



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Graph representation and elementary algorithms

Data Structures and Algorithms

Tran Ngoc Bao Duy

*Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM*

① Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

② Searching a graph

Breadth-first search

Depth-first search

③ Topological sorting



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

GRAPH REPRESENTATION



Two standard ways to represent a graph $G = (V, E)$ apply to both directed and undirected graphs:

- 1 **Adjacency-list:** provides a compact way to represent **sparse** graph ($|E| \ll |V|^2$).



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Two standard ways to represent a graph $G = (V, E)$ apply to both directed and undirected graphs:

① **Adjacency-list:** provides a compact way to represent **sparse** graph ($|E| \ll |V|^2$).

② **Adjacency-matrix:** preferred when the graph is **dense** ($|E| \approx |V|^2$) or when need to tell quickly if there is an edge connecting two given vertices.

Adjacency-list representation

Graph

Tran Ngoc Bao Duy



Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V .

Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting



Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V .
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$ or all the vertices adjacent to u in G .

Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

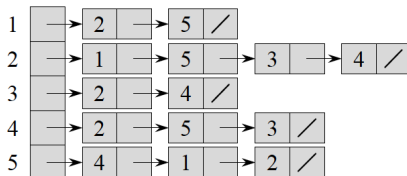
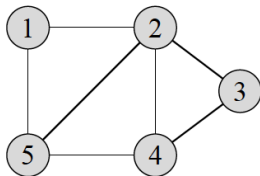
Adjacency-list representation



Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V .
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$ or all the vertices adjacent to u in G .



(Source: Introduction to algorithms - 3rd edition)

Adjacency-list representation

Graph

Tran Ngoc Bao Duy



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

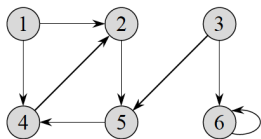
Depth-first search

Topological sorting

Definition

The **adjacency-list representation** of a graph $G = (V, E)$:

- Consists of an array Adj of $|V|$ lists, one for each vertex in V .
- For each $u \in V$, the adjacency list $Adj[u]$ contains all the vertices such that there is an edge $(u, v) \in E$ or all the vertices adjacent to u in G .



1	→	2	→	4	/
2	→	5	/		
3	→	6	→	5	/
4	→	2	/		
5	→	4	/		
6	→	6	/		

(Source: Introduction to algorithms - 3rd edition)

Adjacency-list: Properties

- 1 If G is a **directed** graph, the sum of the lengths of all the adjacency lists is $|E|$.
- 2 If G is a **undirected** graph, the sum of the lengths of all the adjacency lists is $2|E|$.
- 3 The amount of memory required for both cases of graph is $\Theta(V + E)$.



Adjacency-list: Properties

- 1 If G is a **directed** graph, the sum of the lengths of all the adjacency lists is $|E|$.
- 2 If G is a **undirected** graph, the sum of the lengths of all the adjacency lists is $2|E|$.
- 3 The amount of memory required for both cases of graph is $\Theta(V + E)$.

Disadvantages: provides no quicker way to determine whether a given edge (u, v) is present in the graph than to search for in the adjacency list $Adj[u]$.





Definition

Assume that the vertices are numbered $1, 2, \dots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



Definition

Assume that the vertices are numbered $1, 2, \dots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

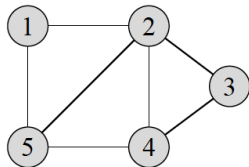
$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Adjacency-matrix representation

Definition

Assume that the vertices are numbered $1, 2, \dots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	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

(Source: Introduction to algorithms - 3rd edition)

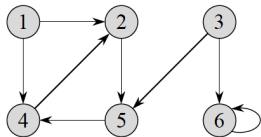


Adjacency-matrix representation

Definition

Assume that the vertices are numbered $1, 2, \dots, |V|$ in some arbitrary manner, the **adjacency-matrix representation** of a graph $G = (V, E)$ consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$



	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

(Source: Introduction to algorithms - 3rd edition)



Adjacency-matrix: Properties

- ① Requires $\Theta(V^2)$ memory, independent of the number of edges in the graph.
- ② Along the main diagonal of the adjacency matrix, there is a symmetry such that the adjacency matrix A of an undirected graph is its own transpose: $A = A^T$.





Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

To represent **weighted graphs**, that is, graphs for which each edge has an associated **weight**, typically given by a weight function $w : E \rightarrow \mathbb{R}$.

- Adjacency-list: store the weight $w(u, v)$ of the edge $(u, v) \in E$ with vertex v in u 's adjacency list.
- Adjacency-matrix: store the weight $w(u, v)$ of the edge $(u, v) \in E$ as the entry in row u and column v of the adjacency matrix. If an edge does not exist, it stores NIL, 0 or ∞ up to your convenience.

- ① Adjacency-list: requires less memory, required simply by most of the graph algorithms.
- ② Adjacency-matrix: simpler and preferred when graphs are reasonably small, requires only one bit per entry.





Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

SEARCHING A GRAPH

Searching a graph

Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.
- Many algorithms begin by searching their input graph to obtain this structural information.





Searching a graph means systematically following the edges of the graph so as to visit the vertices of the graph.

- A graph-searching algorithm can discover much about the structure of a graph.
- Many algorithms begin by searching their input graph to obtain this structural information.
- Techniques for searching a graph lie at the heart of the field of graph algorithms.



Graph representation

Adjacency-list representation
Adjacency-matrix representation
Weighted graphs

Searching a graph

Breadth-first search
Depth-first search

Topological sorting

Definition

- Given a graph $G = (V, E)$ and a distinguished **source** vertex s , **breadth-first search** systematically explores the edges of G to *discover* every vertex that is reachable from s by computing the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a **breadth-first tree** with root s that contains all reachable vertices.



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Definition

- Given a graph $G = (V, E)$ and a distinguished **source** vertex s , **breadth-first search** systematically explores the edges of G to *discover* every vertex that is reachable from s by computing the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a **breadth-first tree** with root s that contains all reachable vertices.

Properties:

- For any vertex reachable from s , the simple path in the breadth-first tree from s to it corresponds to a *shortest path* from s to it in G , that is, a path containing the smallest number of edges.



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Definition

- Given a graph $G = (V, E)$ and a distinguished **source** vertex s , **breadth-first search** systematically explores the edges of G to *discover* every vertex that is reachable from s by computing the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a **breadth-first tree** with root s that contains all reachable vertices.

Properties:

- For any vertex reachable from s , the simple path in the breadth-first tree from s to it corresponds to a *shortest path* from s to it in G , that is, a path containing the smallest number of edges.
- Works on both directed and undirected graphs.

Breadth-first search: Pseudocode

BFS(G, s)

```
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = WHITE$ 
3       $u.d = \infty$ 
4       $u.\pi = NIL$ 
5   $s.color = GRAY$ 
6   $s.d = 0$ 
7   $s.\pi = NIL$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = DEQUEUE(Q)$ 
12     for each  $v \in G.Adj[u]$ 
13         if  $v.color == WHITE$ 
14              $v.color = GRAY$ 
15              $v.d = u.d + 1$ 
16              $v.\pi = u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color = BLACK$ 
```



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

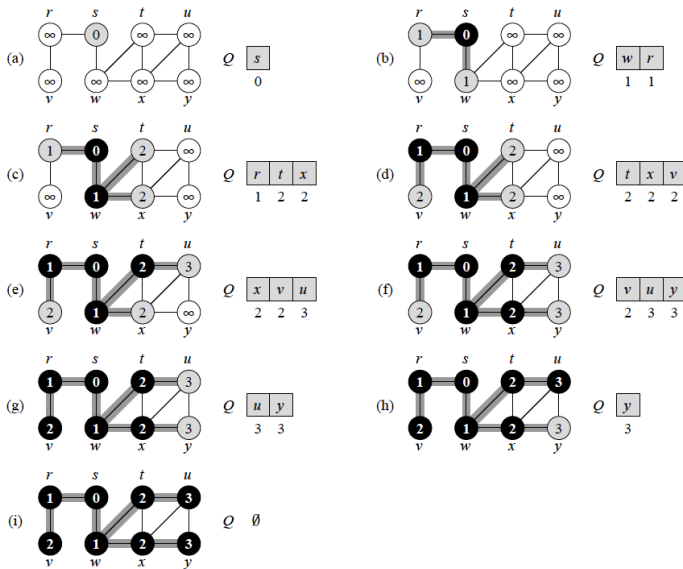
Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Breadth-first search



