Algorithms and Data Structures

All Pairs Shortest Paths

Ulf Leser
Content of this Lecture

- All-Pairs Shortest Paths
  - Transitive closure: Warshall’s algorithm
  - Shortest paths: Floyd’s algorithm
- Reachability in Trees
Recall: DFS

- We put every node exactly once on the stack
  - Once visited, never visited again
- We look at every edge exactly once
  - Outgoing edges of a visited node are never considered again
- U can be implemented as bit-array of size $|V|$, allowing $O(1)$ operations
  - Add, remove, getNextUnseen
- Altogether: $O(n+m)$

```java
func void traverse (G, v node, U set) {
    t := new Stack();
    t.put( v);
    U := U \ {v};
    while not t.isEmpty() do
        n := t.pop();
        print n;
        c := n.outgoingNodes();
        foreach x in c do
            if x ∈ U then
                U := U \ {x};
                t.push( x);
            end if;
        end for;
    end while;
}
```
Recall: Transitive Closure

- **Definition**
  
  Let $G=(V,E)$ be a digraph and $v_i, v_j \in V$. The transitive closure of $G$ is a graph $G'=(V, E')$ where $(v_i, v_j) \in E'$ iff $G$ contains a path from $v_i$ to $v_j$.

- TC usually is dense and represented as adjacency matrix
- Compact encoding of reachability information
Shortest Path Problems

- Given a weighted digraph G
- Dijkstra finds the shortest path between a given start node and all other nodes for the case that all edge weights are positive
- All-pairs shortest paths: Given a digraph G with positive or negative edge weights, find the distance between all pairs of nodes
All-Pairs Shortest Paths: General Case

- Transitive closure with distances
- Result is $O(|V|^2)$ space, so don’t try this for large graphs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>B</td>
<td>-3</td>
<td>na</td>
<td>-2</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>C</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td>na</td>
</tr>
<tr>
<td>D</td>
<td>-2</td>
<td>1</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph Diagram](image-url)
Why Negative Edge Weights?

- One application: Transportation company
  - Every route *incurs cost* (for fuel, salary, etc.)
  - Every route *creates income* (for carrying the freight)
- If cost > income, edge weights become negative
  - But still important to find the *best route*
  - Example: Best tour from X to C
No Dijkstra

- Dijkstra’s algorithm does not work
  - Recall that Dijkstra enumerates nodes by their shortest paths
  - Now: Adding a subpath to a so-far shortest path may make it “shorter” (by negative edge weights)
No Dijkstra

- Dijkstra’s algorithm does not work
  - Recall that Dijkstra enumerates nodes by their shortest paths
  - Now: Adding a subpath to a so-far shortest path may make it “shorter” (by negative edge weights)
Negative Cycles

- Shortest path between X and K5?
  - X-K3-K4-K5: 5
  - X-K3-K4-K5-X-K3-K4-K5: 4
  - X-K3-K4-K5-X-K3-K4-K5-X-K3-K4-K5: 3
  - …

- SP-Problem undefined if G contains a **negative cycle**
All-Pairs: First Approach

• We start with a simpler problem: Computing the transitive closure of a digraph $G$ without edge weights

• First idea
  – Reachability is transitive: $x \rightarrow_p y \land y \rightarrow_p z \Rightarrow x \rightarrow_p y \rightarrow_p z = x \rightarrow z$
  – We use this idea to iteratively build longer and longer paths
  – First extend edges with edges – path of length 2
  – Extend paths of length 2 with edges – paths of length 3
  – …
  – No necessary path can be longer then $|V|$
    • Or it would contain a cycle

• In each step, we store “reachable by a path of length $\leq k$” in a matrix
Example – After $z=1, 2, 3, 4$

![Graph and table with path length constraints]

Path length:

