UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100

Búsqueda en grafos

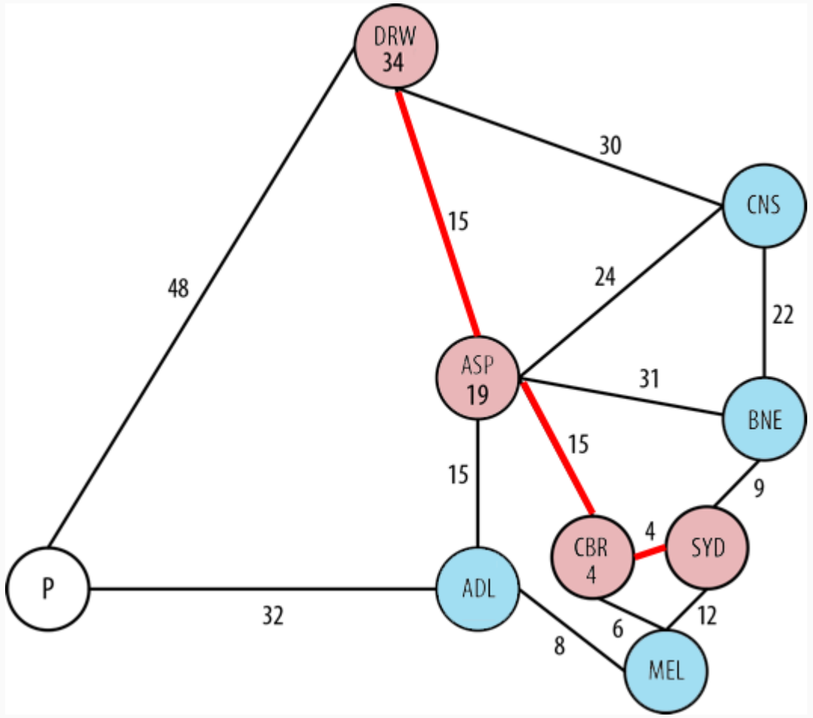
Se refiere a la exploración en un grafo.

Cada manera de explorar un grafo puede utilizar una estructura de apoyo diferente

El peor de los casos es $O(|V|^2)$

Para cada caso se utilizará memoria como ayuda. Recordando los vértices que ya han sido visitados

Se tiene varias aplicaciones: encontrar los vértices conexos, encontrar ciclos, caminos más cortos, etc

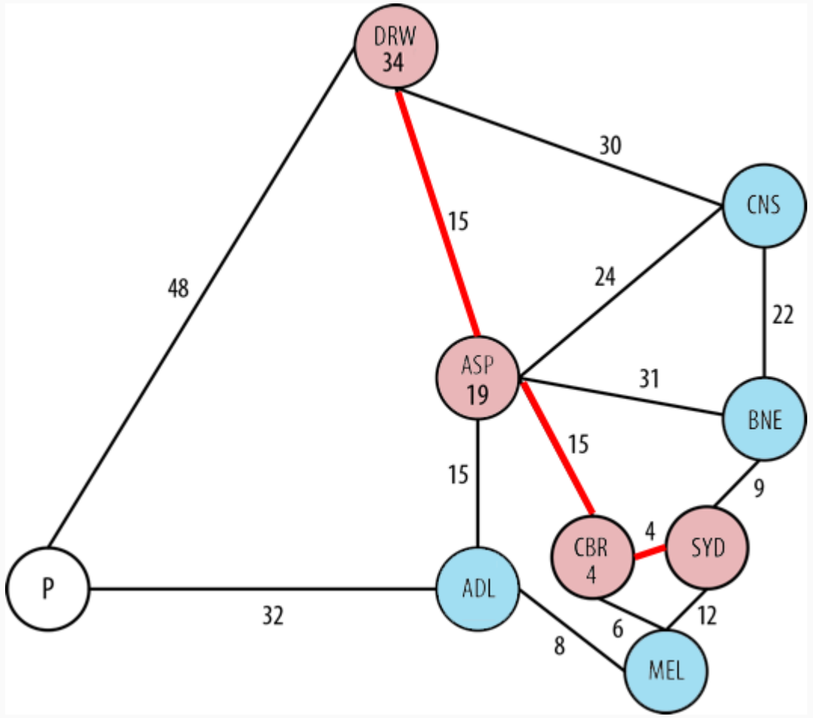


Búsqueda en grafos

La idea básica es ir expandiendo un árbol de estados al generar sucesores (estados) de un vértice ya visitado.

