CSE525 Monsoon 2020 Ashima Garg (PhD19003)
Homework 13 Problem 1 Pramil Panjawani (PhD19008)

(a) Describe an algorithm to compute the transitive closure of an n-vertex directed graph in $O(n^W log n)$ time.

Solution: Explanation:
Matrix multiplication and Transitive closure estimation are comparable to each other. Task of multiplication is given by C=A.B such that $c_{ij} = \sum_{k=1}^{n} [a_{ik}.b_{kj}]$, where as task of Transitive closure is by C=f(A,B) such that
$c_{ij} = (\forall k = 1 \rightarrow n)\ OR_k(a_{ik}$ AND $b_{kj})$. Since both these matrix multiplication are comparable(summation with min operation and multiplication with addition operation) we can represent Transitive closure problem as a matrix multiplication problem hence they both have same complexity.
Matrix C can be calculated in $log(n)$ times by squaring operation
This gives us a recurrence for the function c(u,v).

$$c_{uv} = (\forall x = u \text{ to } v)\ (\text{OR } (a_{ux} \text{ AND } b_{xv})) \text{ where } a_{ux} \in A \text{ and } b_{xv} \in B$$

$$c(u,v) = \begin{cases} 1 & \text{if path exists} \\ 0 & \text{otherwise} \end{cases}$$

```
def transitive-Closuse(V,E):
  n= |log V|
  ⟨⟨Initializing 2D matrix. Time Complexity: O(n^2)⟩⟩
  for all vertices u:
      for all vertices v:
          d'(u,v)=0
  for i = 1 to n-1:
      for all vertices u:
          for all vertices v:
              for all vertices x:
                  d'(u,v) = [d'(u, v) OR [d(u, x) AND d(x, v)]]
```

Proof:
If the graph contains no negative cycles ,then $c_{uv} = 1$ if path exists and 0 if not. $c_{uv} = 1$ iff $c_{ux} = 1$ (i.e path from vertex u to vertex x exist) and $c_{xv} = 1$ (i.e path from vertex x to vertex v exist)
Observation $C_{n-1} = C_n = C_{n+1} = ...$
$C_X = C_{X-1}W = WW....W(L = 2^{logV} times)$
Induction: $C_X[u,v] = 1$ iff path exists i.e we hit all the vertices on the path of u to v
When we hit x while traveling from u we have already hit all the previous vertices, including x, C(u,x)=1. Similarly when we hit v from x we have hit all the previous vertices leading up to v. C(x,v)=1
If we don't hit from u to x and x to v then the path doesn't exist, so we don't relax the edges.
By, induction if all the edges are relaxed then a path exists.
We repeat this log|V| times selecting the middle vertex x.
Complexity Calculation:
Time Complexity $T(n)$= Number of problems * complexity of Problem + $O(n^2)$
Number of problems = $log(n) - 1 = O(log(n))$
complexity of Problem = Complexity of Boolean multiplication= $O(n^\omega)$ (Given in problem)
T(n) = $O(log(n)) * O(n^\omega) + O(n^2) = O(n^\omega log(n))$ ⟨⟨since $\omega > 2$⟩⟩
Space complexity =$O(n^2)$ since we use a 2d memoization array to store the weight of paths

■

CSE525 Monsoon 2020 Ashima Garg (PhD19003)
Homework 13 Problem 2 Pramil Panjawani (PhD19008)

(b)

