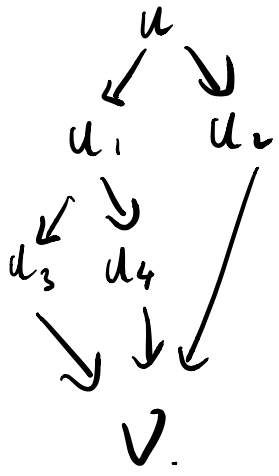
Hongqi Guo.

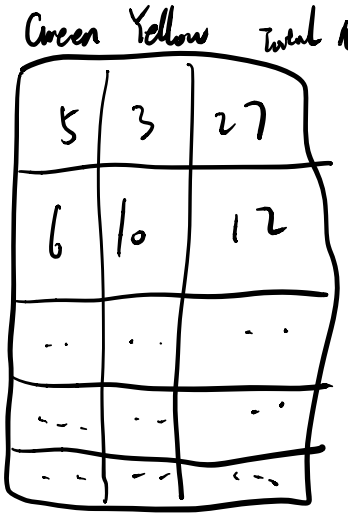
1.

$$u \to u_1 \to u_2 \to u_3 \to u_4 \to v.$$



We will use Topological sort and forward propagation. We will first use Topological sort to determine that the position of u is always before v. Then we plug it into the forward propagation. The principle of the forward propagation is shown in the figure. We will get all possible situations and calculate the weighted value of each route. The one with the highest weight is the shortest path from u to v.

2.



Create a table after traversing one side of the data first. then calculate green, yellow and total nodes in the table. Then use the forward propagation algorithm to analyze. the number of green nodes is greater than the number of yellow nodes is a good path.

3. γ : regexp on colors. → FA on.

G : a graph where each node has G color.

G → G' such that in G', colors are on edges.

$G' \times M_\gamma = M$

count paths in M (from initial to final)

① run scc on M to decide if the circuit is loop or not

so. if it's loop. do SCC.

② if it's not, we can run part I on M, using Topological sort and forward propagation.

4. linear algebra → linear transformation is not a form for CS
→ Matrix = graph.

perron thm.

Let G be a graph with n nodes. M is nxn Matrix, called adjacent matrix of G. set:

$M[i,j] = 1$. if node i → node j is an edge in G.
$M[i,j] = 0$ if node i → node j is not an edge in G.

Assume that G is one big SCC. Then
① M has a unique and largest eigenvalue λ_1
② All other eigenvalues of M, λ', sat $|\lambda'| < \lambda$.

then.
① $M^n = (\underbrace{M \cdot M \cdot M \cdot \cdots M}_{n})$
② $M^n[i,j]$ = the total # of walk from node i to node j in G with length n.
③ M^n can be approx. by when n large. $\lambda^n \; v u^T$.
↑ Perron number of M. ↑ left eigenvector of λ. ↑ the right eigenvector of λ. ← transpose.
④ $M^n[i,j]$ can be approx by $\frac{v_i v_j}{\|u\|} \cdot \lambda^n \cdot v u^T$.
where $\|u\| = \sum_k u_k$, v_i, u_j are the components in vectors u, v.

(5) the total # of walks from node j in G. with length n can be approx by

$$\frac{u_j}{\|u\|} \cdot \lambda^n \cdot u u^T.$$

So path grows to unbounded length are decided by the λ!

$$\lim_{n\to\infty} \frac{\log|S_n|}{n} = \log \lambda.$$

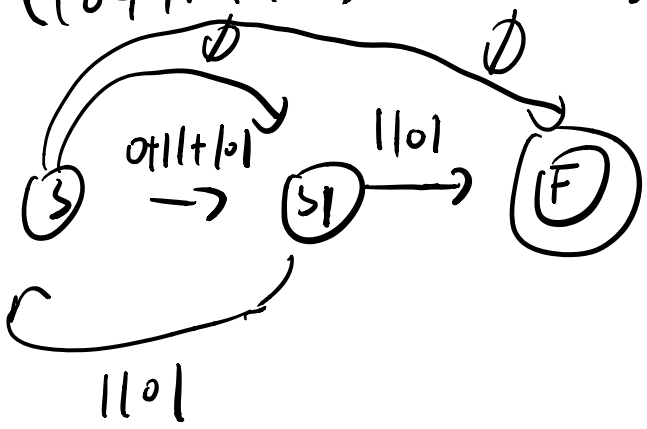
where S_n is the set of all paths with leg $= n$.

For each node u, can be pow number λ_u s.t.:

$$\limsup_{n\to\infty} \frac{\log|C_n(u)|}{n} = \log \lambda_u$$

where $C_n(u)$ is the set of all paths in $C(u)$ with len $= n$.

6. $((0+11+101)^*(1101)^*$



$x[i]$, $y[i]$, $z[i]$ } = the number of strings that can be formed in state S having length L.

→ $x[0] = 1$. $y[0] = 1$ $z[0] = 1$.

→ if $L - 4 > 0$
do $x[L] = y[L]$

else
$x[L] = 0$. (as number of strings of length 1, 2, 3, and 0)

→ $y[L] = x[L] + x[L-1] + x[L-2] + x[L-3]$

(number of strings in S1 of length L is sum of the number of strings of length L, L-1, L-2 and L-3 in state S as from state on generating a string of length null or 1 (string 0) or 2 (string 11) or 3 (string 101) we move to state S1)

→ $x[L] = y[L-4] + x[L]$

→ $x[N]$ return the total number of strings of length N that can be formed.