Cada estado es evaluado para saber si ya cumplimos con nuestro objetivo.

Se debe usar alguna heurística para generar sucesores, como el siguiente vértice al que se llega con menor peso, o uno aleatorio.

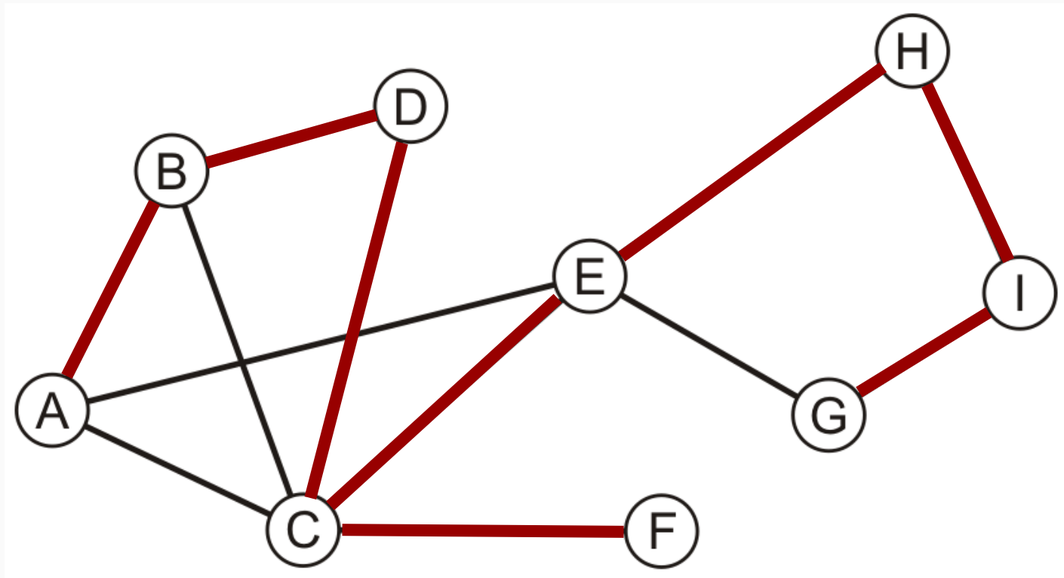


Búsqueda en profundidad (DFS)

Para explorar un grafo de esta manera vamos a necesitar una pila (stack)

DFS puede ser implementado de manera recursiva o iterativa

1. Elige cualquier vértice u, y agrégalo al stack marcándolo como visitado
   a. Desde u, si hay algún vértice v que no haya sido visitado agrégalo al stack y continúa la búsqueda desde v
   b. De otra forma, remueve el vértice u del stack y continúa desde el siguiente en la pila
2. Continúa el proceso hasta que no hayan más vértices que visitar

