aplicarse el Algoritmo Topological Sort y muestre el resultado de aplicarlo.



Resultado