CS 3313
Foundations of Computing:

Properties of Regular Languages

http://gw-cs3313.github.io

© slides based on material from
Peter Linz book, Hopcroft, Narahari

Properties of Regular Languages

- Closure properties
  - Form new languages/sets by performing operations on regular languages
    - Is the new language regular ?

- Decision properties
  - Ask questions about a language – is there an algorithm that answers the question
    - Is the language empty ? Is it finite ? .....

Review: Closure Properties

- Regular languages are closed under:
  - Union, concatenation, star closure, complement, intersection, reversal, difference, homomorphisms

- If two languages are regular then resulting language from the operations above is also regular
  - There is a DFA (Reg. Expr.) that accepts the language

- Constructing product DFA…..why ?
  - Provides an algorithm to solve the new problem

- Questions ?

Review: Closure Properties – Product DFA

- If L_1 and L_2 are regular then there are DFAs M_1 and M_2 that accept the languages
  - each "problem" can be solved by the DFAs
- