Poonam Gupta

COT 5407 – Introduction to Algorithms **Final Exam Take Home**

Problem 1 (5 points) – Transitive Closure of a Dynamic Graph
Given a directed graph $G = (V, E)$, we need to update the transitive closure of the edges inserted so far. Assume that the graph has no edges initially and the transitive closure is represented by a Boolean matrix.

(1) (2 points) Give an $O(V^2)$ algorithm to update the transitive closure $G^* = (G, E^*)$ of a graph $G = (V, E)$ when a new edge $e = (u, v)$ is added;

Let 'A' be the |V|×|V| matrix representing the transitive closure, such that A[i,j] is 1 if there is a path from i to j, and 0 if not:

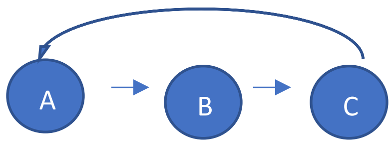
initialize 'A' as follows:
A[i,j] = { 1 if i=j, 0 otherwise}
'A' can be updated as follow when an edge (u, v) is added to G.

```
Transitive-closure-update (u, v)
For i= 1 to |V|
     do for j = 1 to |V|
        do if A [i, u] = 1 and A [v, j] = 1
                then A[i,j] =1
```

In order to add edge (u, v) to graph G, we have to create a path from every vertex that could already reach u to every vertex that could already be reached from v. To accomplish this, we set A[u,v] =1 if and only if the initial values A[u,u] = A[v,v] =1.
As this algorithm involves two nested loops, this takes $O(V^2)$ time.

(2) (1 point) Give an example and an edge $e = (u, v)$ such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used;

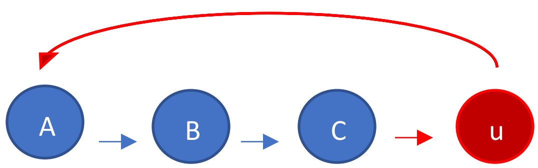
Let's consider the following Graph G with vertices A, B, C. i.e. n=3



In this matrix, there are a total of n^2 (9) entries.

$$G\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

If we add another edge u, we have to change at least $(n+1)^2 - (((n+1)*(n+1+1))/2)$ or $(n+1)^2 - (((n+1)*(n+2))/2)$ or $\Theta(n^2) = \Theta(V^2)$entries. This in the following G we have to change 6 entries.



$$G\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$$

Thus, any algorithm that updates the transitive closure must take $\Omega(V^2)$ time.

(3) (2 points) Give an efficient algorithm for updating the transitive closure as edges are inserted into the graph. For any sequence of n insertions, your algorithm should have a total time$\sum_{i=0}^{n}(t)\mathrm{i}$ = $V(O^3)$, where t_i is the time to update the transitive closure upon inserting the ith edge. Analyze the complexity of your algorithm.

The algorithm in part 1 would take O (V^4) time to insert all possible O (V^2) edges, so we need a more efficient algorithm in order for any sequence of insertion to take only O (V^3) total time.

To improve the algorithm, notice that the loop over j is pointless when A[i,v] = 1. That is , if there is already a path i → v. then adding the edge (u,v) can't make any new vertices reachable from i. the loop to set A[i,j] to 1 for j such that there is a path v → j is just setting entries that are already 1. Remove this redundant processing as follows:

Transitive-closure-update (u, v)

```
1. For i= 1 to |V|
2.     do for A [i, u] = 1 and A [i, v] = 0
3.         then for j =1 to |V|
4.             do if A [v, j] = 1
5.                 then A[i,j] =1
```

Analysis: There can't be more than $|V|^2$ edges in G, so $n \leq |V|^2$. Summed over n insertions, time for the first two lines O(GV) = O (V^3). Lines 3,4 and 5 which takes O(V) time are executed only O(V^2) times for n insertions. To see this, notice that the last three lines are executed only when A [i, v] = 0, and in that case, the last line sets A [i, v] =1. Thus, the number of 0 entries in 'A' is reduced by at least 1 each time the last three lines run. Since there are only $|V^2|$ entries in 'A', these lines can run at most $|V|^2$ times. Hence the total running time over n insertions is O (V^3)

Problem 2 (5 points) – String Matching

(1) (1 point) Suppose that all characters in the pattern P are different. Describe how to accelerate NAIVE-STRING-MATCHER to run in time $O(n)$ on an character text T.

