Introduction to graph theory and algorithms

Jean-Yves L’Excellent and Bora Uçar

GRAAL, LIP, ENS Lyon, France

CR-09: Sparse Matrix Computations, 23 September 2009
http://graal.ens-lyon.fr/~bucar/CR09/
Outline

1 Definitions and some problems

2 Basic algorithms
   - Breadth-first search
   - Depth-first search
   - Topological sort
   - Strongly connected components

3 Questions
A graph \( G = (V, E) \) consists of a finite set \( V \), called the vertex set and a finite, binary relation \( E \) on \( V \), called the edge set.

Three standard graph models

**Undirected graph:** The edges are unordered pair of vertices, i.e., \( \{u, v\} \in E \) for some \( u, v \in V \).

**Directed graph:** The edges are ordered pair of vertices, that is, \((u, v)\) and \((v, u)\) are two different edges.

**Bipartite graph:** \( G = (U \cup V, E) \) consists of two disjoint vertex sets \( U \) and \( V \) such that for each edge \((u, v) \in E, u \in U \) and \( v \in V \).

An ordering or labelling of \( G = (V, E) \) having \( n \) vertices, i.e., \(|V| = n\), is a mapping of \( V \) onto \( 1, 2, \ldots, n \).
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

The set of rows corresponds to one of the vertex set $R$, the set of columns corresponds to the other vertex set $C$ such that for each $a_{ij} \neq 0$, $(r_i, c_j)$ is an edge.
Matrices and graphs: Square unsymmetric pattern

The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square unsymmetric pattern matrices

\[
A = \begin{pmatrix}
1 & 2 & 3 \\
1 & \times & \times \\
2 & \times & \times \\
3 & & \\
\end{pmatrix}
\]

Graph models

- Bipartite graph as before.
- Directed graph

The set of rows/cols corresponds the vertex set \( V \) such that for each \( a_{ij} \neq 0 \), \((v_i, v_j)\) is an edge. Transposed view possible too, i.e., the edge \((v_i, v_j)\) directed from column \( i \) to row \( j \). Usually self-loops are omitted.
Matrices and graphs: Symmetric pattern

The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square symmetric pattern matrices

\[ A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & & \times \\ 2 & \times & \times & \times \\ 3 & \times & \times & \end{pmatrix} \]

Graph models

- Bipartite and directed graphs as before.
- Undirected graph

The set of rows/cols corresponds the vertex set \( V \) such that for each \( a_{ij}, a_{ji} \neq 0, \{v_i, v_j\} \) is an edge. No self-loops; usually the main diagonal is assumed to be zero-free.
Definitions: Edges, degrees, and paths

Many definitions for directed and undirected graphs are the same. We will use \((u, v)\) to refer to an edge of an undirected or directed graph to avoid repeated definitions.

- An edge \((u, v)\) is said to incident on the vertices \(u\) and \(v\).
- For any vertex \(u\), the set of vertices in \(\text{adj}(u) = \{v : (u, v) \in E\}\) are called the neighbors of \(u\). The vertices in \(\text{adj}(u)\) are said to be adjacent to \(u\).
- The degree of a vertex is the number of edges incident on it.
- A path \(p\) of length \(k\) is a sequence of vertices \(\langle v_0, v_1, \ldots, v_k \rangle\) where \((v_{i-1}, v_i) \in E\) for \(i = 1, \ldots, k\). The two end points \(v_0\) and \(v_k\) are said to be connected by the path \(p\), and the vertex \(v_k\) is said to be reachable from \(v_0\).
Definitions: Components

- An undirected graph is said to be connected if every pair of vertices is connected by a path.

- The connected components of an undirected graph are the equivalence classes of vertices under the “is reachable” from relation.

- A directed graph is said to be strongly connected if every pair of vertices are reachable from each other.

- The strongly connected components of a directed graph are the equivalence classes of vertices under the “are mutually reachable” relation.
A **tree** is a connected, acyclic, undirected graph. If an undirected graph is acyclic but disconnected, then it is a **forest**.

**Properties of trees**

- Any two vertices are connected by a unique path.
- \(|E| = |V| - 1\)

A **rooted tree** is a tree with a distinguished vertex \(r\), called the **root**. There is a **unique path** from the root \(r\) to every other vertex \(v\). Any vertex \(y\) in that path is called an **ancestor** of \(v\). If \(y\) is an ancestor of \(v\), then \(v\) is a **descendant** of \(y\).

