Welch-Powell Algorithms

Sejin Chun

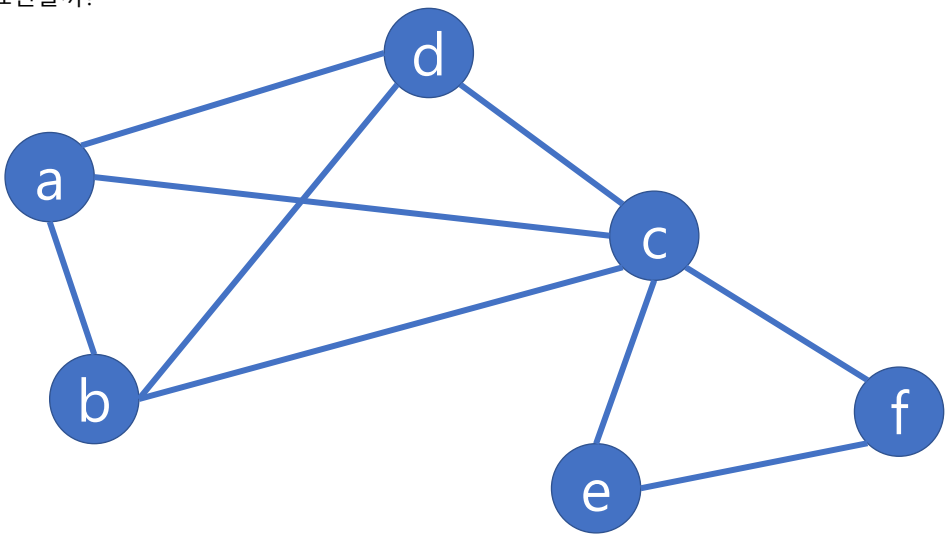
1

알고리즘

- ① 그래프 G의 정점의 차수가 내림차순(descending order)이 되게 배열한다(이 배열은 차수가 같은 정점이 여러 개 있을 수 있으므로 몇 가지 다른 순서가 존재할 수 있다).
- ② 배열의 첫 번째 정점은 첫 번째 색으로 착색하고 계속해서 배열의 순서대로 이미 착색된 정점과 인접하지 않은 정점을 모두 같은 색으로 착색한다.
- ③ 배열에서 먼저 나타나는 착색되지 않은 정점을 두 번째 색으로 착색하고 계속해서 배열의 순서대로 지금 착색하고 있는 색으로 이미 착색된 정점과 인접하지 않은 정점을 모두 착색한다.
- ④ 계속해서 위의 과정을 그래프의 모든 정점이 착색될 때까지 반복한다.

2

어떻게 코드로 표현할까?



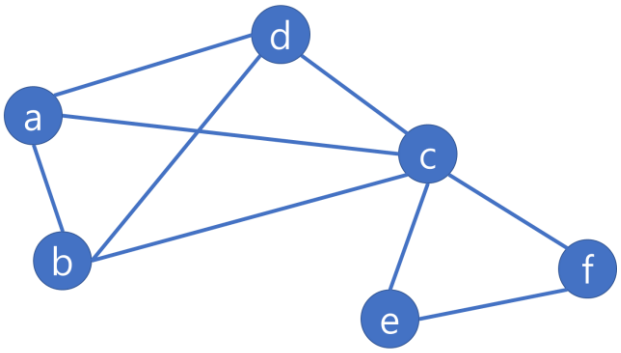
3

0. 그래프는 dictionary와 list로 조합

- graph = {}
- 노드는 key로 표현, 이웃노드는 리스트로 표현
  - 'a' : ['b', 'c', 'd']
  - 'b' : ['c', 'f'
  - (Un-)Directed 그래프 모두 표현가능