A character mismatch P[i] ≠ T [s + i] for I > 1 indicates that characters in P [1….i] and T[s+1….s+i) matched successfully. As all characters in P are distinct, this partial match means that only P [1] = T[s+1] and thus none of T (s+1…s+i) could match P [1] and start a new potentially valid match. Taking advantage of this fact, our algorithm can skip to character T [s + i] – the first character that can potentially match P [1]:

```
Distinct-chars-pattern-matcher (T, P)
    n = T.length
    m = P.length
    s = 0
    While s ≤ n – m
        i=1
        While i ≤ m and P[i] = T[s+i]
            i = i + 1
        If i = m + 1
            Print "Pattern occurs with shift" s
        s = max (s + 1, s + i -1)
```

(2) (1 point) Working with q=11, how many spurious hits does the Rabin-Karp matcher encounter in the text T=3141592653589793 when looking for pattern P=26?

The number of digits in P and q is 2 therefore window size m becomes 2.
Assume x is the remainder.
X = P mod q
  = 26 mod 11
  = 4
A spurious hit will occur when p has a value, such that the value of P mod q is 4.
A proper match will occur when remainder is 4 and p is 26.
Now, each time extract two digits from the text T, and match as follow.

	Different(P)	R (p mod q)	status
P1	31	9	No hit
P2	14	3	No hit
P3	41	8	No hit
P4	15	4	spurious hit
P5	59	4	spurious hit
P6	92	4	spurious hit
P7	26	4	Proper match
P8	65	10	No hit
P9	53	9	No hit

P10	35	2	No hit
P11	58	3	No hit
P12	89	1	No hit
P13	97	9	No hit
P14	79	2	No hit
P15	93	5	No hit

Observe the above table, the remainder is 4 is obtained, when P is 26, the match is a proper match, but not a spurious hit.
Therefore, the number of spurious hits is 3.

(3) (2 points) Construct the string-matching automaton for the pattern P=aabab and illustrate its operation on the text string T=aaababaabaababaab.

Applying the DFA construction method with P = aabab and {a, b} gives the following transition table:

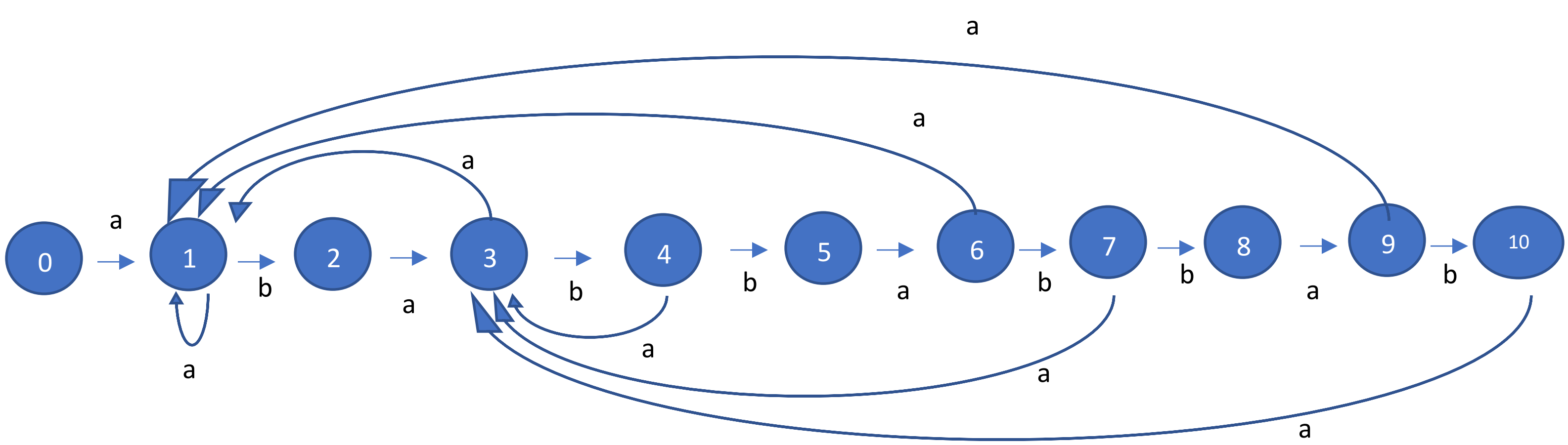
State	a	b
0	1	0
1	2	0
2	2	3
3	4	0
4	2	5
5	1	0

With the following sequence of state transitions for T = aababaabaababaab:

a: 0 → 1
a: 1 → 2
a: 2 → 2
b: 2 → 3
a: 3 → 4
b: 4 → 5 (match)
a: 5 → 1
a: 1 → 2
b: 2 → 3

a: 3 → 4
a: 4 → 2
a: 2 → 3
a: 3 → 4
b: 4 → 5 (match)
a: 5 → 1
a: 1 → 2
b: 2 → 3

(4) (1 point) Draw a state-transition diagram for a string-matching automaton for the pattern P=ab$a$bb$a$bb$a$b over the alphabet Σ= {a, b}.



State	a	b
0	1	0
1	1	2
2	3	0
3	1	4
4	3	5
5	6	0
6	1	7
7	3	8
8	9	0
9	1	10
10	3	0