Recorridos de Gráficas

comp-420

Algoritmos de búsqueda en gráficas

- Una de las operaciones más frecuentes en una gráfica es la de visitar sus vértices uno por uno o en un orden deseado.

- Este procedimiento se llama búsqueda o recorrido de un grafo.

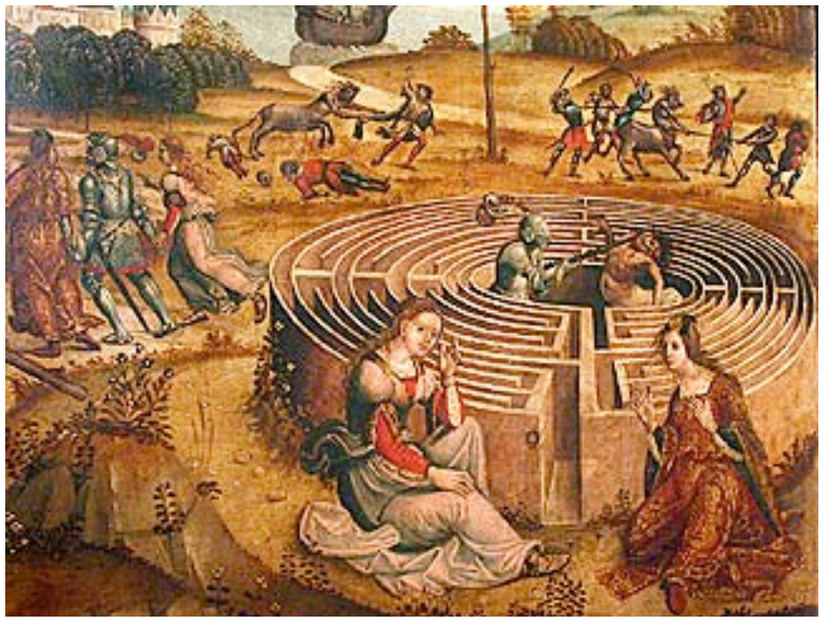
- Los algoritmos clásicos para búsqueda o recorridos en gráficas son:
  - depth-first search
  - breath-first search

- Por ahora solo vamos a "visitar el nodo" pero más tarde veremos acciones específicas a realizar en un nodo.

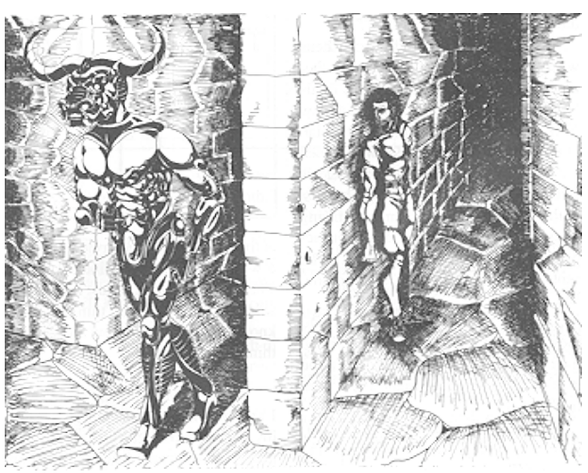
- Se puede elegir visitar el vértice la primera vez que se ve (preorden) o la última vez que se ve (postorden).

Explorando un laberinto

- Queremos encontrar la salida en un laberinto que consiste en pasajes conectados por intersecciones y también revisar todo el laberinto.
- Además del hilo podemos suponer luces, inicialmente apagadas y puertas, inicialmente cerradas en las intersecciones.



- Seguimos la exploración de Trémaux:
- Desenrollar un hilo detrás de nosotros.
- Marcar cada intersección visitada prendiendo una luz.
- Marcar cada pasaje visitado abriendo una puerta.

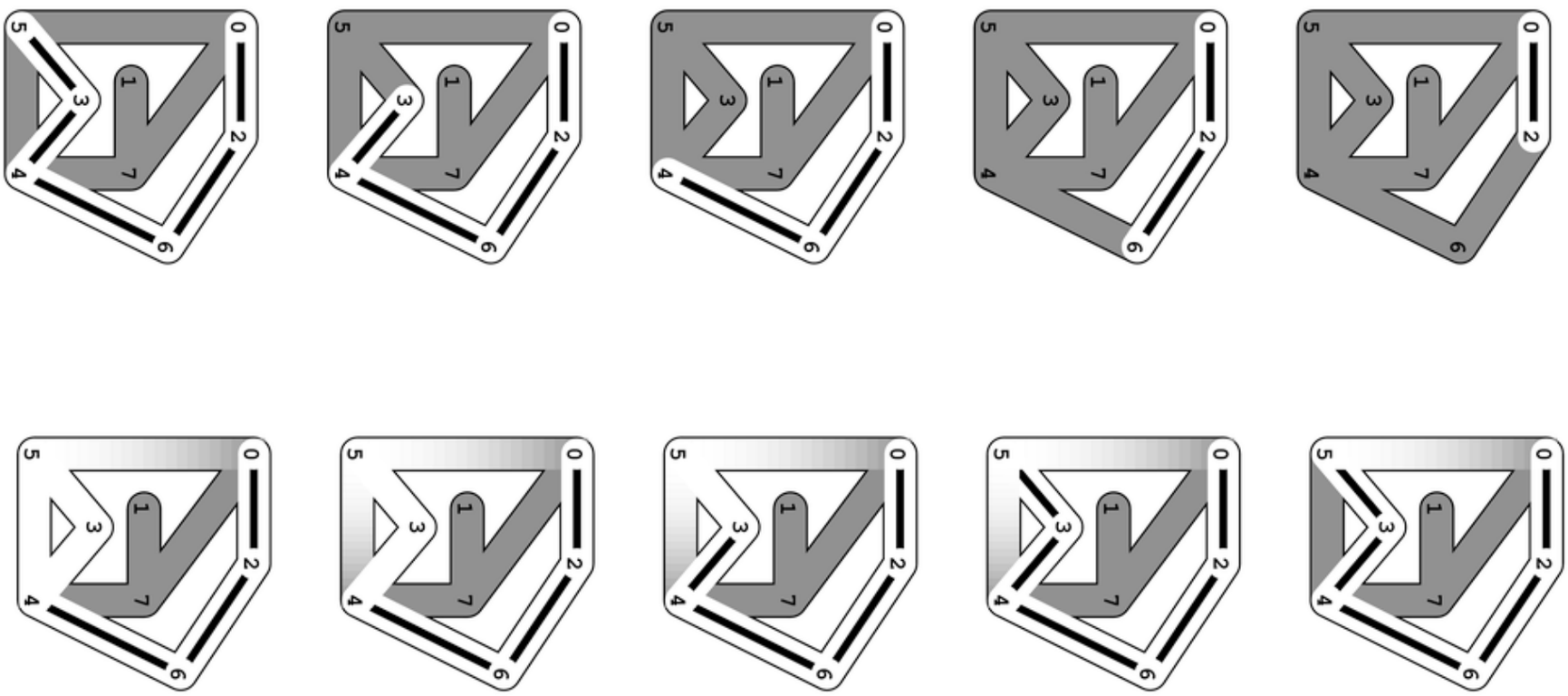


Explorando un laberinto

(I) Si no hay puertas cerradas en la intersección actual, ir al paso (III). Si no, abrir cualquier puerta cerrada para salir a un pasaje y dejar la puerta abierta.

(II) Si se puede ver una intersección al otro lado del pasaje que ya este encendido, probar otra puerta en la intersección actual (del paso 1). Si no (la intersección está apagada), seguir el pasaje hasta la intersección desenrollando el hilo, prender la luz y regresar al paso (I).

(III) Si todas las puertas en la intersección actual están abiertas, verificar si se está en el punto inicial. Si es el caso, parar. Si no, usar el hilo hasta llegar a la intersección que nos llevó allí buscando otra puerta