(Source: Introduction to algorithms - 3rd edition)



Graph representation

Adjacency-list representation
Adjacency-matrix representation
Weighted graphs

Searching a graph

Breadth-first search
Depth-first search

Topological sorting

Shortest path between two vertices

Assuming that BFS has already computed, the following procedure prints out the vertices on a shortest path from s to v :

```
PRINT-PATH( $G, s, v$ )  
1  if  $v == s$   
2      print  $s$   
3  elseif  $v.\pi == \text{NIL}$   
4      print “no path from”  $s$  “to”  $v$  “exists”  
5  else PRINT-PATH( $G, s, v.\pi$ )  
6      print  $v$ 
```



Graph representation

- Adjacency-list representation
- Adjacency-matrix representation
- Weighted graphs

Searching a graph

- Breadth-first search
- Depth-first search

Topological sorting

Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

- 1 Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.



Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

- ① Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- ② Once all of v 's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.



Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

- ① Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- ② Once all of v 's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.
- ③ Continues until all the vertices that are reachable from the original source vertex discovered.



Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

- ① Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- ② Once all of v 's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.
- ③ Continues until all the vertices that are reachable from the original source vertex discovered.
- ④ If any undiscovered vertices remain, then depth-first search selects one of them as a new source, and it repeats the search from that source.



Depth-first search

Depth-first search is the strategy to search *deeper* in the graph whenever possible.

- ① Explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- ② Once all of v 's edges have been explored, the search *backtracks* to explore edges leaving the vertex from which v was discovered.
- ③ Continues until all the vertices that are reachable from the original source vertex discovered.
- ④ If any undiscovered vertices remain, then depth-first search selects one of them as a new source, and it repeats the search from that source.
- ⑤ The algorithm repeats this entire process until it has discovered every vertex.



Depth-first search: Pseudocode

DFS(G)

```
1  for each vertex  $u \in G.V$ 
2       $u.color = \text{WHITE}$ 
3       $u.\pi = \text{NIL}$ 
4   $time = 0$ 
5  for each vertex  $u \in G.V$ 
6      if  $u.color == \text{WHITE}$ 
7          DFS-VISIT( $G, u$ )
```

DFS-VISIT(G, u)

```
1   $time = time + 1$                                 // white vertex  $u$  has just been discovered
2   $u.d = time$ 
3   $u.color = \text{GRAY}$ 
4  for each  $v \in G.Adj[u]$                             // explore edge  $(u, v)$ 
5      if  $v.color == \text{WHITE}$ 
6           $v.\pi = u$ 
7          DFS-VISIT( $G, v$ )
8   $u.color = \text{BLACK}$                                 // blacken  $u$ ; it is finished
9   $time = time + 1$ 
10  $u.f = time$ 
```

Graph

Tran Ngoc Bao Duy



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Depth-first search

Graph

Tran Ngoc Bao Duy



Graph representation

Adjacency-list representation

Adjacency-matrix representation

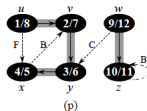
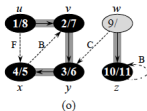
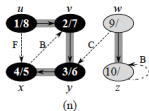
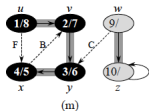
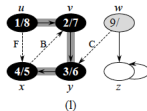
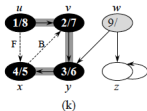
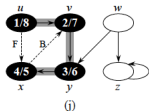
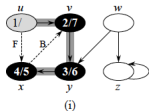
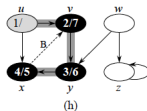
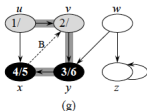
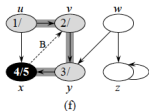
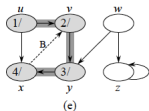
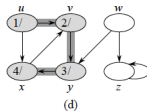
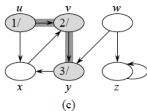
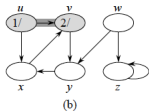
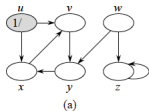
Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting



TOPOLOGICAL SORTING



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

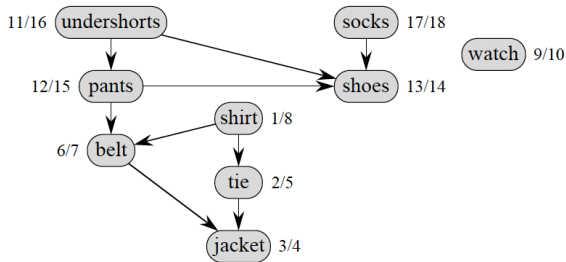
Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Bumstead dressing



Graph

Tran Ngoc Bao Duy



Graph representation

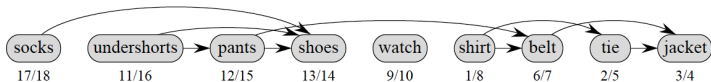
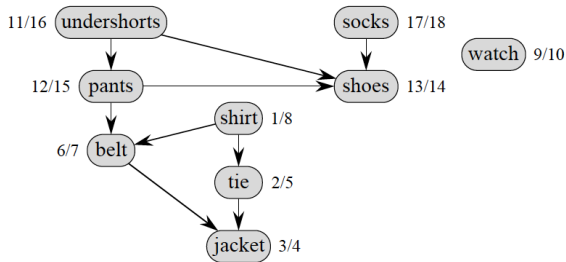
Adjacency-list representation
Adjacency-matrix representation
Weighted graphs

Searching a graph

Breadth-first search
Depth-first search

Topological sorting

Bumstead dressing



Graph

Tran Ngoc Bao Duy



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

Definition

A **topological sort** of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before in the ordering. If the graph contains a cycle, then no linear ordering is possible.



Graph representation

- Adjacency-list representation
- Adjacency-matrix representation
- Weighted graphs

Searching a graph

- Breadth-first search
- Depth-first search

Topological sorting



Graph representation

Adjacency-list representation
Adjacency-matrix representation
Weighted graphs

Searching a graph

Breadth-first search
Depth-first search

Topological sorting

Definition

A **topological sort** of a directed acyclic graph $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v) then u appears before in the ordering. If the graph contains a cycle, then no linear ordering is possible.

TOPOLOGICAL-SORT(G)

- 1 call DFS(G) to compute finishing times $v.f$ for each vertex v
- 2 as each vertex is finished, insert it onto the front of a linked list
- 3 **return** the linked list of vertices

- A common application of topological sorting is in scheduling a sequence of jobs.



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

- A common application of topological sorting is in scheduling a sequence of jobs.
- The jobs are represented by vertices, and there is an edge from x to y if job x must be completed before job y can be started

For example, in constructing a building, the basement must be completed before the first floor, which must be completed before the second floor and so on.



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

- A common application of topological sorting is in scheduling a sequence of jobs.
- The jobs are represented by vertices, and there is an edge from x to y if job x must be completed before job y can be started

For example, in constructing a building, the basement must be completed before the first floor, which must be completed before the second floor and so on.

- A topological sort gives an order in which we should perform the jobs.



Graph representation

Adjacency-list representation

Adjacency-matrix representation

Weighted graphs

Searching a graph

Breadth-first search

Depth-first search

Topological sorting

THANK YOU.



Graph representation

- Adjacency-list representation
- Adjacency-matrix representation
- Weighted graphs

Searching a graph

- Breadth-first search
- Depth-first search

Topological sorting