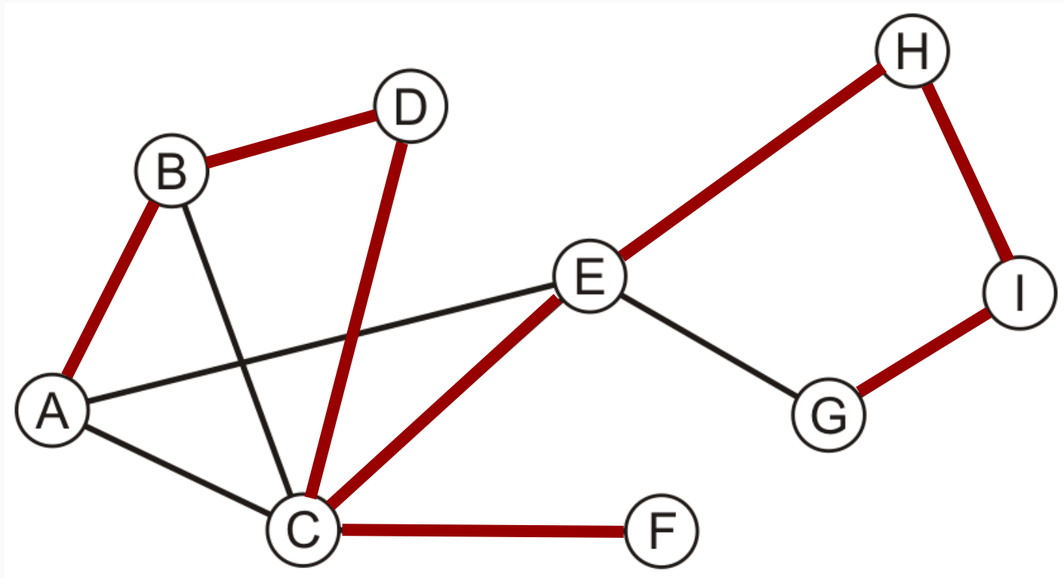


Búsqueda en profundidad (DFS)

```
function DFS(Graph G(V, E))
{
    for (i = 0; i < V.length; i++)
        visited[i] = false;

    for (i = 0; i < V.length; i++)
        if (!visited[i])
            DFS(G, i);
}

function DFS(Graph G(V, E), int i)
{
    visited[i] = true; //add to solution
    foreach(v[j] adjacent to v[i])
        if (!visited[j])
            DFS(G, j);
}
```



O(|V|+|E|)

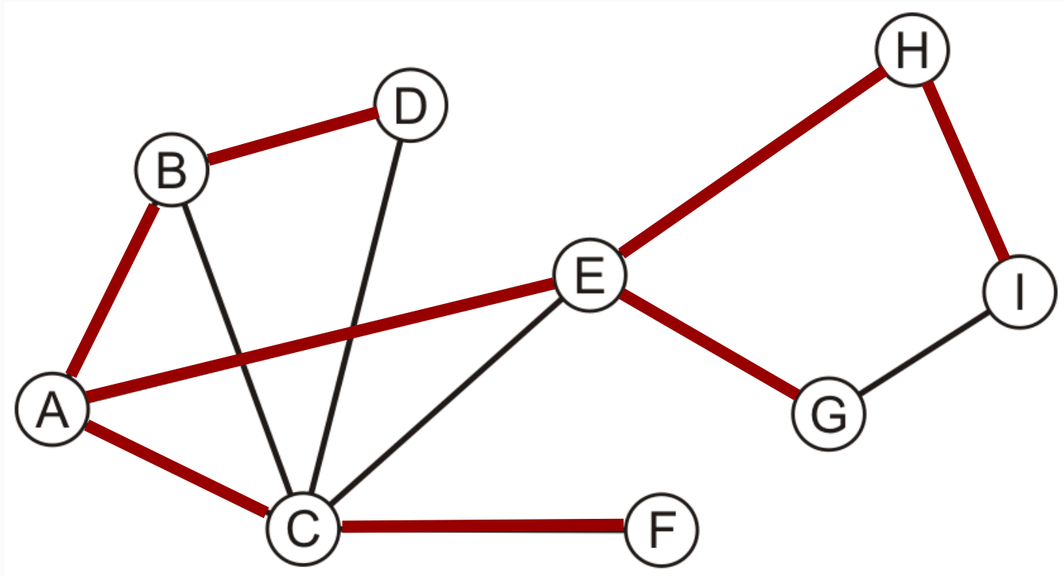
https://www.cs.usfca.edu/~galles/visualization/DFS.html
https://visualgo.net/es/dfsbfs

Búsqueda en amplitud (BFS)

Para explorar un grafo de esta manera vamos a necesitar una cola (queue)