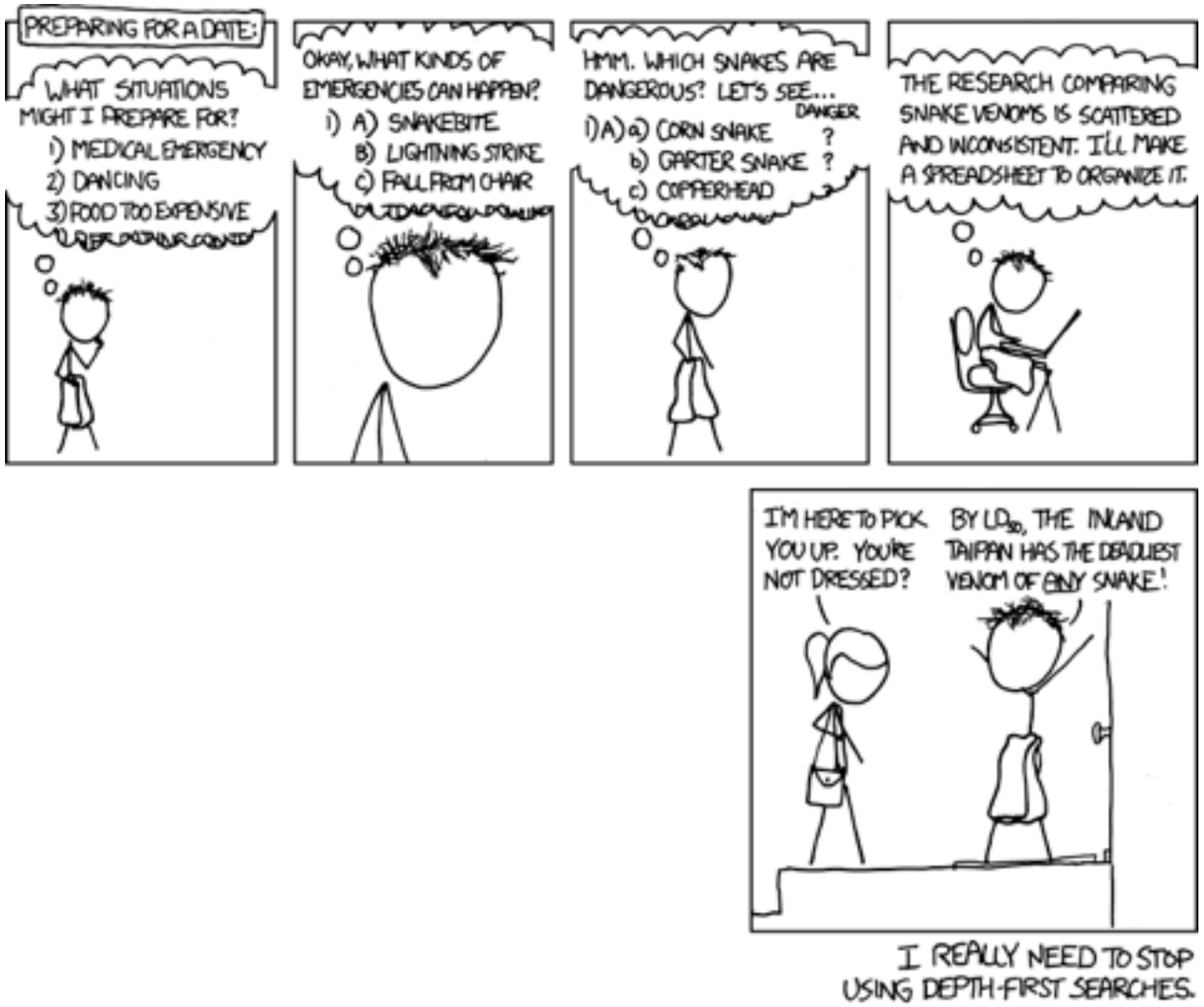

0
2
1
7
3
6
5
4

Explorando un laberinto

- Al usar el algoritmo de exploración de Trémaux, encendemos todas las luces, abrimos todas las puertas y regresamos al punto inicial.
- Hay 4 posibles situaciones:
  - El pasaje está apagado, entonces lo seguimos.
  - El pasaje es el que usamos para entrar (tiene nuestro hilo), lo usamos para salir.
  - La puerta del otro lado esta cerrada (pero la intersección está prendida), no recorremos el pasaje.
  - La puerta al otro lado del pasaje está abierta y la intersección prendida, nos saltamos el pasaje.

