Algorithms and Data Structures II

Lecture 6:

Transitive Closure

https://elms.u-aizu.ac.jp

Definition: Closure

Let $G(V, E)$ be a directed graph.

- Graph $G^+ = (V, E^+)$, where $(v, w) \in E^+$ if and only if there is a path of length at least 1 from v to w in G, is called the transitive closure of G.

- Graph $G^* = (V, E^*)$, where $(v, w) \in E^*$ if and only if there is a path of length at least 0 from v to w in G, is called the reflexive transitive closure of G.

Construction: Closure

Let $G(V, E)$ be a directed graph. $G^n(V, E^n)$ is defined by

$$E^n = \{(i, j) | \exists k \in V.[(i, k) \in E^{n-1} \land (k, j) \in E]\}$$

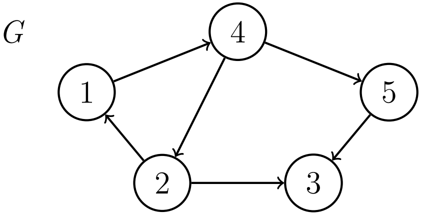
where $E^0 = \{(i, i) | i \in V\}$.

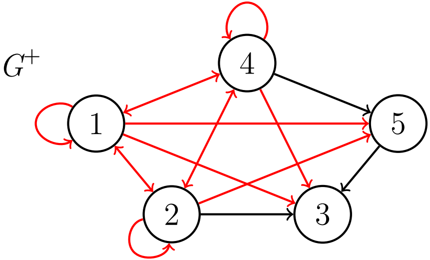
- transitive closure:
$$G^+ = G \cup G^2 \cup \cdots \cup G^n$$

- reflexive transitive closure:
$$G^* = G^0 \cup G \cup G^2 \cup \cdots \cup G^n$$

Example of Transitive Closure

- Let A and A^+ be the adjacency matrices of G and G^+, respectively. A^+ is called the transitive closure of A.





$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A^+ = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Computation of transitive closure

- The transitive closure of $G(V, E)$ can be computed by running DFS or BFS algorithm $|V|$ times on G, taking every vertex of G as the root.
- It takes $O(|V|(|E| + |V|))$ time for a sparse graph, and $O(|V|^3)$ time for a dense graph to compute the transitive closure of G by DFS or BFS.

Warshall's Algorithm

- The following is another algorithm (called Warshall's algorithm) which computes the transitive closure of a graph in $O(|V|^3)$ time.

```
//A is the adjacency matrix of a G(V,E) with |V| = n.
A^0 = A;
for (k = 1; k <= n; k++){
  for (i = 1; i <= n; i++){
    for (j = 1; j <= n; j++){
      // A^n is the transitive closure A^+ of A.
      A^k[i,j] = A^{k-1}[i,j] ∨ (A^{k-1}[i,k] ∧ A^{k-1}[k,j]);}
  }
}
```

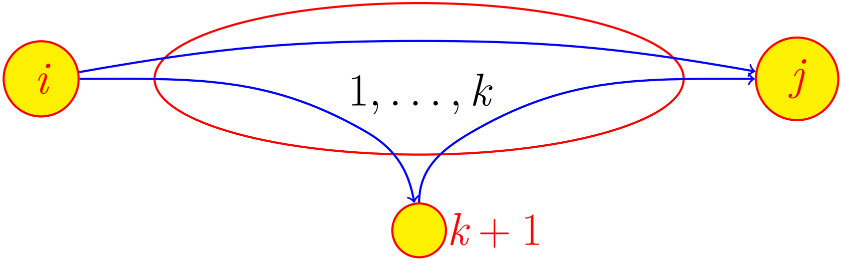
Proof of Warshall's Algorithm by Induction

- Prove by induction on k that:
  - $A^k[i, j] = 1$ if and only if there is path P from vertex i to vertex j such that P (excluding i and j) passes through only the vertices of $\{1, 2, ..., k\}$.
  - **Basis:** $k = 0$. By the definition, $A^0[i, j] = 1$ if there is an edge from i to j. The basis holds.
  - **Induction Hypothesis:** Assume the induction step is valid for k.

