Graph Algorithms

Graph Categories

Example of Digraph

Connectedness of Digraph

Adjacency Matrix

Adjacency Set

Breadth-First Search Algorithm (2 slides)

Dfs()

Strong Components

Graph G and Its Transpose G^T

Shortest-Path Example (2 slides)

Dijkstra Minimum-Path Algorithm (2 slides)

Minimum Spanning Tree Example

Minimum Spanning Tree: vertices A and B

Completing the Minimum Spanning-Tree with Vertices C and D
Graph Categories

- A graph is *connected* if each pair of vertices have a path between them.
- A *complete graph* is a connected graph in which each pair of vertices are linked by an edge.

(a: Connected)  (b: Disconnected)  (c: Complete)
Example of Digraph

- Graph with ordered edges are called *directed graphs* or *digraphs*

Vertices $V = \{A, B, C, D, E\}$

Edges $E = \{(A, B), (A, C), (A, D), (B, D), (B, E), (C, A), (D, E)\}$
Connectedness of Digraph

- **Strongly connected** if there is a path from any vertex to any other vertex.
- **Weakly connected** if, for each pair of vertices \( v_i \) and \( v_j \), there is either a path \( P(v_i, v_j) \) or a path \( P(v_j, v_i) \).

![Diagrams](image)

- (a) Not Strongly or Weakly Connected (No path from \( E \) to \( D \) or \( D \) to \( E \))
- (b) Strongly Connected
- (c) Weakly Connected (No path from \( D \) to a vertex)
An m by m matrix, called an adjacency matrix, identifies the edges. An entry in row i and column j corresponds to the edge $e = (v_i, v_j)$. Its value is the weight of the edge, or -1 if the edge does not exist.
Adjacency Set

(a)

Vertices: A, B, C, D, E
Set of Neighbors:
- A: B
- B: C
- C: B
- D: E
- E: B

(b)

Vertices: A, B, C, D, E
Set of Neighbors:
- A: B, C
- B: A, 2
- C: B, 3
- D: E, 4
- E: C, 1
Breadth-First Search Algorithm

BFS(G, s)
1  for each vertex u ∈ G.V – {s}
2    u.color = WHITE
3    u.d = ∞
4    u.π = NIL
5  s.color = GRAY
6  s.d = 0
7  s.π = NIL
8  Q = ∅
9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11    u = DEQUEUE(Q)
12    for each v ∈ G.Adj[u]
13      if v.color == WHITE
14        v.color = GRAY
15        v.d = u.d + 1
16        v.π = u
17        ENQUEUE(Q, v)
18    u.color = BLACK
Breadth-First Search Algorithm

Diagram:
- A is connected to B, C, D, and F.
- B is connected to D.
- C is connected to F and G.
- D is connected to E.
- E is connected to G.
- F is connected to G.
Depth-First Search Algorithm

**DFS**$(G)$

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

**DFS-\text{VISIT}(G, u)**

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

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>E</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>G</td>
</tr>
</tbody>
</table>
```

Diagram:

- A -> B
- A -> D
- D -> E
- F -> E
- F -> G
- G -> C

Depth-First Search Algorithm
A strongly connected component of a graph G is a maximal set of vertices SC in G that are mutually accessible.
Graph $G$ and Its Transpose $G^T$

- The transpose has the same set of vertices $V$ as graph $G$ but a new edge set $E^T$ consisting of the edges of $G$ but with the opposite direction.
Strong Components

Components
A, B, C
D, F, G
E
Strong Components

**STRONGLY-CONNECTED-COMPONENTS**($G$)

1. call DFS($G$) to compute finishing times $u.f$ for each vertex $u$
2. compute $G^T$
3. call DFS($G^T$), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$ (as computed in line 1)
4. output the vertices of each tree in the depth-first forest formed in line 3 as a separate strongly connected component

![Diagram showing components A, B, C, D, F, G, E]
• Use breadth-first search
Dijkstra Minimum-Path Algorithm

\begin{algorithm}
\textbf{Dijkstra}(G, w, s)
\begin{enumerate}
\item \textsc{Initialize-Single-Source}(G, s)
\item $S = \emptyset$
\item $Q = G. V$
\item \textbf{while} $Q \neq \emptyset$
\item \quad $u = \textsc{Extract-Min}(Q)$
\item \quad $S = S \cup \{u\}$
\item \quad \textbf{for} each vertex $v \in G.\text{Adj}[u]$
\item \quad \quad \textsc{Relax}(u, v, w)
\end{enumerate}
\end{algorithm}

\begin{algorithm}
\textbf{Relax}(u, v, w)
\begin{enumerate}
\item \textbf{if} $v.d > u.d + w(u, v)$
\item \quad $v.d = u.d + w(u, v)$
\item \quad $v.\pi = u$
\end{enumerate}
\end{algorithm}
Dijkstra Minimum-Path Algorithm From A to D Example

- minInfo(B, 4)
- minInfo(C, 11)
- minInfo(E, 4)

Priority queue
Dijkstra Minimum-Path Algorithm From… (Cont…)

minInfo(C, 10)  minInfo(C, 11)  minInfo(E, 4)  minInfo(D, 12)

priority queue

minInfo(C, 10)  minInfo(C, 11)  minInfo(D, 12)

priority queue

minInfo(D, 12)

priority queue

minInfo(D, 12)
Minimum Spanning Tree Example

Network of Hubs

Minimum spanning tree

Minimum amount of cable = 241

Minimum amount of cable = 241
Minimum Spanning Tree: Vertices A and B

Spanning tree with vertices A, B

minSpanTreeSize = 2, minTreeWeight = 2
Completing the Minimum Spanning-Tree Algorithm with Vertices C and D

Spanning tree with vertices A, B, D
minSpanTreeSize = 3, minTreeWeight = 7

Spanning tree with vertices A, B, D, C
minSpanTreeSize = 4, minTreeWeight = 14
Backtracking

```python
solve(S):
    if (final(S)):
        return true
    A = actions(S)
    foreach (a in A):
        S = move(S,a)
        if (solve(S)):
            return true
        S = undo(S,a)
    return false
```
N-queens Problem
Magic-Square Problem

\[
\begin{array}{ccc}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2 \\
\end{array}
\quad
\begin{array}{ccc}
16 & 3 & 2 & 13 \\
5 & 10 & 11 & 8 \\
9 & 6 & 7 & 12 \\
4 & 15 & 14 & 1 \\
\end{array}
\]