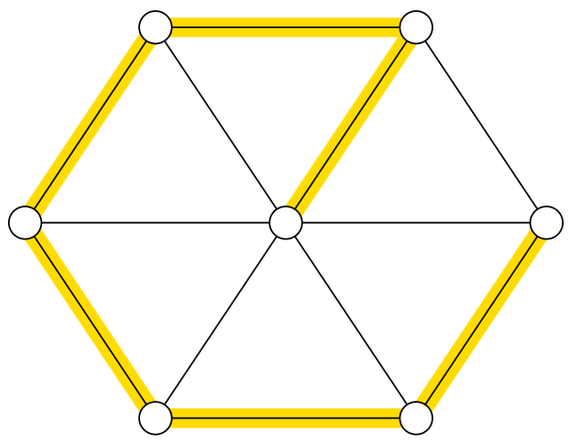
Depth-first search

- La técnica de exploración de Trémaux nos lleva a la función recursiva clásica para recorrer gráficas:
  - para visitar un vértice, lo marcamos como visitado y (recursivamente) visitamos todos los vértices adyacentes a este que no han sido marcados. (DFS)



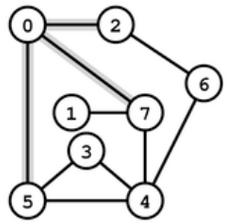
Depth-first search

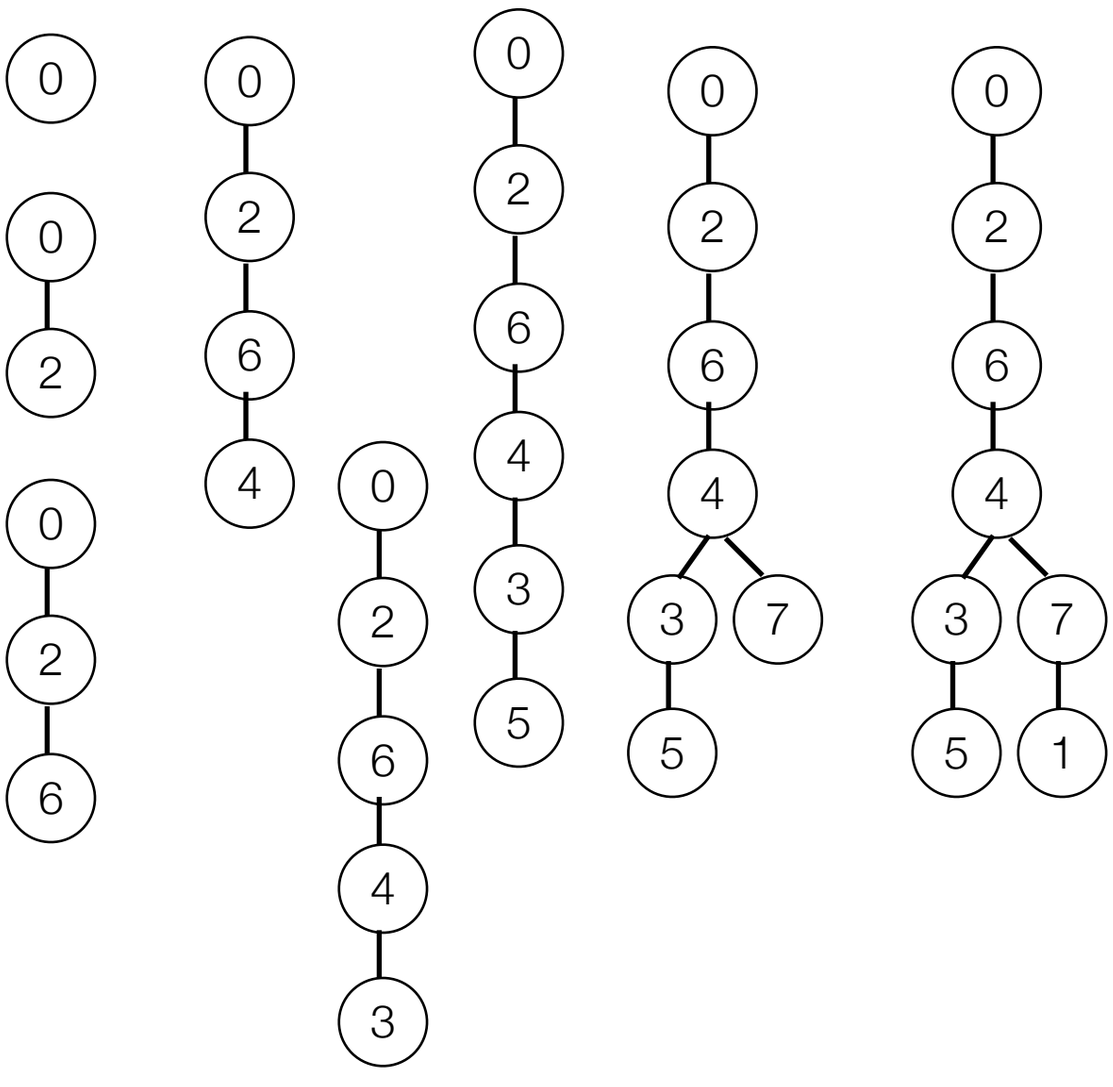
- En depth-first (profundidad primero) search:
  - exploramos en un solo camino en la gráfica hasta lo más lejos que podamos llegar (no se encuentren más vértices).
  - regresamos hasta un punto donde haya vértices que explorar y continuamos (como explorar un laberinto).
- Ejemplo de depth-first search empezando en el centro de la gráfica:



