DFS

Depth First Search

Joël Mathys

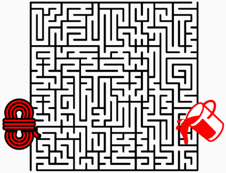
2017-10-14

Swiss Olympiad in Informatics

Introduction

DFS Introduction

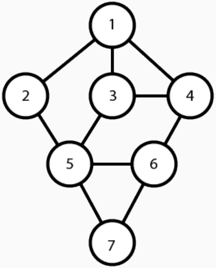
- Depth First Search
- graph traversal
- applications
    - graph coloring
    - component counting



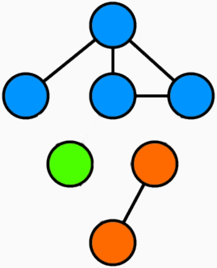
Repetition

Repetition

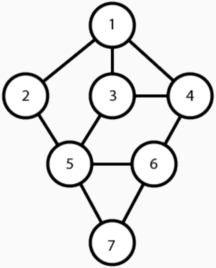
graph traversal



components

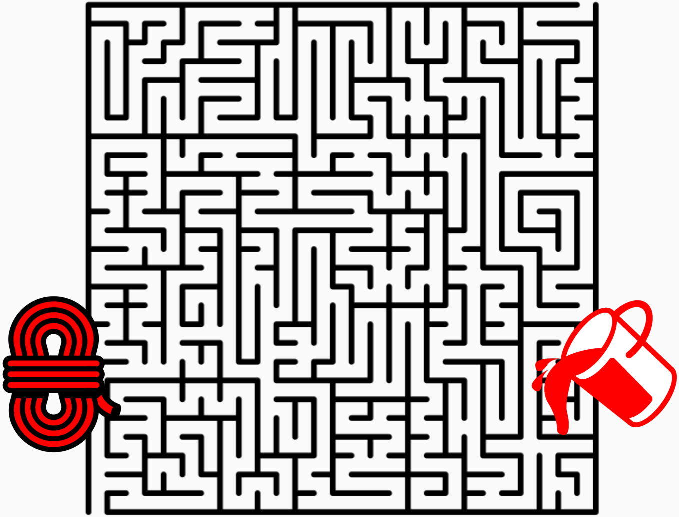


adjacency list

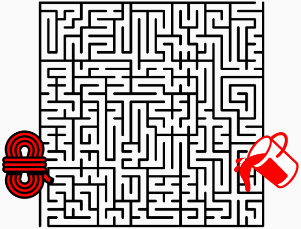


DFS Algorithm

Trapped in the labyrinth



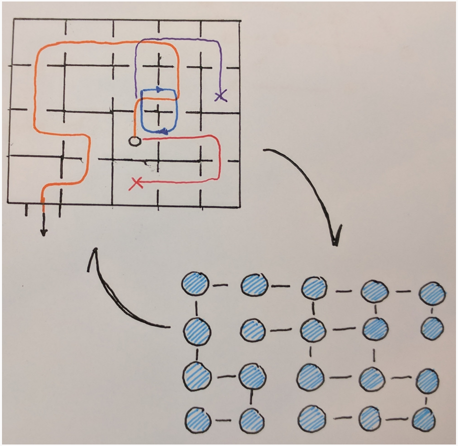
Escape strategy



- fix red string
    - know way back
- draw red points
    - know where we've been
- visit neighbors
- return if dead end or marked

⇒ take new path - repeat unless nothing new

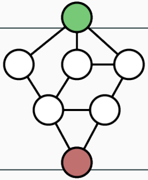
labyrinth example



DFS Implementation

- determine if end reachable
- setup datastructures
- visited ⇔ red bucket
- dfs stack ⇔ red string

```
visited = [False] * n
def dfs(current):
    ...
```



DFS Implementation

- enter new node / room
- check if we were already here
- return ⇔ follow string

```
...
def dfs(current):
    if visited[current]:
        return
    visited[current] = True
```

DFS Implementation

- visit the neighbors

```
for neighbor in graph[current]:
    dfs(neighbor)
```

DFS Implementation

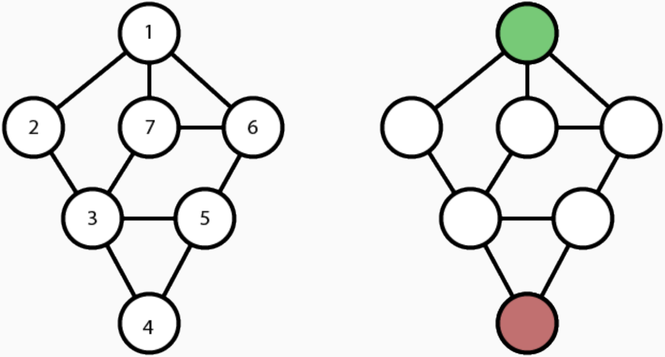
- look for exit

```
...
dfs(start)
if visited[end]:
    print("can␣reach␣it!")
...
```