Solution: • **Pseudo Code:**

⟨⟨*Topologically sorts the graph G and returns the topologically sorted vertices in S*⟩⟩ ⟨⟨*Reference: book*⟩⟩
def topological_sort(G):
  $clock \leftarrow V$
  for all vertices v in postorder:
    $S[clock] \leftarrow v$
    $clock \leftarrow clock - 1$
  return $S[1 \ldots V]$

def opt(A, B): ⟨⟨*Given two matrices, A and B, compute C*⟩⟩
  $c_{ij} = (\forall k = i$ to $j)$ (OR (a_{ik} AND b_{kj})) where $a_{ik} \in A$ and $b_{kj} \in$B
⟨⟨*This operation is similar to calculating boolean matrix multiplication by just replacing the product in the regular* matrix multiplication with AND operation, and sum of the elementwise multiplication is replaced with OR operation.
⟨⟨*The best known time complexity of matrix multiplication is given in* $O(n^{2.3728}) \equiv O(n^w)$*: Given in question.* ⟩⟩

⟨⟨*Computes second quarter of the matrix and combines all the four quarters.*⟩⟩
def merge($A, i, j, n, m, mid\_h, mid\_v$):
  $A1 \leftarrow A[i : mid\_h][j : mid\_v]$
  $B1 \leftarrow A[mid\_h : n][j : mid\_v]$
  $C1 \leftarrow A[i : mid\_h][mid\_v : m]$
  $D1 \leftarrow A[mid\_h : n][mid\_v : m]$
  $B_{updated} = (A1$ opt $B1)$ opt $C1$ ⟨⟨*opt is defined above* ⟩⟩ ⟨⟨*Time Complexity: O(2*n^ω) ≡ O(n^ω)*⟩⟩
  $A = [[A1, B_{updated}], [C1, D1]]$ ⟨⟨*Concatenate four quarters.*⟩⟩
  return A

⟨⟨*i, j: starting indices of row and column. n, m: ending indices of row and column respectively.* ⟩⟩
⟨⟨*mid_h, mid_v: middle indices of row and column respectively.*⟩⟩
def transitive_closure(A, i, j, n, m):
  if $i < n$ and $j < m$:
    $mid\_h \leftarrow (n - i + 1)/2$, $mid\_v \leftarrow (m - j + 1)/2$
    ⟨⟨*Recursively compute transitive closure of the top-left quarter of the matrix A*⟩⟩
    transitive_closure(A, i, j, mid_h, mid_v)
    ⟨⟨*Recursively compute transitive closure of the bottom-right quarter of the matrix A*⟩⟩
    transitive_closure(A, mid_h, mid_v, n, m)
    return merge($A, i, j, n, m, mid\_h, mid\_v$)

def main(G):
  ⟨⟨*Time complexity of topological sorting of G is: O(n+E)*⟩⟩
  topological_sort(G)
  Create matrix A of size $V * V$ with indices assigned in increasing order of topological sort
⟨⟨*A represents transitive closure matrix where if there is a direct edge from vertex i to j then A[i][j] = 1*⟩⟩
  ⟨⟨*Time Complexity of initializing 2D matrix is:* $O(n^2)$⟩⟩
  for $i \leftarrow 1$ to V:
    for $j \leftarrow 1$ to V:
      if $i \rightsquigarrow j$: $A[i][j] \leftarrow 1$
      if $i \not\rightsquigarrow j$: $A[i][j] \leftarrow 0$
  transitive_closure($A, 1, 1, V, V$):
  return A

- **Explanation:** Find the topological sort of the graph G. With A defined as above stores the final transitive closure of G. We recursively compute transitive closure of the matrix by dividing into four nearly equal parts. Let the four quarters of matrix A: top-left be called as $A1$, top-right as $B1$, bottom-left as $C1$, bottom-right as $D1$. Now, since the given graph is a DAG, and the vertices are topologically sorted, therefore, the entries in $C1$ will always be 0 (explained in Claims and Proofs). While the entries of $A1$ and $D1$ will be recursively computed. To calculate $B1$, we can simply use the $merge$ step as defined in the algorithm. Let E be the number of edges, and in the dense graph, the maximum number of edges possible is n^2

