8.1 Introduction

8.2 Introduction to Graphs

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Graphs

A *Graph* : $G = (V, E)$

1. Definition: Undirected Graphs; Directed Graphs
2. Terms

   vertices, edges, paths, degrees, connected graphs, cycles etc.
3. Facts such as:

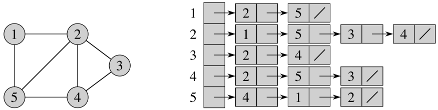
$$\text{Handshaking Lemma} : \sum_{v \in V} \deg(v) = 2|E|$$

4. Representation: Adjacency Matrix; Adjacency List

Illustrations: Graphs and its basic features I

Illustrations: Graphs and its basic features II

Undirected Graphs



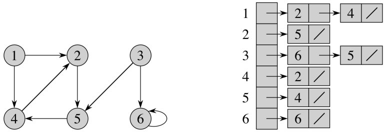
(a) (b)

$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

(c)

Figure : Representing an un-directed graph

Directed Graphs



(a) (b)

	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

(c)

Figure : Representing a directed graph

Question

Draw the adjacency matrix for the *directed* graph $H = (V_H, E_H)$ where

1. Vertex set of $H = V_H = \{1, 2, 3, 4, 5\}$
2. The set of edges of H:

$$E_H = \{(1\ 2), (2\ 3), (1\ 3), (2\ 4), (5\ 1), (5\ 3)\}$$

Answer:

Illustrations: Adjacency Lists and its features

8.3 Breadth First Search I

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

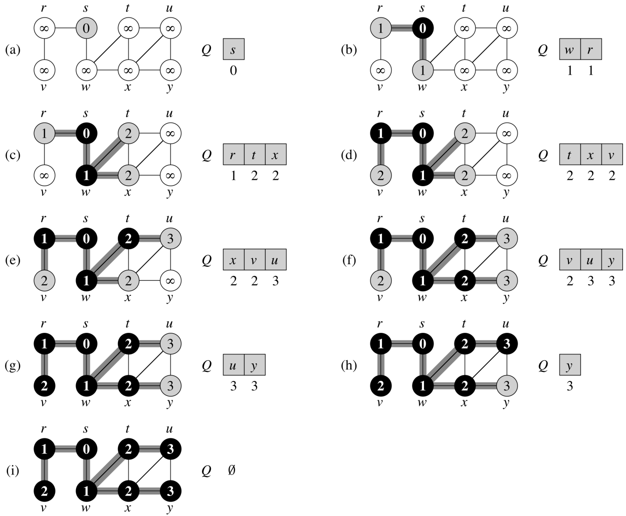
Graph Search: Basic Algorithms

1. Breadth First Search
2. Depth First Search

Illustrations for Basic Graph Search Strategies

Breadth First	Depth First

Breadth First Search Example



Illustrations: Breadth First Search

Question

Question Fill in the blanks:

When performing *breadth first search*, we use the __________________ data structure to help selecting the next vertex to be examined.

The Breadth First Search Algorithm

BFS(V, E, s)

- **for** each $u \in V - \{s\}$
  - $u.d = \infty$
- $s.d = 0$
- $Q = \emptyset$
- ENQUEUE(Q, s)
- **while** $Q \neq \emptyset$
  - $u = \text{DEQUEUE}(Q)$
  - **for** each $v \in G.Adj[u]$
    - **if** $v.d == \infty$
      - $v.d = u.d + 1$
      - ENQUEUE(Q, v)

Question

Fill in the blanks:

In the pseudocode of the *breadth first search* algorithm given:

s stands for ____________________

$d.u$ stands for ____________________

$G.Adj[u]$ stands for ____________________

8.4 Breadth First Search II

Analyzing Breadth First Search: Background

1. For a graph $G = (V, E)$, its 'size' has two parameters: $|V|$ and $|E|$
2. If the graph is a tree, then $|E| = |V| - 1 = O(|V|)$
3. If the graph is a *complete graph*, then

$$|E| = \frac{|V|(|V| - 1)}{2} = O(|V|^2)$$

4. When $|E| = \Theta(|V|^2)$, we may call that graph a *dense* graph.
5. When $|E| = O(|V|)$, we may call that graph a *sparse* graph.