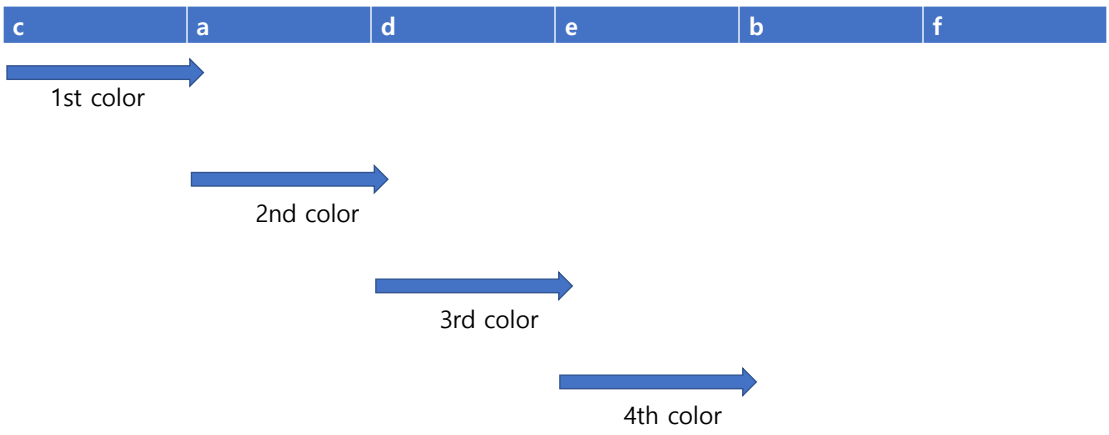
4

Try it



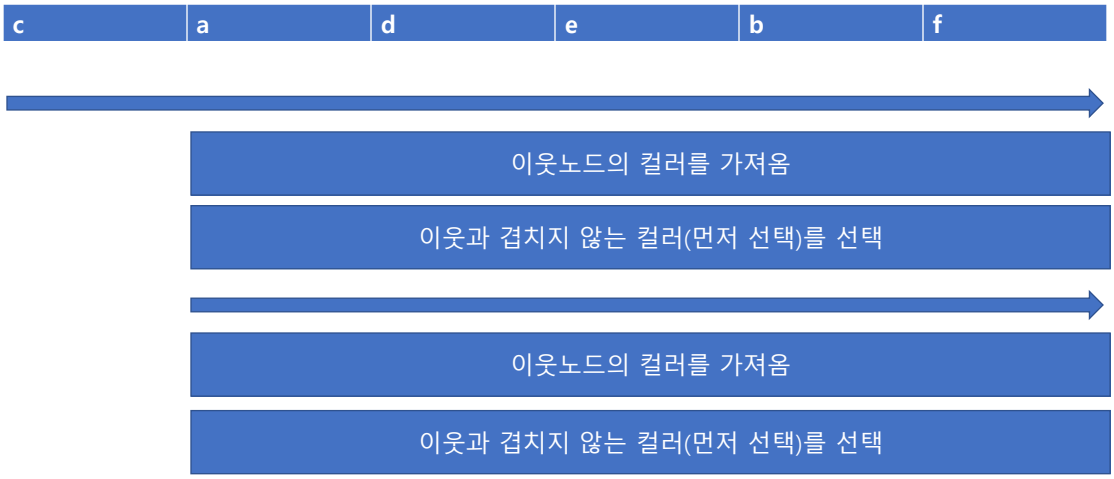
5

Iteration



6

Iteration



7

#1. 차수로 정렬하기

- sorted(**data**, key=lambda x: **func**, reverse=True/**False**)

8

#2. color_map에서 이웃의 컬러 확인

• color_map = {} // 'c' : 0, 'a' : 1

• for each neighbor_node,
    if neighbor_node in color_map:
        color = color_map[neighbor_node]
        available_colors[color] = False # 이웃인 노드들은 False

9

#3. 색칠 가능

• for each neighbor_node,
    if neighbor_node in color_map:
        color = color_map[neighbor_node]
        available_colors[color] = False # 이웃인 노드들은 False

    for color, is_colorable in enumerate (available_colors):
        if is_colorable: # 색칠 가능할 때
                color_map[node] = color
                break

10

#4. color_map 반환

- return color_map

11

Full code

```
def color_nodes(graph):
  nodes = sorted(list(graph.keys()), key=lambda x: len(graph[x]), reverse=True)
  color_map = {}

  for node in nodes:
    available_colors = [True] * len(nodes)
    for neighbor in graph[node]:
      if neighbor in color_map:
        color = color_map[neighbor]
        available_colors[color] = False
    for color, is_colorable in enumerate(available_colors):
      if is_colorable:
        color_map[node] = color
        break

  return color_map
```

12

Optimize some codes

```
def color_nodes(graph):
  nodes = sorted(list(graph.keys()), key=lambda x: len(graph[x]), reverse=True)
  color_map = {}

  for node in nodes:
    available_colors = [True] * len(nodes)
    for neighbor in graph[node]:
      if neighbor in color_map:
        color = color_map[neighbor]
        available_colors[color] = False
    for color, is_colorable in enumerate(available_colors):
      if is_colorable:
        color_map[node] = color
        break

  return color_map
```

Generator expression using set

13

Optimize some codes

```
def color_nodes(graph):
  nodes = sorted(list(graph.keys()), key=lambda x: len(graph[x]), reverse=True)
  color_map = {}

  for node in nodes:
    available_colors = [True] * len(nodes)
    for neighbor in graph[node]:
      if neighbor in color_map:
        color = color_map[neighbor]
        available_colors[color] = False
    for color, is_colorable in enumerate(available_colors):
      if is_colorable:
        color_map[node] = color
        break

  return color_map
```

next (generator expression with a condition)

14