$$
\begin{aligned}
&T(n) = S(n) + O(n+E) + O(n^2)\\
&S(n) = S(n/2) + O((n/2)^\omega)\\
&S(n) = S(n/4) + O((n/2)^\omega + 2*(n/4)^\omega)\\
&S(n) = S(n/2^k) + (2^{k-1}*(n/2^k)^\omega + 2^{k-2}*(n/2^{k-1})^\omega + \ldots + 2^0*(n/2)^\omega)\\
&S(n) = S(n/2^k) + (n/2)^\omega(1/2^{(k-1)w-1} + 1/2^{(k-2)w-1} + \ldots + 1)\\
&S(n) = S(n/2^k) + (n/2)^\omega(\sum_{j=1}^{k=logn(maxlevels)} (1/2^{(j-1)w-1}) \langle\langle \textit{Since decreasing geometric series} \rangle\rangle\\
&S(n) = O(n^\omega)\\
&T(n) = O(n+E) + O(n^2) + O(n^\omega)\\
&T(n) = O(n+n^2) + O(n^\omega) = O(n^\omega) \qquad \langle\langle \textit{E can be max of order } n^2 \rangle\rangle
\end{aligned}
$$

  Space Complexity: Since, 2D matrix is used, therefore, $O(n^2)$ is the space complexity.

- **Proof Of Correctness:** In Topologically sorted DAG if there are vertices $a1$, $a2$, $a3$, $\ldots an$, then there is no path from aj to ai if $aj > ai$. The indices of row and column are arranged in topologically sorted order. Therefore, third quarter $C1$ (a[5][1] to a[8][4]) will always be zero as it has entries $aj][i]$ where $j > i$
  Quarter $A1$ (a[1][1] to a[4][4]) and $D1$ (a[5][5] to a[8][8]) can be recursively computed by dividing into four quarters. Following the same argument of $C1$, the following values will be 0 in quarter $A1$ and $D1$: $a[2][1], a[3][1], a[4][1], a[3][2], a[4][2], a[4][3]$ and $a[6][5], a[7][5], a[7][6], a[8][5], a[8][6], a[8][7]$.
  If $i == j$, implies, there is a path to itself, so, $a[i][j] = 1$. Rest of the values are computed by merge step of algorithm.
  Values in the second quarter $B1$, is equivalent to if there is a path between vertices of $A1$ and $D1$. opt represents $c_{ij} = (\forall k = i \to j)$ (or $(a_{ik}$ and $b_{kj})$) where $a_{ik} \in A1$ and $b_{kj} \in B1$
  A path between $ai \rightsquigarrow bj$ ($ai \in A1$, $bj \in B1$) exists if:

  - it has have direct edge or,
  - it can have a path from ai to ak where $ak \in A1$ and ak has direct edge to bj. This case will be ensured by the first $A1$ opt $B1$.
  - it can have a path from ai to bk where $bk \in B1$ and bk has a path to cl where $cl \in C1$ and cl has a direct edge to bj. This case will be ensured by (Al opt $B1$) opt $C1$.

  Below is transitive closure table after divide and conquer. Since, the graph is DAG and topologically sorted, 0 in the cell values indicate there cannot be any path between them otherwise it will form a cycle. Diagnol 1 indicate that every node has path to itself, and rest Nil values indicate there can be direct and indirect path between them depending on the graph. It shows one level division. Further, it gets divided into blocks of two, then further in blocks of 1.

$TransitiveClosure(\cdot,\cdot)$	a1	a2	a3	a4	a5	a6	a7	a8
a1	1	Nil	Nil	Nil	Nil	Nil	Nil	Nil
a2	0	1	Nil	Nil	Nil	Nil	Nil	Nil
a3	0	0	1	Nil	Nil	Nil	Nil	Nil
a4	0	0	0	1	Nil	Nil	Nil	Nil
a5	0	0	0	0	1	Nil	Nil	Nil
a6	0	0	0	0	0	1	Nil	Nil
a7	0	0	0	0	0	0	1	Nil
a8	0	0	0	0	0	0	0	1

■