The Breadth First Search Algorithm

```
BFS(V, E, s)
  for each u ∈ V − {s}
      u.d = ∞
  s.d = 0
  Q = ∅
  ENQUEUE(Q, s)
  while Q ≠ ∅
      u = DEQUEUE(Q)
      for each v ∈ G.Adj[u]
          if v.d == ∞
              v.d = u.d + 1
              ENQUEUE(Q, v)
```

Analyzing Breadth First Search I

Discussions What is the worst case running time for Breadth First Search ?

Analyzing Breadth First Search II

Question

Fill in the blanks:

Given a graph G with n vertices, the worst case running time for running *breadth first search* over G is (in Big-O notation):

Case 1: The graph is *sparse* ____________________

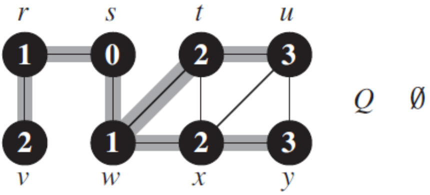
Case 2: The graph is *dense* ____________________

Finding Shortest Path via BFS

1. When performing BFS, we are building a BFS tree.
2. We can *remember* the *predecessor* π of each node in the tree.
3. We can recover the shortest path from the predecessor relation π after running BFS.

Question

Fill in the blanks:



vertex	r	s	t	u	v	w	x	y
predecessor								

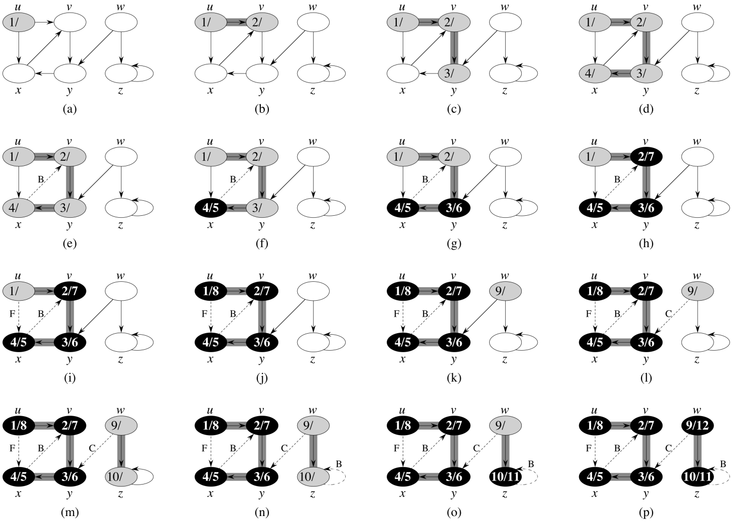
Illustrations: Finding Shortest Path

Finding Shortest Path: Code

```
PRINT-PATH(G, s, v)
if v == s
    print s
elseif v.π == NIL
    print "no path from" s "to" v "exists"
else PRINT-PATH(G, s, v.π)
    print v
```

8.5 Depth First Search I

Depth First Search: Example



Question

Question: Which of the following best describes the key features in the previous sequence of diagrams:

1. Each node is colored either in white, grey or black
2. Each node is eventually labelled by 2 numbers
3. Each edge is either without labelled or eventually labelled by one of these symbols: C, B or F.
4. All of the above

Illustrations: Depth First Search I

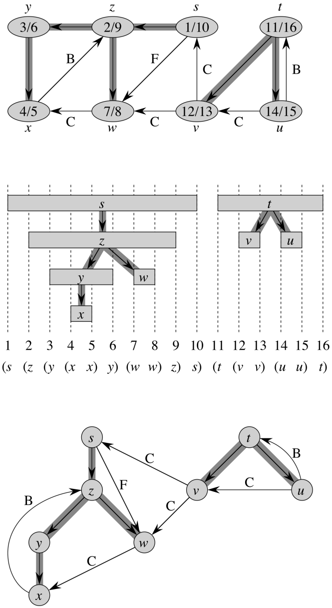
Illustrations: Depth First Search II

Properties of Depth First Search

(a)