DFS Implementation

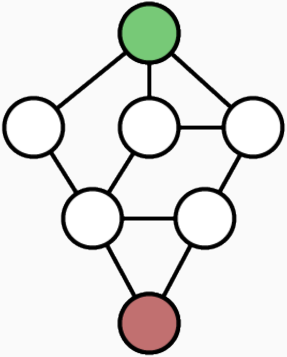
```
visited = [False] * n  # setup datastructure

def dfs(current):

    if visited[current]  # been there?
        return
    visited[current] = True #mark with red spot

    for neighbor in graph[current]:  # check
    dfs(neighbor)  # neighbors

# end of dfs function
dfs(start)
if visited[end]:  # check target
    print("can␣reach␣it!")
```

DFS Example



DFS Analysis

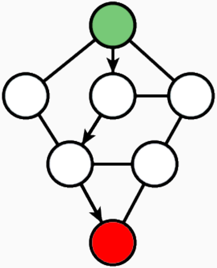
- memory:
    - graph, visited Array
    - $\Rightarrow O(n)$
- runtime:
    - visit each node
    - walk over each edge
    - $\Rightarrow O(n+m)$



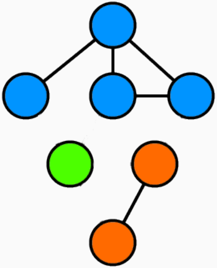
DFS Applications

Applications

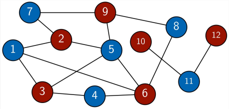
paths



components



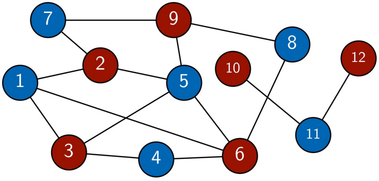
coloring



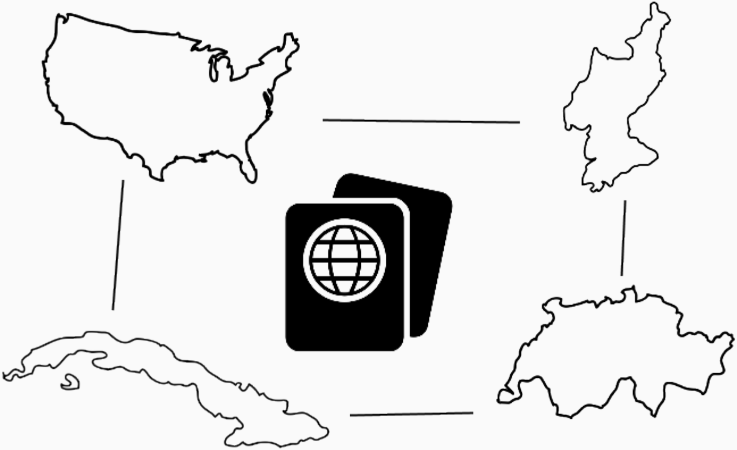
Graph coloring

Coloring

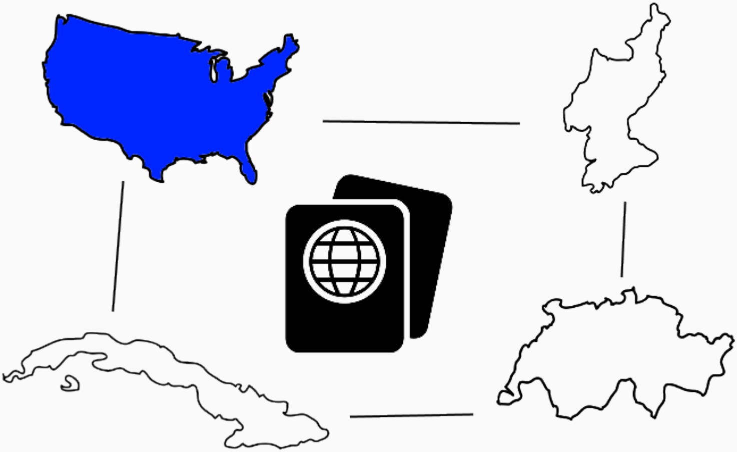
- coloring is hard!
- special case for two
- place mice at a table
- prelim.soi.ch
- passports task