Proof of Warshall's Algorithm by Induction

- **Inductive Step:** Prove the statement for $k+1$. Assume path P from i to j passes through only the vertices of $\{1, 2, ..., k+1\}$. Then P passes through either only the vertices of $\{1, 2, ..., k\}$ or vertex $k+1$ and $\{1, 2, ..., k\}$. The later implies a path from i to $k+1$ and a path from $k+1$ to j such that these two paths pass through only the vertices of $\{1, 2, ..., k\}$ (see next slide).

Proof of Warshall's Algorithm by Induction



From the induction hypothesis,
either $A^k[i,j] = 1$ or $A^k[i,k+1] = 1$ and $A^k[k+1,j] = 1$.
Since $A^{k+1}[i,j] = A^k[i,j] \vee (A^k[i,k+1] \wedge A^k[k+1,j])$,
the statement holds for $k+1$.

All Pairs Shortest Path Problem (APSP)

- Let $G(V, E, W)$ be a weighted graph. APSP is to find the shortest paths between every pair of vertices in G.
- If the edge cost in G is non-negative, then APSP can be solved by calling $|V|$ times Dijkstra's algorithm, taking every vertex of G as the source. The following is another algorithm for APSP, called Floyd's Algorithm..

All Pairs Shortest Path Problem (APSP)2

```
// D is the distance matrix of a G(V, E, W) with
|V| = n.
D_0 = D;
for (k = 1; k < = n; k++){
   for (i = 1; i < = n; i++){
      for (j = 1; j < = n; j++){
         //D_n[i, j] is the shortest distance from i to j.
         D_k[i, j] = min{D_{k-1}[i, j], D_{k-1}[i, k] + D_{k-1}[k, j]};
      }
   }
}
```

Floyd's algorithm

- Floyd's algorithm solves APSP in $O(|V|^3)$ time.
- If the graph has negative edge costs then Dijkstra's algorithm **may not work**.
- Floyd's algorithm can solve APSP for the graph G **with negative edge costs** if there is no negative cycle in G.
  - A cycle is negative if the sum of the costs of the edges in the cycle is negative.

Floyd's algorithm

- Given a graph G, Floyd's algorithm can be used to **detect if G has negative cycles**. If there is a negative cycle from i to i, $D^k[i, i]$ will become negative in Floyd's algorithm.

- The superscripts of the matrices in Warshall's and Floyd's algorithms can be dropped.

Matrix Multiplication and Path Problems

- Observing that the transitive closure
$$A^+ = A \vee A^2 \vee \cdots \vee A^n$$
the transitive closure of a graph G can be computed by multiplying the adjacency matrix A, considered as Boolean Matrix, of G.

Matrix Multiplication and Path Problems

- Given an $n \times n$ Boolean matrix, the multiplication $C^2 = C \times C$ can be computed as shown below:

```
for (i = 1; i ≤ n; i++){
    for (j = 1; k ≤ n; j++){
        C²[i, j] = 0;
        for (k = 1; k ≤ n; k++){
            C²[i, j] = C²[i, j] ∨ (C[i, k] ∧ C[k, j]);
        }
    }
}
```

Matrix Multiplication and Path Problems

The transitive closure is then can be computed as:

for (l = 1; l <= n; l = 2*l) $A^{2l} = A^{l} \vee \ (A^{l} \times A^{l})$**;**

It is true, because of,

$$
\begin{aligned}
A^{(1)} &= A, \\
A^{(2)} &= A^{(1)} \vee (A^{(1)} \times A^{(1)}) = A \vee (A \times A) = A \vee A^2, \\
A^{(4)} &= A^{(2)} \vee (A^{(2)} \times A^{(2)}) \\
&= (A \vee A^2) \vee ((A \vee A^2) \times (A \vee A^2)) \\
&= A \vee A^2 \vee A^3 \vee A^4.
\end{aligned}
$$