The **subtree** rooted at \(v\) is the tree induced by the descendants of \(v\), rooted at \(v\).

A **spanning tree** of a connected graph \(G = (V, E)\) is a tree \(T = (V, F)\), such that \(F \subseteq E\).
A topological ordering of a rooted tree is an ordering that numbers children vertices before their parent.

A postorder is a topological ordering which numbers the vertices in any subtree consecutively.

Connected graph G

Rooted spanning tree

with topological ordering

Rooted spanning tree

with postordering
A permutation matrix is a square $(0, 1)$-matrix where each row and column has a single 1.

If $P$ is a permutation matrix, $PP^T = I$, i.e., it is an orthogonal matrix. Let,

$$A = \begin{pmatrix}
1 & 2 & 3 \\
\times & \times & \\
2 & \times & \times \\
3 & \times & \\
\end{pmatrix}$$

and suppose we want to permute columns as $[2, 1, 3]$. Define $p_{2,1} = 1$, $p_{1,2} = 1$, $p_{3,3} = 1$, and $B = AP$ (if column $j$ to be at position $i$, set $p_{ji} = 1$)

$$B = \begin{pmatrix}
2 & 1 & 3 \\
1 & \times & \times \\
2 & \times & \times \\
3 & \times & \\
\end{pmatrix} = \begin{pmatrix}
1 & 2 & 3 \\
1 & \times & \times \\
2 & \times & \times \\
3 & \times & \\
\end{pmatrix} = \begin{pmatrix}
1 & 2 & 3 \\
1 & 1 & \\
2 & \times & \times \\
3 & \times & \\
\end{pmatrix}$$
A matching in a graph is a set of edges no two of which share a common vertex. We will be mostly dealing with matchings in bipartite graphs.

In matrix terms, a matching in the bipartite graph of a matrix corresponds to a set of nonzero entries no two of which are in the same row or column.

A vertex is said to be matched if there is an edge in the matching incident on the vertex, and to be unmatched otherwise. In a perfect matching, all vertices are matched.

The cardinality of a matching is the number of edges in it. A maximum cardinality matching or a maximum matching is a matching of maximum cardinality. Solvable in polynomial time.
Given a square matrix whose bipartite graph has a perfect matching, such a matching can be used to permute the matrix such that the matching entries are along the main diagonal.

\[
\begin{pmatrix}
1 & 2 & 3 \\
\times & \times & \\
\times & \times & \\
\times & & \\
\end{pmatrix}
\]

Matching in bipartite graphs and permutations
Definitions: Reducibility

Reducible matrix: An $n \times n$ square matrix is reducible if there exists an $n \times n$ permutation matrix $P$ such that

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} \\ O & A_{22} \end{pmatrix},$$

where $A_{11}$ is an $r \times r$ submatrix, $A_{22}$ is an $(n - r) \times (n - r)$ submatrix, where $1 \leq r < n$.

Irreducible matrix: There is no such permutation matrix.

Theorem: An $n \times n$ square matrix is irreducible iff its directed graph is strongly connected.

Proof: Follows by definition.
Definitions: Fully indecomposability

**Fully indecomposable matrix:** There is no permutation matrices $P$ and $Q$ such that

$$PAQ = \begin{pmatrix} A_{11} & A_{12} \\ O & A_{22} \end{pmatrix},$$

with the same condition on the blocks and their sizes as above.

**Theorem:** An $n \times n$ square matrix $A$ is fully indecomposable iff for some permutation matrix $P$, the matrix $PA$ is irreducible and has a zero-free main diagonal.

**Proof:** We will come later in the semester to the “if” part.

Only if part (by contradiction): Let $B = PA$ be an irreducible matrix with zero-free main diagonal. $B$ is indecomposable iff $A$ is as such (why?), therefore assume $B = A$ (that is $A$ is irreducible and has a zero-free diagonal). Suppose $A$ is not fully indecomposable.
Definitions and some problems
Basic algorithms
Questions

Fully indecomposable matrices

Fully indecomposable matrix

There is no permutation matrices $P$ and $Q$ such that

$$PAQ = \begin{pmatrix} A_{11} & A_{12} \\ O & A_{22} \end{pmatrix},$$

with the same condition on the blocks and their sizes as above.