El tamaño de la queue puede llegar a ser O(|V|)

1. Elige cualquier vértice u, y agrégalo a la queue marcándolo como visitado
   a. Pop el vértice v en la cima de la queue
   b. Agrega todos los vértices adyacentes a v que no hayan sido visitados
   c. Marca como visitados todos los vértices agregados a la queue
2. Continúa el proceso hasta que no hayan más vértices que visitar

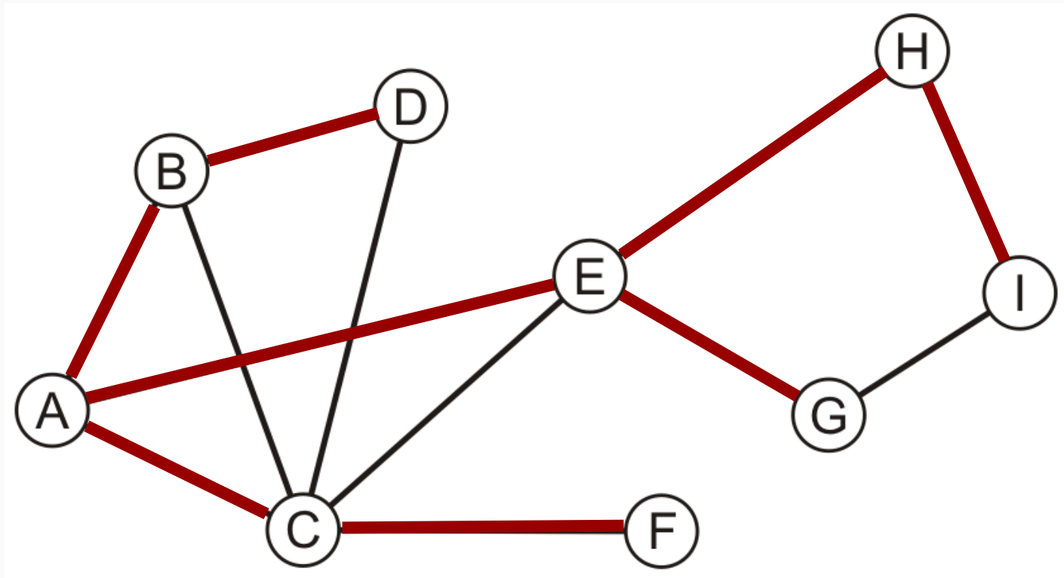


Búsqueda en amplitud (BFS)

```
function BFS(Graph G(V, E))
{
    for (i = 0; i < V.length; i++)
        visited[i] = false;
    for (i = 0; i < V.length; i++)
        if (!visited[i])
            BFS(G, i);
}

function  BFS(Grafo G(V, E), int i)
{
    Queue Q;
    visited[i] = true; // add to solution
    Q.add(i);
    while (!Q.empty()) {
        x = Q.extract();
        foreach(v[j] adjacent to v[x])
            if (!visited[j]) {
                visited[j] = true;  // add to solution
                Q.add(i);
            }

    }
}
```



O(|V|+|E|)