Matrix Multiplication and Path Problems

In general, sequence of the processes

$$A^{(2k)} = A^{(k)} \vee \left(A^{(k)} \times A^{(k)}\right), \; k = 1, 2, 4, \ldots, n/2$$

can compute

$$A^{(n)} = A \vee A^2 \vee \cdots \vee A^n$$

Matrix Multiplication and Path Problems

Similarly, the **all pairs shortest path problem** can be computed by multiplying the distance matrix over the **closed semi-ring**$(R, \min, +, +\infty, 0)$, R is the set of non-negative reals with $+\infty$.

```
// D is the distance matrix of G(V, E, W) with
|V| = n.
for (l = 1; l ≤ n; l = 2 * l){
   for (i = 1; i ≤ n; i++){
      for (j = 1; j ≤ n; j++){
       //D^n[i, j] gives the shortest distance from i to j.
        D^{2l}[i, j] = min_{k=1...n}(D^l[i, j], D^l[i, k] + D^l[k, j]);}
   }
}
```

Algebraic Path Problem

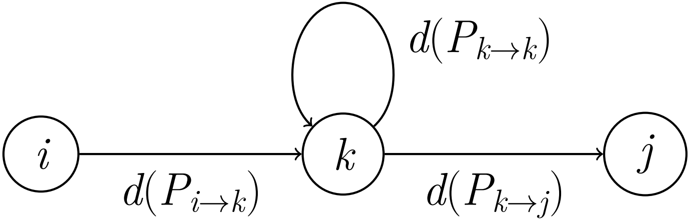
- The only difference in computing the transitive closure of a graph and all pairs shortest distances in the above approaches is that logical operations $\vee$ and $\wedge$ are used for transitive closure while min and $+$ are used for shortest distances.
- In fact these two problems belong to the same class of problems, Algebraic Path Problem (APP).

General form:

$$C_{ij}^{k} = C_{ij}^{k-1} \oplus \left(C_{ik}^{k-1} \otimes (C_{kk}^{k-1})^{*} \otimes C_{kj}^{k-1} \right)$$

Concept of Closure operation ``$(C_{kk}^{k-1})^*$''

- Closure operation ``$(C_{kk}^{k-1})^*$'' stands for $d(P_{k\to k})$, a cost of path from k to k itself, on the following schematic diagram:



Algebraic Path Problem

- Let $G(V, E, w)$ be a weighted graph, where $V = \{1, 2, \ldots, n\}$, $E \subset V \times V$, and $w : E \rightarrow H$ is a function whose codomain is a closed semiring $(H, \oplus, \otimes, *)$ of weights.
- A path p is a sequence of vertices $(v_0, v_1)(v_1, v_2)\ldots(v_{l-1}, v_l)$, where $0 \leqq l$ and $(v_{i-1}, v_i) \in E$. The weight of the path is defined as: $w(p) = w_1 \otimes w_2 \otimes \cdots \otimes w_l$ where w_i is the weight of edge (v_{i-1}, v_i).

Algebraic Path Problem

- The algebraic path problem is that for all pairs of vertices (i, j), find
$$d_{ij} = \oplus\{w(p) : M_{ij}\}$$
where M_{ij} is the set of all paths from i to j.

- The above problem may be formulated in the matrix form. We associate the weighted graph G with an $(n \times n)$-matrix $C = [c_{ij}]$, where $c_{ij} = w(i, j)$ if $(i, j) \in E$ otherwise $c_{ij} = +\infty$.

Algebraic Path Problem

- If $M_{ij}^{(k)}$ stands for the set of all paths from i to j which contain only the vertices x with $1 \leq x \leq k$ as intermediate vertices then we have

$$\begin{aligned} c_{ij}^{(0)} &= c_{ij} \\ c_{ij}^{(k)} &= \bigoplus\{w(p) : p \in M_{ij}^{(k)}\}, \end{aligned}$$

and finally $c_{ij}^{(n)} = d_{ij}$.

