Algorithmic Strategies 2024/25

Week 8 – Graph Algorithms

· U C ·

Universidade de Coimbra

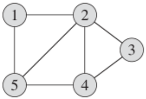
Outline

1. Introduction
2. Graph Representation
3. Breath-First Search
4. Depth-First Search
5. Topological Ordering

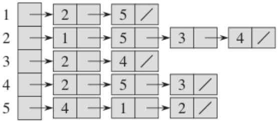
Reading about graph algorithms

- J. Erickson, Algorithms, Chapters 5 – 9
- Cormen et al., Introduction to Algorithms, Chapters 22 – 25

Undirected graph representation



Graph



Adjacency list

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

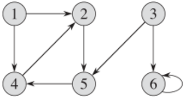
Adjacency matrix

Source: T. Cormen et al., Introduction to Algorithms, 2001

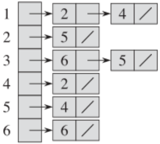
- Adjacency matrix is faster if you want to know whether two vertices are connected or not
- Adjacency list is faster if you want to visit all neighbors

Graph representation

Directed graph representation



Graph



Adjacency list

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Adjacency matrix

Source: T. Cormen et al., Introduction to Algorithms, 2001

- Adjacency matrix is faster if you want to know whether two vertices are connected or not
- Adjacency list is faster if you want to visit all neighbors

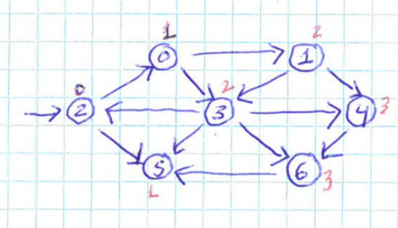
Breath-First Search

Function $BFS(G, v)$

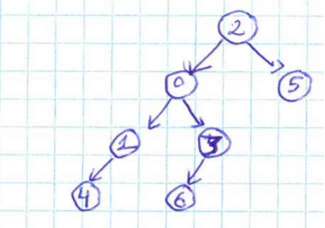
 Q is an empty queue
 mark v
 enqueue(v, Q)
 while $Q \neq \emptyset$ **do**
  $t =$ dequeue(Q)
  for each arc $(t, u) \in A$ **do**
   if u is not marked **then**
    mark u
    enqueue(u, Q)

- BFS has $O(|V| + |A|)$ time complexity
- It finds the shortest path (number of links) between two vertices

Breath-First Search



Graph



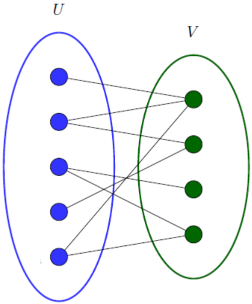
BFS tree

- The red number at each vertex indicates the shortest distance from vertex 2
- The vertices at the same level of the BFS tree are at the same shortest distance from vertex 2

Example: Shortest distance

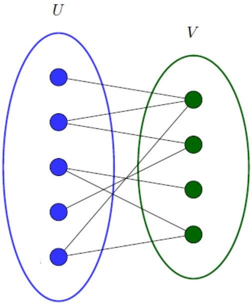
Function $BFS(G, v)$
  Q is an empty queue
  $visited[v] = true$
  $dist[v] = 0$
  enqueue(v, Q)
  while $Q \neq \emptyset$ **do**
    $t =$ dequeue(Q)
    for each arc $(t, u) \in A$ **do**
      if $visited[u] = false$ **then**
        $visited[u] = true$
        $dist[u] = dist[t] + 1$
        enqueue(u, Q)
  return $dist$

Example: Bipartite Matching

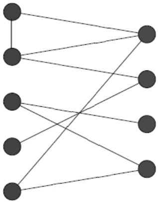


A bipartite graph is a graph whose vertices can be split into two disjoint sets, U and V, such that every edge connects only vertices from the two sets.

Example: Bipartite Matching



Bipartite graph



Non-bipartite graph

Example: Bipartite Matching

Function $BFS(G, v)$
  Q is an empty queue
  $color(v) = 1$
  enqueue(v, Q)
  while $Q \neq \emptyset$ **do**
    $t =$ dequeue(Q)
    for each edge $\{t, u\} \in E$ **do**
      if u has no color **then**
        $color(u) = 1 - color(t)$
        enqueue(u, Q)
      else if $color(u) = color(t)$ **then**
        return False
  return True

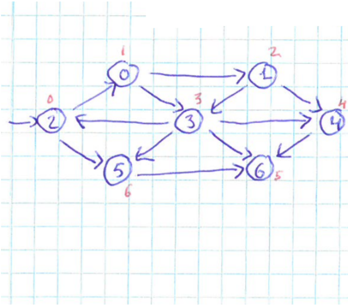
- Each vertex is colored with 0 or 1.