Proof cont.: Let $P_1AQ_1$ be of the form above with $A_{11}$ of size $r \times r$. We may write $P_1AQ_1 = A'Q'$, where $A' = P_1AP_1^T$ with zero-free diagonal, and $Q' = P_1Q_1$ is a permutation matrix which has to permute (why?) the first $r$ columns among themselves, and similarly the last $n - r$ columns among themselves. Hence, $A'$ is in the above form, and $A$ is reducible: contradiction. □
Definitions: Cliques and independent sets

### Clique

In an undirected graph \( G = (V, E) \), a set of vertices \( S \subseteq V \) is a clique if for all \( s, t \in S \), we have \((s, t) \in E\).

**Maximum clique**: A clique of maximum cardinality (finding a maximum clique in an undirected graph is NP-complete).

**Maximal clique**: A clique is a maximal clique, if it is not contained in another clique.

In a symmetric matrix \( A \), a clique corresponds to a subset of rows \( R \) and the corresponding columns such that the matrix \( A(R, R) \) is full.

### Independent set

A set of vertices is an **independent set** if none of the vertices are adjacent to each other. Can we find the largest one in polynomial time?

In a symmetric matrix \( A \), an independent set corresponds to a subset of rows \( R \) and the corresponding columns such that the matrix \( A(R, R) \) is either zero, or diagonal.
Definitions: More on cliques

**Clique:** In an undirected graph $G = (V, E)$, a set of vertices $S \subseteq V$ is a clique if for all $s, t \in S$, we have $(s, t) \in E$.

In a symmetric matrix $A$ corresponds to a subset of rows $R$ and the corresponding columns such that the matrix $A(R, R)$ is full.

**Cliquess in bipartite graphs: Bi-cliques**

In a bipartite graph $G = (U \cup V, E)$, a pair of sets $\langle A, B \rangle$ where $A \subseteq U$ and $B \subseteq V$ is a bi-clique if for all $a \in A$ and $b \in B$, we have $(a, b) \in E$.

In a matrix $A$, corresponds to a subset of rows $R$ and a subset of columns $C$ such that the matrix $A(R, C)$ is full.

The maximum node bi-clique problem asks for a bi-clique of maximum size (e.g., $|A| + |B|$), and it is polynomial time solvable, whereas maximum edge bi-clique problem (e.g., asks for a maximum $|A| \times |B|$) is NP-complete.
Definitions: Hypergraphs

Hypergraph: A hypergraph $H = (V, N)$ consists of a finite set $V$ called the vertex set and a set of non-empty subsets of vertices $N$ called the hyperedge set or the net set. A generalization of graphs.

For a matrix $A$, define a hypergraph whose vertices correspond to the rows and whose nets correspond to the columns such that vertex $v_i$ is in net $n_j$ iff $a_{ij} \neq 0$ (the column-net model).

A sample matrix

\[
\begin{pmatrix}
1 & 2 & 3 & 4 \\
1 & \times & \times & \times \\
2 & \times & \times \\
3 & \times & \times & \times
\end{pmatrix}
\]

The column-net hypergraph model
Basic graph algorithms

Searching a graph: Systematically following the edges of the graph so as to visit all the vertices.
- Breadth-first search,
- Depth-first search.

Topological sort (of a directed acyclic graph): It is a linear ordering of all the vertices such that if \((u, v)\) directed is an edge, then \(u\) appears before \(v\) in the ordering.

Strongly connected components (of a directed graph; why?): Recall that a strongly connected component is a maximal set of vertices for which every pair its vertices are reachable. We want to find them all.

We will use some of the course notes by Cevdet Aykanat (http://www.cs.bilkent.edu.tr/~aykanat/teaching.html)
Breadth-first search: Idea

Graph $G = (V, E)$, directed or undirected with adjacency list representation.

**GOAL:** Systematically explores edges of $G$ to
- discover every vertex reachable from the source vertex $s$.
- compute the shortest path distance of every vertex from the source vertex $s$.
- produce a breadth-first tree (BFT) $G_\Pi$ with root $s$.
  - BFT contains all vertices reachable from $s$.
  - the unique path from any vertex $v$ to $s$ in $G_\Pi$ constitutes a shortest path from $s$ to $v$ in $G$.