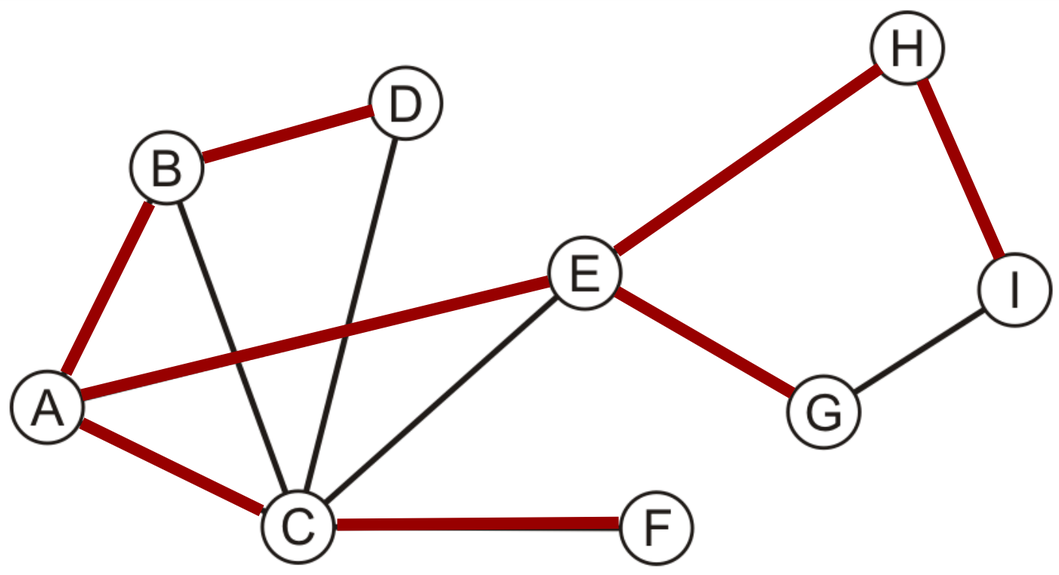
https://www.cs.usfca.edu/~galles/visualization/BFS.html
https://visualgo.net/es/dfsbfs

Búsqueda en amplitud (BFS)

¿Cómo harían para obtener el camino de un vértice inicial al final?

Se tendría que guardar (quizás en un map) el padre del nodo actual para poder iterar hacia atrás. Por ejemplo:

P(B) = A, P(E) = A, P(C) = A, P(D) = B, P(H) = E, P(G) = E, P(I) = H, y P(C) = F



Comparación

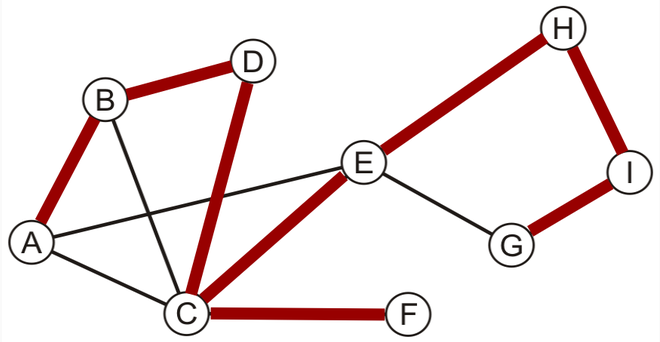
DFS es un algoritmo basado en vértices, mientras que BFS es basado en aristas. Utilizan estructuras de apoyo diferentes.

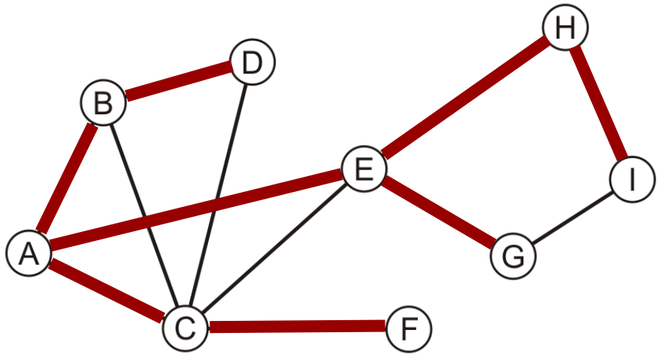
El uso de memoria en BFS es ineficiente

BFS puede encontrar el camino más corto en grafos con aristas del mismo peso, grafos bipartitos, etc

DFS se utiliza para obtener el ordenamiento topológico, saber si un grafo es fuertemente conexo

Cómo serán los árboles que generan?
DSF son árboles profundos y estrechos mientras que BFS produce árboles pequeños y amplios





Aplicaciones

1. Determinar la conectividad de un grafo.
2. Encontrar el camino de un vértice a todos los demás
3. Probar si un grafo es bipartito
4. Análisis de una red y sus relaciones
5. Detección de ciclos
6. Ordenamiento topológico
7. Detección de árboles