Depth-First Search

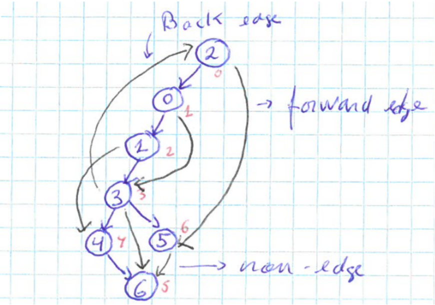
Function $DFS(G, v)$
    mark v
    for each arc $(v, u) \in A$ **do**
        if u is not marked **then**
            $DFS(G, u)$

- DFS has $O(|V| + |A|)$ time complexity
- It is the basis of many graph algorithms

Depth-First Search



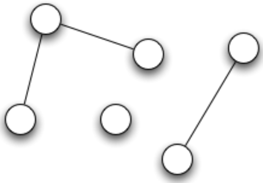
Graph



DFS tree

- The red number at each vertex indicates the visiting order

Example: Find if graph G is connected

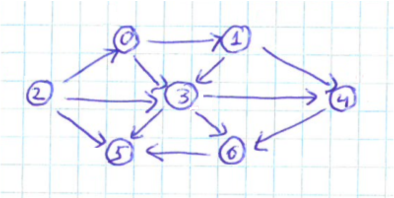


Example: Find if graph G is connected

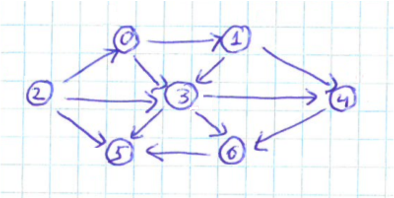
Function $DFS(G, v)$
  mark v
  for each arc $(v, u) \in A$ **do**
    if u is not marked **then**
      $DFS(G, u)$

Function $Connected(G, v)$
  $DFS(G, v)$
  for each vertex $u \in V$ **do**
    if u not marked **then**
      return False
  return True

Topological sorting in acyclic directed graphs



Topological sorting in acyclic directed graphs



- Sequence: 2, 0, 1, 3, 4, 6, 5

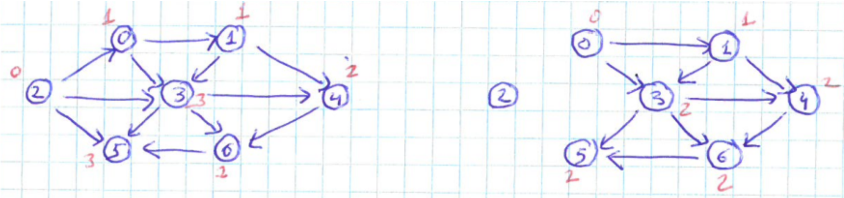
Topological sorting in acyclic directed graphs

1. Compute in-degree of all vertices
2. While exist vertices with null in-degree
   2.1 Select vertex v with null in-degree
   2.2 Process vertex v
   2.3 Remove vertex v and outgoing arcs
   2.4 Update in-degree of vertices adjacent to v

- It has $O(|V| + |A|)$ time complexity
- For every vertex (u, v), u comes before v in the ordering

Graph Traversal

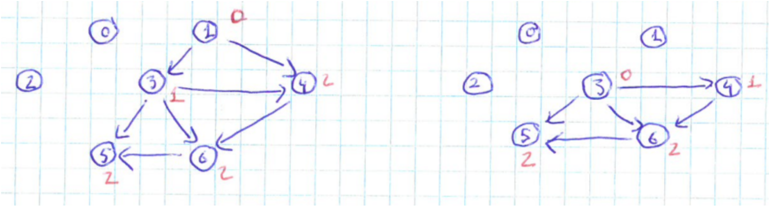
Topological sorting in acyclic directed graphs



iteration 1   iteration 2

- The red number at each vertex indicates its in-degree
- Sequence: 2, 0,. . .

Topological sorting in acyclic directed graphs

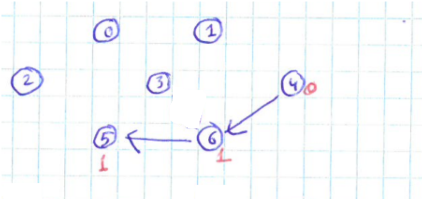


iteration 3 iteration 4

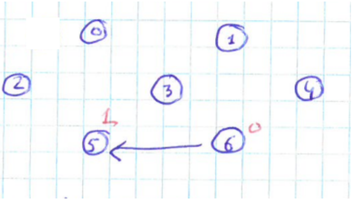
- The red number at each vertex indicates its in-degree
- Sequence: 2, 0, 1, 3,. . .