**IDEA:** Expanding frontier across the breadth-greedy-
- propagate a wave 1 edge-distance at a time.
- using a FIFO queue: $O(1)$ time to update pointers to both ends.
Breadth-first search: Key components

Maintains the following fields for each \( u \in V \)

- \( \text{color}[u] \): color of \( u \)
  - \( \text{WHITE} \): not discovered yet
  - \( \text{GRAY} \): discovered and to be or being processed
  - \( \text{BLACK} \): discovered and processed
- \( \Pi[u] \): parent of \( u \) (NIL of \( u = s \) or \( u \) is not discovered yet)
- \( d[u] \): distance of \( u \) from \( s \)

Processing a vertex = scanning its adjacency list
Breadth-first search: Algorithm

BFS($G, s$)

for each $u \in V - \{s\}$ do
    color[$u$] $\leftarrow$ WHITE
    $\Pi[u] \leftarrow$ NIL; $d[u] \leftarrow \infty$

color[$s$] $\leftarrow$ GRAY
$\Pi[s] \leftarrow$ NIL; $d[s] \leftarrow 0$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$ do
    $u \leftarrow$ head[$Q$]
    for each $v$ in Adj[$u$] do
        if color[$v$] $= \text{WHITE}$ then
            color[$v$] $\leftarrow$ GRAY
            $\Pi[v] \leftarrow u$
            $d[v] \leftarrow d[u] + 1$
            $\text{ENQUEUE}(Q, v)$
    $\text{DEQUEUE}(Q)$
    color[$u$] $\leftarrow$ BLACK
Breadth-first search: Example

Sample Graph:

FIFO queue $Q$ just after processing vertex $a$
Breadth-first search: Example

FIFO just after queue $Q$ processing vertex

$\langle a \rangle$ -
$\langle a,b,c \rangle$ a
Breadth-first search: Example

FIFO just after queue \( Q \) processing vertex

\[
\begin{align*}
\langle a \rangle & \quad - \\
\langle a, b, c \rangle & \quad a \\
\langle a, b, c, f \rangle & \quad b
\end{align*}
\]
Breadth-first search: Example

Breadth-First Search

\[ s \]
\[ \quad 0 \]
\[ a \]
\[ b \]
\[ c \]
\[ d \]
\[ f \]
\[ g \]
\[ h \]
\[ i \]

FIFO just after
queue \( Q \) processing vertex

\[ \langle a \rangle \quad - \]
\[ \langle a,b,c \rangle \quad a \]
\[ \langle a,b,c,f \rangle \quad b \]
\[ \langle a,b,c,f,e \rangle \quad c \]
Breadth-first search: Example

FIFO just after queue Q processing vertex

\[\langle a \rangle - a\]
\[\langle a, b, c \rangle - b\]
\[\langle a, b, c, f \rangle - c\]
\[\langle a, b, c, f, e, g, h \rangle - f\]
Breadth-first search: Example

\begin{itemize}
\item BFS expands in a wave-like manner, level by level.
\item BFS visits the nodes in the order of their distances from the source node.
\item BFS uses a queue to keep track of the nodes to be visited.
\end{itemize}
Breadth-first search: Example

Breadth-first search: Example

BFS

FIFO just after queue Q processing vertex

\[
\begin{align*}
\langle a \rangle & \rightarrow - \\
\langle a,b,c \rangle & \rightarrow a \\
\langle a,b,c,f \rangle & \rightarrow b \\
\langle a,b,c,f,e \rangle & \rightarrow c \\
\langle a,b,c,f,e,g,h \rangle & \rightarrow f \\
\langle a,b,c,f,e,g,h,d,i \rangle & \rightarrow g
\end{align*}
\]
Breadth-first search: Example

FIFO just after
queue \( Q \) processing vertex

\[
\begin{align*}
\langle a \rangle & \quad - \\
\langle a, b, c \rangle & \quad a \\
\langle a, b, c, f \rangle & \quad b \\
\langle a, b, c, f, e \rangle & \quad c \\
\langle a, b, c, f, e, g, h \rangle & \quad f \\
\langle a, b, c, f, e, g, h, d, i \rangle & \quad h
\end{align*}
\]
Breadth-first search: Example

- Breadth-First Search

- FIFO just after queue Q processing vertex

- (a) -
- (a,b,c) a
- (a,b,c,f) b
- (a,b,c,f,e) c
- (a,b,c,f,e,g,h) f
- (a,b,c,f,e,g,h,d,i) d
Breadth-first search: Example