- $\leq 2$
- $\leq 3$
- $\leq 4$
- $\leq 5$
**Naïve Algorithm**

- \( M \) is the adjacency matrix of \( G \), \( M'' \) eventually the TC of \( G \)
- \( M' \): Represents paths \( \leq z \)
- \( M'' \): Represents paths \( \leq z+1 \)
- Reachability is transitive:
  \[ i \rightarrow j \land j \rightarrow k \Rightarrow i \rightarrow k \]
- Loops \( i \) and \( j \) look at all pairs reachable by a path of length \( \leq z+1 \)
- Loop \( k \) extends path of length \( \leq z \) by all outgoing edges
- Obviously \( O(n^4) \)

---

\[ G = (V, E); \]
\[ M := \text{adjacency\_matrix}( G); \]
\[ M'' := M; \]
\[ n := |V|; \]
\[ \text{for } z := 1..n-1 \text{ do} \]
\[ \quad M' := M''; \]
\[ \quad \text{for } i = 1..n \text{ do} \]
\[ \quad \quad \text{for } j = 1..n \text{ do} \]
\[ \quad \quad \quad \text{if } M'[i,j]=1 \text{ then} \]
\[ \quad \quad \quad \quad \text{for } k=1 \text{ to } n \text{ do} \]
\[ \quad \quad \quad \quad \quad \text{if } M[j,k]=1 \text{ then} \]
\[ \quad \quad \quad \quad \quad \quad M''[i,k] := 1; \]
\[ \quad \quad \quad \quad \end{if} \]
\[ \quad \quad \end{for} \]
\[ \quad \end{for} \]
\[ \end{for} \]

\( z \) appears nowhere; it is there to ensure that we stop when the longest possible shortest paths has been found.
In the first step, we actually compute $M \times M$, and then replace each value $\geq 1$ with 1.

- We only state that there is a path; not how many and not how long.

- Computing $TC$ can be described as *matrix operations*. 

---

**Observation**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

$\times$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Paths in the Naïve Algorithm

- The naive algorithm always extends paths by one edge
  - Computes $M \times M$, $M^2 \times M$, $M^3 \times M$, ... $M^{n-1} \times M$
Idea for Improvement

• Why not extend paths by all paths found so-far?
  – We compute
    \[ M_2' = M \times M \]: Path of length ≤ 2
    \[ M_3' = M_2' \times M \cup M_2' \times M_2' \]: Path of length ≤ 2+1 and ≤ 2+2
    \[ M_4' = M_3' \times M \cup M_3' \times M_2' \cup M_3' \times M_3' \], lengths ≤ 4+1, ≤ 4+2, ≤ 4+3/4
    ...
    \[ M_n' = ... \cup M_{n-1}' \times M_{n-1}' \]
  – [We will implement it differently]

• Trick: We can stop much earlier
  – The longest shortest path can have length at most n
  – Thus, it suffices to compute \[ M_{\log(n)}' = ... \cup M_{\log(n)}' \times M_{\log(n)}' \]
Algorithm Improved

- We use only one matrix $M$
- We “add” to $M$ matrices $M^2'$, $M^3'$ ...
- In the extension, we see if a path of length $\leq 2^z$ (stored in $M$) can be extended by a path of length $\leq 2^z$ (stored in $M$)
  - Computes all paths $\leq 2^z + 2^z = 2^{z+1}$
- Analysis: $O(n^3 \log(n))$
- But ... we can be even faster

```
G = (V, E);
M := adjacency_matrix( G);
n := |V|;
for z := 0..ceil(log(n)) do
  for i = 1..n do
    for j = 1..n do
      if M[i,j]=1 then
        for k=1 to n do
          if M[j,k]=1 then
            M[i,k] := 1;
          end if;
        end for;
      end if;
    end for;
  end for;
end for;
```
Example – After z=1, 2, 3

Path length: \( \leq 2 \) \( \leq 4 \) Done
Further Improvement

- Note: The path A→D is found twice: A→B→D / A→C→D
- Can we stop “searching” A→D once we found A→B→D?
- Can we enumerate paths such that redundant paths are discovered less often (i.e., less paths are tested)?
Warshall’s Algorithm


- **Preparations**
  - Fix an arbitrary order on nodes and assign each node its rank as ID
  - Let $P_t$ be the set of all paths that contain only nodes with ID $< t+1$
  - $t$ gives the highest allowed node ID inside a path