Graph Traversal

Topological sorting in acyclic directed graphs



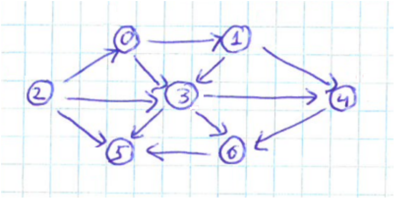
iteration 5



iteration 6

- The red number at each vertex indicates its in-degree
- Sequence: 2, 0, 1, 3, 4, 6, . . .

Topological sorting in acyclic directed graphs



- Sequence: 2, 0, 1, 3, 4, 6, 5

Topological sorting in acyclic directed graphs

Function $TS(G)$

$Q = \emptyset, S = \emptyset$
for each arc $(u, v) \in A$ **do**
  $indegree[v] = indegree[v] + 1$
for each vertex $v \in V$ **do**
  if $indegree[v] = 0$ **then**
    enqueue(v, Q)
while $Q \neq \emptyset$ **do**
  $u =$ dequeue(Q)
  enqueue(u, S)
  for each arc $(u, v) \in A$ **do**
    $indegree[v] = indegree[v] - 1$
    if $indegree[v] = 0$ **then**
      enqueue(v, Q)
return S

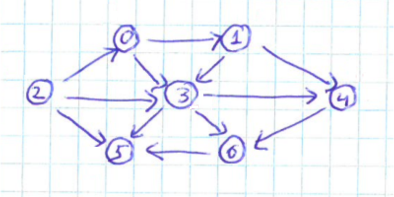
Topological sorting with DFS

Function $DFS(G, v)$

   mark v
   for each arc $(v, u) \in A$ **do**
      if u is not marked **then**
         $DFS(G, u)$
   push(v, S)

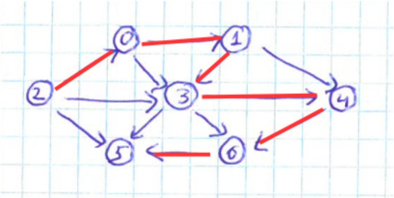
- S contains the sequence of vertices topologically sorted

Topological sorting with DFS



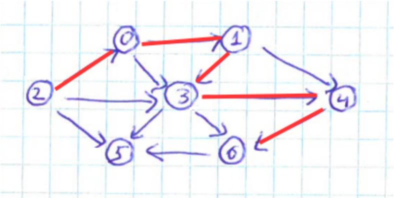
- Sequence:

Topological sorting with DFS



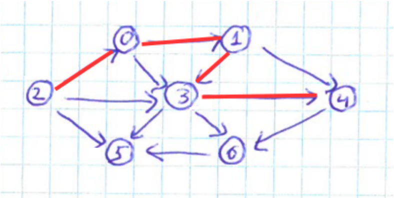
- Sequence: . . . , 5

Topological sorting with DFS



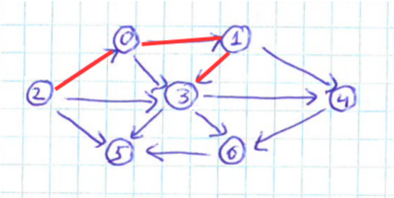
- Sequence: . . . , 6, 5

Topological sorting with DFS



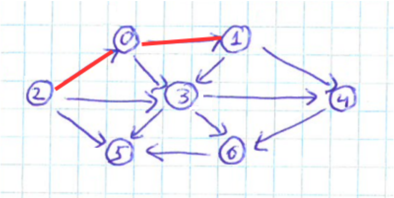
- Sequence: . . . , 4, 6, 5

Topological sorting with DFS



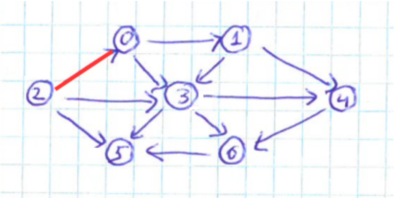
- Sequence: . . . , 3, 4, 6, 5

Topological sorting with DFS



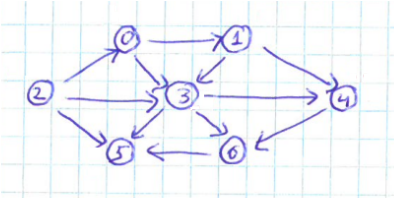
- Sequence: . . . , 1, 3, 4, 6, 5

Topological sorting with DFS



- Sequence: . . . , 0, 1, 3, 4, 6, 5

Topological sorting with DFS



- Sequence: 2, 0, 1, 3, 4, 6, 5