FIFO queue $Q$ just after processing vertex

- \( \langle a \rangle \)
- \( \langle a,b,c \rangle \)
- \( \langle a,b,c,f \rangle \)
- \( \langle a,b,c,f,e \rangle \)
- \( \langle a,b,c,f,e,g,h \rangle \)
- \( \langle a,b,c,f,e,g,h,d,i \rangle \)

Algorithm terminates: all vertices are processed
Breadth-first search: Analysis

Running time: $O(V+E) = \text{considered linear time in graphs}$

- initialization: $\Theta(V)$
- queue operations: $O(V)$
  - each vertex enqueued and dequeued at most once
  - both enqueue and dequeue operations take $O(1)$ time
- processing gray vertices: $O(E)$
  - each vertex is processed at most once and
  \[ \sum_{u \in V} |\text{Adj}[u]| = \Theta(E) \]
Breadth-first search: The paths to the root

\[ \text{BFS}(G, s), \text{ where } V_\Pi = \{ v \in V : \Pi[v] \neq \text{NIL} \} \cup \{ s \} \text{ and } \\
E_\Pi = \{ (\Pi[v], v) \in E : v \in V_\Pi - \{ s \} \} \]
is a breadth-first tree such that

- \( V_\Pi \) consists of all vertices in \( V \) that are reachable from \( s \)
- \( \forall v \in V_\Pi \), unique path \( p(v, s) \) in \( G_\Pi \) constitutes a \( \text{sp}(s, v) \) in \( G \)

\[ \text{PRINT-PATH}(G, s, v) \]

if \( v = s \) then print \( s \)
else if \( \Pi[v] = \text{NIL} \) then
print no “\( s \rightarrow v \) path”
else
\[ \text{PRINT-PATH}(G, s, \Pi[v]) \]
print \( v \)

Prints out vertices on a \( s \rightarrow v \) shortest path
### Breadth-first search: The BFS tree

**Breadth-First Tree Generated by BFS**

<table>
<thead>
<tr>
<th>BFS(G,a) terminated</th>
<th>BFT generated by BFS(G,a)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="BFS Tree" /></td>
<td><img src="image.png" alt="BFT Tree" /></td>
</tr>
</tbody>
</table>

- BFS(G,a) terminated
- BFT generated by BFS(G,a)
Depth-first search: Idea

- Graph $G = (V, E)$ directed or undirected
- Adjacency list representation
- **Goal**: Systematically explore every vertex and every edge
- **Idea**: search deeper whenever possible
  - Using a LIFO queue (Stack; FIFO queue used in BFS)
Depth-first search: Key components

- Maintains several fields for each \( v \in V \)
- Like BFS, colors the vertices to indicate their states. Each vertex is
  - Initially white,
  - grayed when discovered,
  - blackened when finished
- Like BFS, records discovery of a white \( v \) during scanning \( \text{Adj}[u] \) by \( \pi[v] \leftarrow u \)
Depth-first search: Key components

• Unlike BFS, predecessor graph $G_\pi$ produced by DFS forms spanning forest

• $G_\pi = (V, E_\pi)$ where

$$E_\pi = \{ (\pi[v], v) : v \in V \text{ and } \pi[v] \neq \text{NIL} \}$$

• $G_\pi$ = depth-first forest (DFF) is composed of disjoint depth-first trees (DFTs)
Depth-first search: Key components

- DFS also timestamps each vertex with two **timestamps**
- \(d[v]\): records when \(v\) is first discovered and **grayed**
- \(f[v]\): records when \(v\) is finished and **blackened**
- Since there is only one discovery event and finishing event for each vertex we have \(1 \leq d[v] < f[v] \leq 2|V|\)