(b)

(c)



Pseudocode for DFS (Part I)

```
DFS(G)
  for each u ∈ G.V
        u.color = WHITE
  time = 0
  for each u ∈ G.V
        if u.color == WHITE
              DFS-VISIT(G, u)
```

Pseudocode for DFS (Part II)

```
DFS-VISIT(G, u)
  time = time + 1
  u.d = time
  u.color = GRAY                        // discover u
  for each v ∈ G.Adj[u]                 // explore (u, v)
      if v.color == WHITE
          DFS-VISIT(v)
  u.color = BLACK
  time = time + 1
  u.f = time                            // finish u
```

8.6 Depth First Search II

SYRACUSE UNIVERSITY ENGINEERING & COMPUTER SCIENCE

Pseudocode for DFS

We will re-visit the pseudocode for DFS, which consists of

1. DFS-VISIT
2. DFS

Pseudocode for DFS	Pseudocode for DFS-VISIT

DFS(G)
for each $u \in G.V$
    $u.color = \text{WHITE}$
$time = 0$
for each $u \in G.V$
    if $u.color == \text{WHITE}$
        DFS-VISIT(G, u)
	DFS-VISIT(G, u)
	$time = time + 1$
	$u.d = time$
	$u.color = \text{GRAY}$ // discover u
	for each $v \in G.Adj[u]$ // explore (u, v)
	    if $v.color == \text{WHITE}$
	        DFS-VISIT(v)
	$u.color = \text{BLACK}$
	$time = time + 1$
	$u.f = time$ // finish u

Worst Case Analysis

Discussions It is known that the worst case time complexity for DFS is $\Theta(|V| + |E|)$. Why ?

Question

Question Fill in the blanks:

Given a graph G with n vertices, the worst case running time for running *depth first search* over G is (in Big-O notation):

Case 1: The graph is *sparse* ____________________

Case 2: The graph is *dense* ____________________

Using DFS I

Types of edges:

1. Directed graphs: tree, forward, back, cross edges
2. Undirected graphs: tree and back edges

Question

Fill in the blanks:

After running DFS over a directed graph G, if there is at least one back edge found, then G has a ______________.

Discussions

Discussions If a directed graph G has a cycle, will running DFS over G always produce at least one back edge ?

8.7 Dijkstra's Algorithm

Shortest Path Algorithms I: Introduction

Single Source Shortest Path Problem

- $G = (V, E)$; source vertex s
- Determine a shortest path from s to any vertex v, $v \in V$.

Noteworthy Properties

- Subpaths of shortest paths are shortest paths (Lemma 24.1)
- If G contains no negative weight cycles reachable from the source s, then the shortest-path weight remains well defined (can be negative).
- A shortest path can be made cycle free.

Shortest Path Algorithms II: Variants

Variants may include

1. Traversing an edge in E may include a cost.

   Directed Graph $G = (V, E)$ & weight function $w : E \to \mathbb{R}$

2. The cost of some edges may be negative
3. The graphs we consider has specific properties (e.g. acyclic, disconnected etc.)

Shortest Path Algorithms II: Variants

Similar formulations (See CLRS page 644):

- Single-destination shortest-paths problem
  Reverse the direction of each edge may help
- Single-pair shortest-path problem
  Set the source s to be the beginning vertex
- All-pairs shortest-paths problem
  Run a single source shortest path algorithm *multiple* times
  Can we have a *faster* method ?