- **Idea: Compute $P_t$ inductively**
  - We start with $P_1$
  - We compute $P_t$, $t > 1$, based on the assumption that $P_{t-1}$ is known
  - We are done once $t = n$

- **Induction**
  - Suppose we know $P_{t-1}$
  - If we increase $t$ by one, we admit one additional node, i.e., ID $t$
  - Now, every *additional path* must have the form $i \xrightarrow{p_1 \in P_{t-1}} t \xrightarrow{p_2 \in P_{t-1}} k$
    - Paths with all IDs $< t$ are already known
    - Node $t$ is the only new player, must be in all new paths
Warshall’s Algorithm

- Enumerate paths by the **IDs of the nodes they are allowed to contain**
- $t$ gives the highest allowed node ID inside a path

---


---

**Diagram:**

- Path $p$ using nodes with IDs $\{1, \ldots, t\}$
- Path $p_1$ from $i$ to $t$ using nodes with IDs $\{1, \ldots, t-1\}$
- Path $p_2$ from $t$ to $k$ using nodes with IDs $\{1, \ldots, t-1\}$
Algorithm

- Enumerate paths by the IDs of the nodes they are allowed to contain
- t gives the highest allowed node ID inside a path
- Thus, node t must be on any new path
- We find all pairs i,k with \( i \rightarrow t \) and \( t \rightarrow k \)
- For every such pair, we set the path \( i \rightarrow k \) to 1

1. \( G = (V, E) \);
2. \( M := \text{adjacency}_\text{matrix}(G) \);
3. \( n := |V| \);
4. \text{for} \( t := 1..n \) \text{do}
5. \text{for} \( i = 1..n \) \text{do}
6. \text{if} \( M[i,t]=1 \) \text{then}
7. \text{for} \( k=1 \) \text{to} \( n \) \text{do}
8. \text{if} \( M[t,k]=1 \) \text{then}
9. \( M[i,k] := 1 \);
10. \text{end if;}
11. \text{end for;}
12. \text{end if;}
13. \text{end for;}
14. \text{end for;}

Example – Warshall’s Algorithm

A allowed
Connect
E-A with
A-B, A-C

maxID=A

A B C D E
A 1 1
B 1
C 1
D 1
E 1

A B C D E
A 1 1
B 1
C 1
D 1
E 1 1 1
Example – After t=A,B,C,D,E

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B allowed
Connect A-B/E-B with B-D

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C allowed
Connect A-C/E-C with C-D

No news

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D allowed
Connect A-D, B-D, C-D, E-D with D-E

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

E allowed
Connect everything with everything
Little change – Notable Consequences

G = (V, E);
M := adjacency_matrix(G);
n := |V|;
for z := 1..n do
  for i = 1..n do
    for j = 1..n do
      if M[i,j]=1 then
        for k=1 to n do
          if M[j,k]=1 then
            M[i,k] := 1;
          end if;
        end for;
      end if;
    end for;
  end for;
end for;

O(n^4)

G = (V, E);
M := adjacency_matrix(G);
n := |V|;
for t := 1..n do
  for i = 1..n do
    for k=1 to n do
      if M[i,k]=1 then
        M[i,k] := 1;
      end if;
    end for;
  end for;
end for;

Drop z-Loop
Swap i and j loop
Rephrase j into t

O(n^3)
Content of this Lecture

- All-Pairs Shortest Paths
  - Transitive closure: Warshall’s algorithm
  - Shortest paths: Floyd’s algorithm
- Reachability in Trees
Shortest Paths

- Shortest paths: We need to compute the distance between all pairs of reachable nodes
- We use the same idea as Warshall: Enumerate paths using only nodes with IDs smaller than $t$ inside a path
  - Invariant: Before step $t$, $M[i,j]$ contains the length of the shortest path that uses no node with ID higher than $t$
  - When increasing $t$, we find new paths $i \rightarrow t \rightarrow k$ and look at their lengths
  - Thus: $M[i,k] := \min( M[i,k] \cup \{ M[i,t] + M[t,k] \mid i \rightarrow t \land t \rightarrow k \} )$