Semiring$(R, \oplus, \otimes)$

- $(R, \oplus, 0)$ is commutative monoid:
  - $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
  - $0 \oplus a = a \oplus 0 = a$
  - $a \oplus b = b \oplus a$
- $(R, \otimes, 1)$ is monoid:
  - $(a \otimes b) \otimes c = a \otimes (b \otimes c)$
  - $1 \otimes a = a \otimes 1 = a$
- Distributive properties hold:
  - $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
  - $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- \`\`0'' works as zero on \`\`$\otimes$'':
  - $0 \otimes a = a \otimes 0 = 0$

Applications of APP

- Applications of algebraic path problem are obtained **by specializing the semiring** $(H, \oplus, \otimes, *)$.

Four of them are given below.

- Reflexive and transitive closure of a binary relation (or a graph)
- The shortest paths in a weighted graph
- The minimum-cost spanning tree
- Inverse of a real matrix

Application 1: Reflexive transitive closure

- $c_{ij} \in H = \{0, 1\}$
- $\oplus$ operation is $\vee$ (Boolean ``or").
- $\otimes$ operation is $\wedge$ (Boolean ``and").
- The closure operation (*) is $\forall c \in H$, $c^* = 1$.

Pseudo Code:

$$\begin{aligned} C_{ij}^{k} &= C_{ij}^{k-1} \vee \left(C_{ik}^{k-1} \wedge 1 \wedge C_{kj}^{k-1}\right) \\ &= C_{ij}^{k-1} \vee \left(C_{ik}^{k-1} \wedge C_{kj}^{k-1}\right); \end{aligned}$$

Application 2: The shortest paths in a weighted graph

- The weights $c_{ij} \in H = \mathbb{R}_+ \cup \{+\infty\}$ (c_{ij} is assumed $+\infty$ if no arc from i to j) $\mathbb{R}_+$ is the set of non-negative real numbers.
- $\oplus$ operation is taking **the minimum** in H.
- $\otimes$ operation is the conventional arithmetic operations $+$ in H.
- The closure operation $(^*)$ is $\forall c \in H,\ c^* = 0$

Pseudo Code:

$$C_{ij}^{k} = \min\{C_{ij}^{k-1}, (C_{ik}^{k-1} + 0 + C_{kj}^{k-1})\};$$

Application 3: The minimum-cost spanning tree

- The weights $c_{ij} \in H = \mathbb{R}_+ \cup \{+\infty\}$ (c_{ij} is assumed $+\infty$ if no arc from i to j.) $\mathbb{R}_+$ is the set of non-negative real numbers.
- $\oplus$ operation is taking **the minimum** in H.
- $\otimes$ operation is taking **the maximum** in H.
- The closure operation (*) is $\forall c \in H$, $c^* = 0$.

Pseudo Code:

$$C_{ij}^k = \min\{C_{ij}^{k-1}, \max\{C_{ik}^{k-1}, 0, C_{kj}^{k-1}\}\};$$

Application 4: Computing Inverse $(I_n - A)^{-1}$ via algebraic path problem

- $c_{ij} = a_{ij} \in H = \mathbb{R}$
- $\oplus$ operation is the conventional arithmetic operation $+$ on $\mathbb{R}$.
- $\otimes$ operation is the conventional arithmetic operation $\times$ on $\mathbb{R}$.
- The closure operation is $\forall c \in \mathbb{R}$, $c^* = 1/(1-c)$, with $|c| < 1$, and c^* is undefined for $c = 1$. Solving algebraic path problem yields $(I_n - A)^{-1}$, where I_n is the unit $(n \times n)$-matrix.

Pseudo Code:

$$C_{ij}^{k} = C_{ij}^{k-1} + (C_{ik}^{k-1} \times (1/(1 - C_{kk}^{k-1})) \times C_{kj}^{k-1})$$