Shortest Path Algorithms III: Design Paradigms and new techniques

- Dynamic Programming
- Greedy Choice
- Relaxation Method
- Use properties of shortest paths and relaxation

Relaxation Method: The Two Steps

INIT-SINGLE-SOURCE(G, s)	RELAX(u, v, w)
for each $v \in G.V$	**if** $v.d > u.d + w(u, v)$
$\quad v.d = \infty$	$\quad v.d = u.d + w(u, v)$
$\quad v.\pi = \text{NIL}$	$\quad v.\pi = u$
$s.d = 0$	

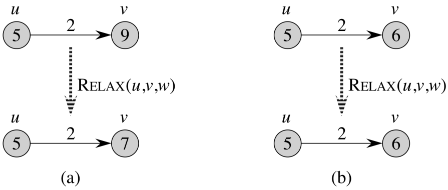


Figure : Relaxing an edge

Relaxation Method: Discussions

Discussions

Can we view relaxation method as a basic step in developing short path algorithms ?

Breadth First Search (BFS) Revisited

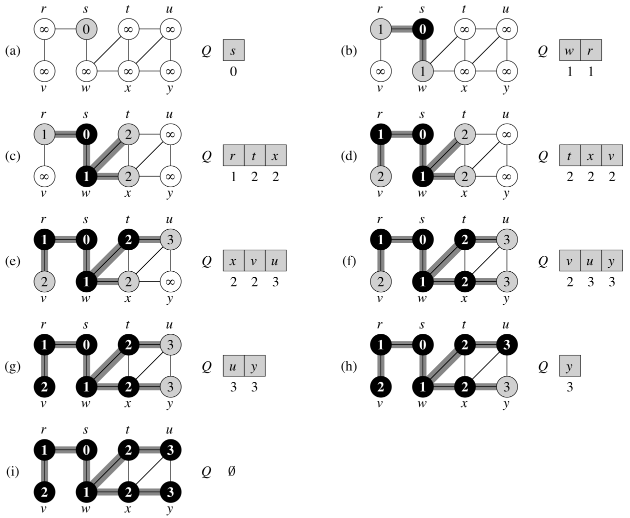


Figure : An illustration for running BFS

Dijkstra's Algorithm

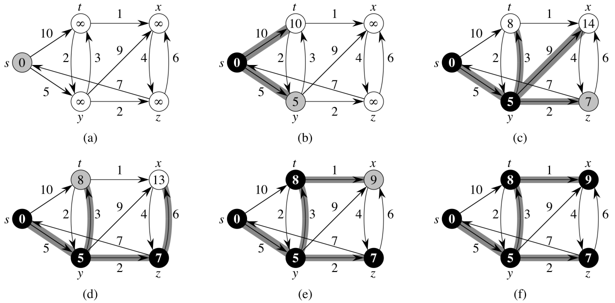


Figure : Running Dijkstra's Algorithm: an example

Questions Can you spot an relaxation step ?
Which data structure may help to locate the next vertex to be explored ?

Dijkstra's Algorithm: Pseudocode

```
DIJKSTRA(G, w, s)
  INIT-SINGLE-SOURCE(G, s)
  S = ∅
  Q = G.V        // i.e., insert all vertices into Q
  while Q ≠ ∅
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Figure : Pseudocode for Dijkstra's Algorithm

Discussions

Does Dijkstra's algorithm apply the *greedy strategy* ?

Dijkstra's Algorithm: Correctness

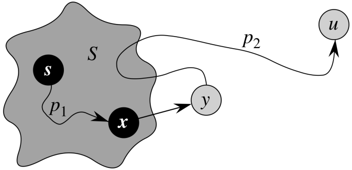


Figure : Why Dijkstra's Algorithm works as expected

Discussions

Use this diagram to explain why Dijkstra's algorithm work.

Dijkstra's Algorithm: Complexity

1. Choice of Data structure matters
2. $O(V^2)$ if an array is used to maintain the priority queue
3. $O((|V| + |E|) lg\,|V|)$ if a binary heap is used to maintain the priority queue
4. Can you more advanced data structure (Fibonacci Heap) to improve the run time (use amortized analysis)