Example 1/3

A | B | C | D | E | F | G
---|---|---|---|---|---|---
A |   | 1 | 3 |   |   |   
B | -2|   |   | -1| 1 |   
C |   |   |   |   |   |   |
D | 1 | 3 | 2 | 2 | 4 |   |
E |   |   |   | 4 | 1 |   |
F | 0 | 2 | 5 | 1 | 3 |   |
G | 6 |   |   | -1|   |   |
### Example 2/3

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

The above table demonstrates the process of modifying the original matrix to satisfy certain properties, possibly related to graph theory or matrix manipulation. The steps involve selecting elements and applying operations to transform the matrix into a desired form, as indicated by the changes in the values and their positions.
Example 3/3

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>G</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

![Graph](image-url)
Summary ($n = |V|$, $m = |E|$)

- Warshall’s algorithm computes the transitive closure of any unweighted digraph $G$ in $O(n^3)$.
- Floyd’s algorithm computes the distances between any pair of nodes in a digraph without negative cycles in $O(n^3)$.
- Johnson’s alg. solves the problem in $O(n^2 \log(n) + n^2 m)$.
  - Which is faster for sparse graphs.
- Storing both information requires $O(n^2)$.
- Problem is easier for ...  
  - Undirected graphs: Connected components.
  - Graphs with only positive edge weights: All-pairs Dijkstra.
  - Trees: Test for reachability in $O(1)$ after $O(n)$ preprocessing.
Content of this Lecture

• All-Pairs Shortest Paths
  – Transitive closure: Warshall’s algorithm
  – Shortest paths: Floyd’s algorithm
• Reachability in Trees
Gene Ontology – Describing Gene Function

Gene Ontology

Molecular Function
- Catalytic Activity
  - Transferase Activity
  - Kinase Activity
- Binding
  - Nucleotide Binding

Biological Process
- Physiological Process
  - Metabolism
  - Protein Metabolism
- Cellular Process
  - Cell Communication
  - Signal Transduction
  - Protein Modification
- Cellular Process
  - Cell Communication
  - Signal Transduction
  - Protein Modification
Database Annotation InterPro

- Used by many databases
- Allows cross-database search
- Provides fixed meaning of terms
  - As informal textual description, not as formal definitions
A Large Ontology

- As of 10.6.2011
  - 34253 terms
  - 20831 biological process
  - 2844 cellular component
  - 9019 molecular function
  - 1559 obsolete terms
- Depth: >30
- Today: More than 40000 terms
Problem

- To see whether a term X IS_A term Y, we need to check whether Y lies on the path from root to X
- Reachability problem
Reachability in Trees

- Let $T$ be a directed tree. A node $v$ is reachable from a node $w$ iff there is a path from $w$ to $v$.
- Testing reachability requires finding paths:
  - Which is simple in trees.
- Path length is bound by the length of the longest path, i.e., the depth of the tree.
- This means $O(n)$ in worst-case.
- Let’s see whether we can preprocess the data to do this in constant time.
Pre-/Postorder Numbers

- Assume a DFS-traversal
- Build an array assigning each node two numbers

### Preorder numbers
- Keep a counter $\text{pre}$
- Whenever a node is entered the first time, assign it the current value of $\text{pre}$ and increment $\text{pre}$

### Postorder numbers
- Keep a counter $\text{post}$
- Whenever a node is left the last time, assign it the current value of $\text{post}$ and increment $\text{post}$

Examples from S. Trissl, 2007
Ancestry and Pre-/Postorder Numbers

- Trick: A node $v$ is reachable from a node $w$ iff
  \[ \text{pre}(v) > \text{pre}(w) \land \text{post}(v) < \text{post}(w) \]

- Explanation
  - $v$ can only be reached from $w$, if $w$ is “higher” in the tree, i.e., $v$ was traversed after $w$ and hence has a higher preorder number
  - $v$ can only be reached from $w$, if $v$ is “lower” in the tree, i.e., $v$ was left before $w$ and hence has a lower postorder number

- Analysis: Test is $O(1)$