Depth-first search

- Tiempo de ejecución:
  - O(E) ya que cada arista es examinada a lo más dos veces.





Depth-first search

- El algoritmo puede escribirse de manera recursiva o iterativa.
- Ambas versiones toman un vértice fuente (source) *s*.

RECURSIVE_DFS(v):

1. if ***v*** is unmarked
2. mark ***v***
3. for each edge ***(v,w)***
4. RECURSIVE_DFS(***w***)

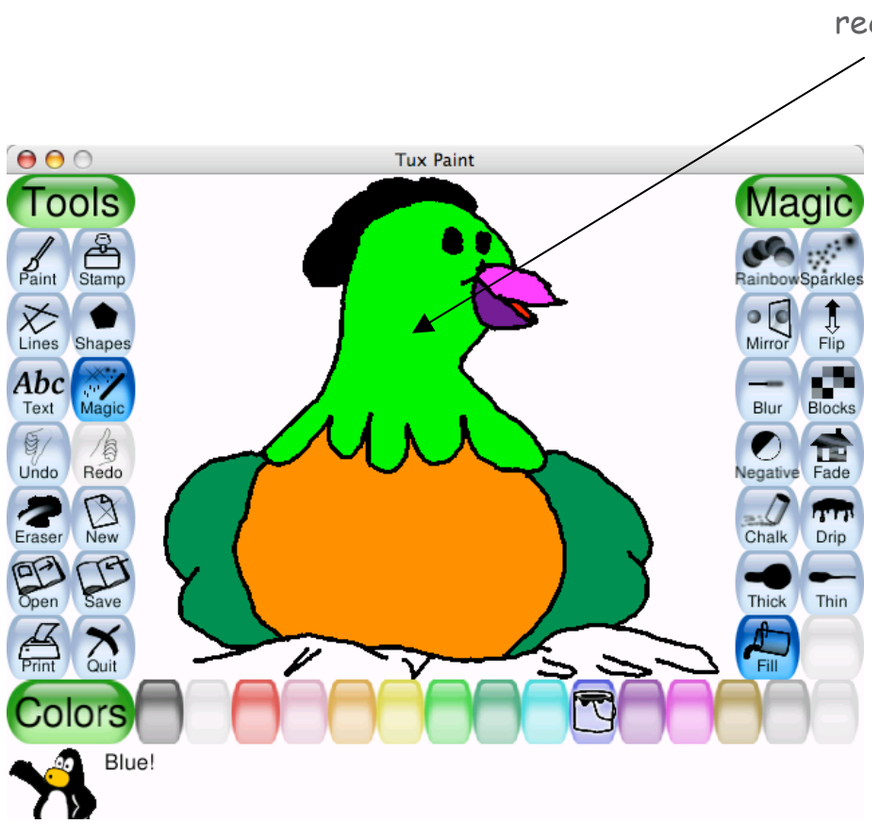
ITERATIVE_DFS(s):

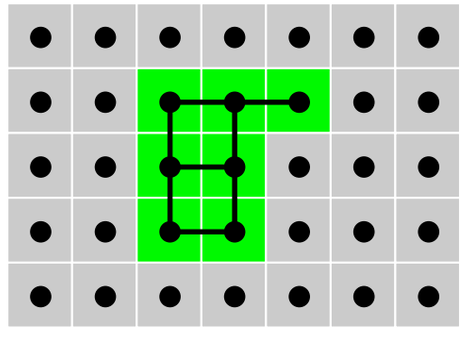
1. ***PUSH(s)***
2. while stack not empty
2. ***v*** ← ***POP***
3. if ***v*** is unmarked
4. mark ***v***
5. for each edge ***(v,w)***
6. ***PUSH(w)***

- Es exactamente el mismo algoritmo con la única diferencia que en la versión iterativa se puede "ver" la pila de la recursión.