**Time axis for the color of a vertex**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[d[v] \quad \quad \quad f[v] \quad \quad \quad 2|V|\]
**Depth-first search: Algorithm**

\[
\text{DFS}(G) \\
\text{for each } u \in V \text{ do} \\
\quad \text{color}[u] \leftarrow \text{white} \\
\quad \pi[u] \leftarrow \text{NIL} \\
\quad \text{time} \leftarrow 0 \\
\text{for each } u \in V \text{ do} \\
\quad \text{if color}[u] = \text{white} \text{ then} \\
\quad \quad \text{DFS-VISIT}(G, u)
\]

\[
\text{DFS-VISIT}(G, u) \\
\quad \text{color}[u] \leftarrow \text{gray} \\
\quad d[u] \leftarrow \text{time} \leftarrow \text{time} + 1 \\
\quad \text{for each } v \in \text{Adj}[u] \text{ do} \\
\quad \quad \text{if color}[v] = \text{white} \text{ then} \\
\quad \quad \quad \pi[v] \leftarrow u \\
\quad \quad \quad \text{DFS-VISIT}(G, v) \\
\quad \quad \text{color}[u] \leftarrow \text{black} \\
\quad f[u] \leftarrow \text{time} \leftarrow \text{time} + 1
\]
Depth-first search: Analysis

- Running time: $\Theta(V+E)$
- Initialization loop in $\text{DFS}$: $\Theta(V)$
- Main loop in $\text{DFS}$: $\Theta(V)$ exclusive of time to execute calls to $\text{DFS-Visit}$
  - $\text{DFS-Visit}$ is called exactly once for each $v \in V$ since
    - $\text{DFS-Visit}$ is invoked only on white vertices and
    - $\text{DFS-Visit}(G, u)$ immediately colors $u$ as gray
- For loop of $\text{DFS-Visit}(G, u)$ is executed $|\text{Adj}[u]|$ time
- Since $\Sigma |\text{Adj}[u]| = E$, total cost of executing loop of $\text{DFS-Visit}$ is $\Theta(E)$
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example

The diagram illustrates a directed graph with nodes labeled S, 2, 3, W, X, Y, Z, and U. The nodes S, 2, and 3 are shaded and represent the starting points of the depth-first search. The search proceeds through the graph, exploring as deeply as possible along each branch before backtracking.
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example

Graph representation of a depth-first search algorithm.
Depth-first search: Example

Graph with nodes labeled 1 to 8, and edges connecting the nodes. The graph includes a node labeled 'S' and another labeled 'T', with various paths and directions indicated by arrows.
Depth-first search: Example
Depth-first search: Example

Diagram of a directed graph with nodes labeled from S to Z, illustrating a depth-first search traversal.
Depth-first search: Example

![Depth-first search example graph](image-url)
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example

[Diagram of a graph with nodes labeled S, W, V, X, Y, Z, and edges between them. The nodes are marked with numbers and arrows indicating the order of exploration.]
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example
Depth-first search: Example

Diagram of a directed graph with nodes labeled 1 to 15 and edges connecting them. The nodes are colored to indicate the order of depth-first search traversal.
Depth-first search: Example
Depth-first search: DFT and DFF

**DFS(G) terminated**

**Depth-first forest (DFF)**
**Thm:** In any DFS of $G=(V,E)$, let $int[v] = [d[v], f[v]]$ then exactly one of the following holds for any $u$ and $v \in V$

- $int[u]$ and $int[v]$ are entirely disjoint
- $int[v]$ is entirely contained in $int[u]$ and $v$ is a descendant of $u$ in a DFT
- $int[u]$ is entirely contained in $int[v]$ and $u$ is a descendant of $v$ in a DFT
Depth-first search: Parenthesis theorem

Parenthesis Theorem

(\(x (s (w w) (v v) s)\)) (\(y (t t) y\)) (\(x)\) (\(z (u u) z\))
Depth-first search: Edge classification

Tree Edge: discover a new (WHITE) vertex
▷GRAY to WHITE◁

Back Edge: from a descendent to an ancestor in DFT
▷GRAY to GRAY◁

Forward Edge: from ancestor to descendent in DFT
▷GRAY to BLACK◁

Cross Edge: remaining edges (btwn trees and subtrees)
▷GRAY to BLACK◁

Note: ancestor/descendent is wrt Tree Edges
Depth-first search: Edge classification

• How to decide which GRAY to BLACK edges are forward, which are cross

Let BLACK vertex \( v \in \text{Adj}[u] \) is encountered while processing GRAY vertex \( u \):

– \((u,v)\) is a forward edge if \( d[u] < d[v] \)
– \((u,v)\) is a cross edge if \( d[u] > d[v] \)
Depth-first search: Edge classification example
Depth-first search: Edge classification example
Depth-first search: Edge classification example