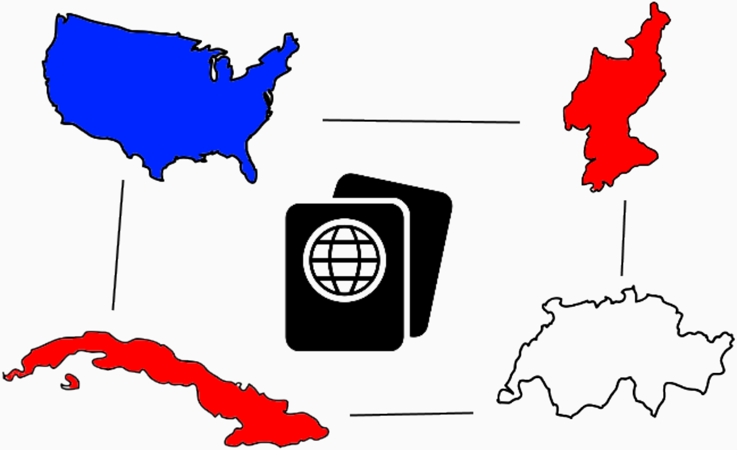
Applications



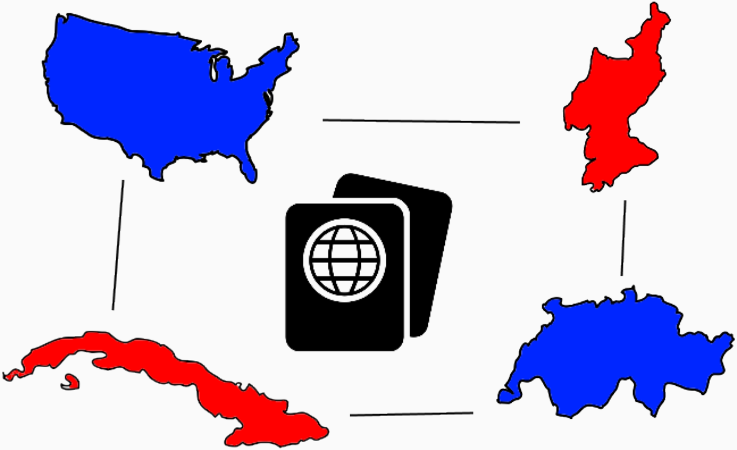
Applications



Applications



Applications



DFS Implementation

- setup datastructures
- visited[]
- color[] (blue=1, red = 2)

```
visited = [False] * n
color  = [0] * n
def dfs(current, col):
    ...
```

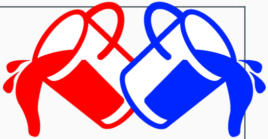
DFS Implementation

- enter new node
- check if we were already here

```
...
def dfs(current, col):
    if visited[current]:
        return True
    visited[current] = True
```

DFS Implementation

- paint node blue or red
- determine color for neighbors



```
...
    visited[current] = True
    color[current] = col
    ncol = 1
    if col == 1:
        ncol = 2
    else:
        ncol = 1
 ...
```

DFS Implementation

- visit the neighbors
- check for conflict!

```
...
for neighbor in graph[current]:
    if color[neighbor] == col:
        return False
    if not dfs(neighbor, ncol):
        return False
return True
```

DFS Implementation

- color all nodes

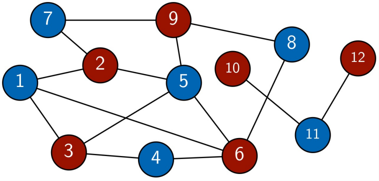
```
...
for node in range(n):
    if not dfs(graph, node, 1):
        print("no␣coloring␣possible")
...
```

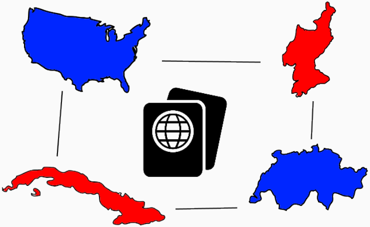
DFS Implementation

```python
visited = [False] * n
color  = [0] * n

def dfs(graph, current, col):
    if visited[current]:
        return True

    visited[current] = True
    color[current] = col
    ncol = 1
    if col == 1:
        ncol = 2
    else:
        ncol = 1
    for neighbor in graph[current]:
        if color[neighbor] == col:
            return False
        if not dfs(graph, neighbor, ncol):
            return False
    return True
# end of dfs function
for node in range(n):
    if dfs(graph, node, 1) == False:
        print("no␣coloring␣possible")
```

Summary

- continue walking as long as you see new nodes
  $\Rightarrow$ depth first!
- standard way to explore graph
- many applications
  coloring, components, ...





Questions

let’s be amazeing