Llenado por inundación

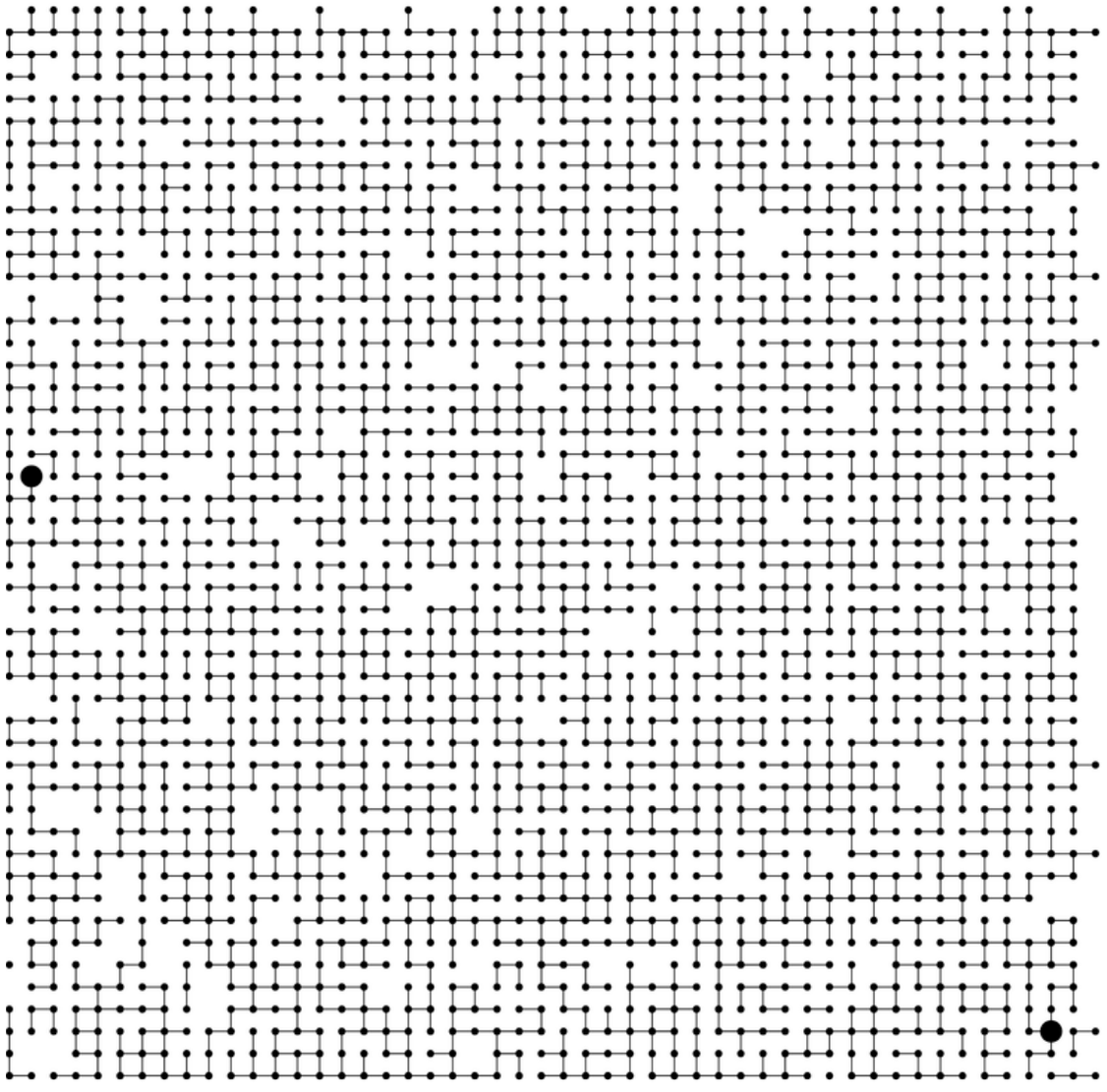
- Dado un pixel de color verde limón en la imagen, cambia el color de todos sus vecinos del mismo color a azul.
- Vértice: pixel
- Arista: pixeles vecinos del mismo color.
- Blob: todos los pixeles alcanzables a partir del pixel inicial.





Encontrar un camino

- Hay un camino de s a t? si lo hay, encuentralo.