- s
- w
- 1
- 2
- 3
- x
- y
- z
- u
- v
- t
- \( T \)
- \( T \)
- \( T \)
- \( T \)
- \( T \)
- \( T \)
Depth-first search: Edge classification example
Depth-first search: Edge classification example

Depth-first search: Edge classification example
Depth-first search: Edge classification example

Depth-First Search: Example

x y z
s t
wv u
1
2
3 4 5
T
T
T
B
Depth-first search: Edge classification example
Depth-first search: Edge classification example

Diagram:

- Edges:
  - X to Y: T
  - X to Z: B
  - Y to S: T
  - Y to V: T
  - Y to U: T
  - Z to T: T
  - S to W: T
  - W to 3: T
  - W to 4: T
  - 3 to 5: C
  - 4 to 5: C
  - 5 to 6: C
  - 6 to V: C
  - V to U: C
Depth-first search: Edge classification example
Depth-first search: Edge classification example
Depth-first search: Edge classification example

Depth-First Search: Example

x y z
s t
w v u
1 2
3 4 5 6
7 8
T T T T
B C C
Depth-first search: Edge classification example

Depth-First Search:

Example

x y z
s t
w v u
1
2
3
4
5
6
7
8
9
T
T
T
T
B
C
C
Depth-first search: Edge classification example

Graph with vertices and edges labeled with classifications such as T, C, B.
Depth-first search: Edge classification example
Depth-first search: Edge classification example
Depth-first search: Edge classification example
Depth-first search: Edge classification example
Depth-first search: Edge classification example

Depth-first search: Edge classification example
Depth-first search: Edge classification example
Depth-first search: Edge classification example

- Definitions and some problems
- Basic algorithms
- Questions

Depth-first search
- Breadth-first search
- Topological sort
- Strongly connected components

CS473 – Lecture 16 Cevdet Aykanat - Bilkent University
Depth-first search: Edge classification example
Depth-first search: Edge classification example

Definitions and some problems
Basic algorithms
Questions
Breadth-first search
Depth-first search
Topological sort
Strongly connected components

Depth-First Search: Example

Graph with nodes labeled and edges marked with classifications.
Depth-first search: Undirected graphs

**Edge classification**

Any DFS on an undirected graph produces only Tree and Back edges.
Topological sort (of a directed acyclic graph): It is a linear ordering of all the vertices such that if \((u, v)\) directed is an edge, then \(u\) appears before \(v\) in the ordering.

Ordering is not necessarily unique.
Topological sort: Example

- under short
- socks
- watch
- pants
- shoes
- shirt
- belt
- jacket
- tie

Edges:
- under short → socks
- socks → watch
- pants → shoes
- shoes → shirt
- belt → jacket
- tie → shirt
- socks → under short
- under short → pants
- pants → shoes
- shoes → watch
- watch → shirt
- shirt → belt
- belt → tie
- tie → jacket

Weights:
- 11/16
- 17/18
- 9/10
- 12/15
- 13/14
- 6/7
- 3/4
- 2/5
- 1/8
- 17/18
- 11/16
- 12/15
- 13/14
- 9/10
- 1/8
- 6/7
- 2/5
- 3/4
Topological sort: Algorithm

The algorithm

- run \( \text{DFS}(G) \)
- when a vertex is finished, output it
- vertices are output in the reverse topologically sorted order

Runs in \( O(V + E) \) time — a linear time algorithm.

The algorithm: Correctness

If \((u, v) \in E\), then \( f[u] > f[v] \)

Proof: Consider the color of \( v \) during exploring the edge \((u, v)\), where \( u \) is \text{GRAY}. \( \square \)

\( v \) cannot be \text{GRAY} (otherwise a \text{Back} edge in an acyclic graph !!!).

If \( v \) is \text{WHITE}, then \( u \) is an ancestor of \( v \), hence \( f[u] > f[v] \).

If \( v \) is \text{BLACK}, \( f[v] \) is computed already, \( f[u] \) is going to be computed, hence \( f[u] > f[v] \).
The **strongly connected components** of a directed graph are the equivalence classes of vertices under the “are mutually reachable” relation.

For a graph $G = (V, E)$, the transpose is defined as $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$.

Constructing $G^T$ from $G$ takes $O(V + E)$ time with adjacency list (like the CSR or CSC storage format for sparse matrices) representation (how do you transpose a matrix in CSR or CSC format?).

Notice that $G$ and $G^T$ have the same SCCs.
Strongly connected components: Algorithm

(1) Run $\text{DFS}(G)$ to compute finishing times for all $u \in V$

(2) Compute $G^T$

(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ computed in Step (1)

(4) Output vertices of each DFT in DFF of Step (3) as a separate SCC
**Lemma 1**: no path between a pair of vertices in the same SCC ever leaves the SCC

**Proof**: let \( u \) and \( v \) be in the same SCC \( \Rightarrow u \overset{\leftarrow}{\rightarrow} v \)

let \( w \) be on some path \( u \overset{\leftarrow}{\rightarrow} w \overset{\rightarrow}{\rightarrow} v \Rightarrow u \overset{\rightarrow}{\rightarrow} w \)

but \( v \overset{\rightarrow}{\rightarrow} u \Rightarrow \exists \) a path \( w \overset{\rightarrow}{\rightarrow} v \overset{\rightarrow}{\rightarrow} u \Rightarrow w \overset{\rightarrow}{\rightarrow} u \)

therefore \( u \) and \( w \) are in the same SCC

QED
Strongly connected components: Example

```
a b c
e
d f g h
```

Diagram: A directed graph with nodes a, b, c, d, e, f, g, h and directed edges between them illustrating the strongly connected component relationships.
Strongly connected components: Example

(1) Run DFS(G) to compute finishing times for all \( u \in V \)
Strongly connected components: Example

(1) Run \textbf{DFS}(G) to compute finishing times for all $u \in V$
Strongly connected components: Example

(1) Run DFS(G) to compute finishing times for all \( u \in V \)
Strongly connected components: Example

Vertices sorted according to the finishing times:

\[ \langle b, e, a, c, d, g, h, f \rangle \]
Strongly connected components: Example

(2) Compute $G^T$
(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
(3) Call **DFS**(G<sup>T</sup>) processing vertices in main loop in decreasing **f**[**u**] order: \( \langle b, e, a, c, d, g, h, f \rangle \)
(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
Strongly connected components: Example

(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
Strongly connected components: Example

(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
(3) Call DFS($G^T$) processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
(3) Call $\text{DFS}(G^T)$ processing vertices in main loop in decreasing $f[u]$ order: $\langle b, e, a, c, d, g, h, f \rangle$
Strongly connected components: Example

(4) Output vertices of each DFT in DFF as a separate SCC

- \( C_b = \{b, a, e\} \)
- \( C_g = \{g, f\} \)
- \( C_h = \{h\} \)
- \( C_c = \{c, d\} \)
Strongly connected components: Example

Acyclic component graph

- $a, b, e$
- $f, g$
- $c, d$
- $h$

$C_a = C_b$
$C_f = C_g$
$C_c$
$C_h$
Strongly connected components: Observations

- In any $\text{DFS}(G)$, all vertices in the same SCC are placed in the same DFT.

- In the $\text{DFS}(G)$ step of the algorithm, the last vertex finished in an SCC is the first vertex discovered in the SCC.

- Consider the vertex $r$ with the largest finishing time. It is a root of a DFT. Any vertex that is reachable from $r$ in $G^T$ should be in the SCC of $r$ (why?)
To detect if there exists a permutation matrix $P$ such that

$$PAP^T = \begin{pmatrix}
A_{11} & A_{12} \\
0 & A_{22}
\end{pmatrix},$$

where $A_{11}$ is an $r \times r$ submatrix, $A_{22}$ is an $(n - r) \times (n - r)$ submatrix, where $1 \leq r < n$:

run SCC on the directed graph of $A$ to identify each strongly connected component as an irreducible block (more than one SCC?). Hence $A_{11}$, too, can be in that form (how many SCCs?).
Could not get enough of it: Questions

How would you describe the following in the language of graphs

- the structure of $PAP^T$ for a given square sparse matrix $A$ and a permutation matrix $P$,
- the structure of $PAQ$ for a given square sparse matrix $A$ and two permutation matrices $P$ and $Q$,
- the structure of $A^k$, for $k > 1$,
- the structure of $AA^T$,
- the structure of the vector $b$, where $b = Ax$ for a given sparse matrix $A$, and a sparse vector $x$.

Can you define:

- the row-net hypergraph model of a matrix.
- a matching in a hypergraph (is it a hard problem?).