Recorridos genéricos en gráficas

- DFS es una (tal vez la más común) instancia de la familia general de algoritmos de recorrido de gráficas.
- El algoritmo de recorrido genérico almacena un conjunto de aristas candidatas en alguna estructura de datos que llamaremos "bolsa".
- Las únicas propiedades importantes de la bolsa es que se puedan poner cosas dentro (como un contenedor en C++) y que se puedan sacar cuando sea necesario.

```
TRAVERSE(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag        (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge (v, w)        (†)
              put (v, w) into the bag      (⋆⋆)
```

Recorridos genéricos en gráficas

```
TRAVERSE(s):
    put (∅, s) in bag
    while the bag is not empty
        take (p, v) from the bag          (⋆)
        if v is unmarked
            mark v
            parent(v) ← p
            for each edge (v, w)          (†)
                put (v, w) into the bag   (⋆⋆)
```

- Notese que estamos guardando las aristas en la bolsa en lugar de los vértices. Esto es porque queremos recordar, cuando visitemos un vértice **v** por primera vez, cuál vértice previamente visitado **p** puso a **v** en la bolsa.
- Llamamos al vértice **p** padre de **v**.

Recorridos genéricos en gráficas

```
TRAVERSE(s):
    put (∅, s) in bag
    while the bag is not empty
        take (p, v) from the bag          (⋆)
        if v is unmarked
            mark v
            parent(v) ← p
            for each edge (v, w)          (†)
                put (v, w) into the bag   (⋆⋆)
```

- TRAVERSE(s) marca cada vértice en una gráfica conectada, exactamente una vez, y el conjunto de aristas *(v,parent(v))* con *parent(v)* $\neq \varnothing$ forma un árbol generador (*spanning tree*) del grafo.
- Cada vértice no se marca más de una vez.
- El conjunto de aristas forman un árbol generador.

```
TRAVERSE(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag        (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge (v, w)        (†)
              put (v, w) into the bag      (⋆⋆)
```

- El tiempo exacto de ejecución de un algoritmo de recorrido de gráficas depende de la representación de la gráfica y de la estructura usada como bolsa. Sin embargo se pueden hacer observaciones generales:

- Ya que cada vértice se visita a lo más una vez, entonces el cíclo $(\dagger)$ se ejecuta a lo más ... V veces .

- Cada arista se pone en la bolsa exactamente dos veces, una como (v,u) y otra como (u,v), entonces la línea $(\star\star)$ se ejecuta a lo más ... $2E$ veces .

- Finalmente, como no podemos sacar más cosas de la bolsa de las que metimos , la línea $(\star)$ se ejecuta a lo más ... $2E + 1$ veces .

Ejemplos de recorridos en gráficas

- Suponiendo una representación con listas de adyacencia, si implementamos la bolsa con una pila (stack), ¿qué algoritmo tenemos?
  - depth-first search.
- Cada ejecución de $(\star)$ o $(\star\ \star)$ toma
  - tiempo constante.
- El tiempo de ejecución total es $O(V+E)$.
- Ya que la gráfica está conectada, $V \leq E+1$, por lo que podemos simplificar el tiempo de ejecución a $O(E)$.
- El árbol generador producido se llama depth-first spanning tree.
- La forma de este árbol depende del orden en que se recorren los nodos adyacentes pero en general serán alargados.

Ejemplos de recorridos en gráficas

- Suponiendo una representación con listas de adyacencia, si implementamos la bolsa con una cola (queue), ¿qué algoritmo tenemos?
  - breath-first search.
- Cada ejecución de $(\star)$ o $(\star\star)$ toma
  - tiempo constante.
- El tiempo de ejecución total es $O(V+E)$.
- Ya que la gráfica está conectada, $V \leq E+1$, por lo que podemos simplificar el tiempo de ejecución a $O(E)$.
- El árbol generador producido contiene los caminos más cortos desde el inicio hasta el vértice actual.
- La forma de este árbol depende del orden en que se recorren los nodos adyacentes pero en general serán anchos y cortos.

Recorridos en gráficas desconectadas

- Si la gráfica está desconectado, entonces TRAVERSE(s) solo visita los nodos en el componente conectado del vértice inicial s.
- Si queremos visitar todos los nodos podemos utilizar el siguiente algoritmo, que calcula el bósque generador de la gráfica.

```
TraverseAll(s):
    for all vertices v
        if v is unmarked
            Traverse(v)
```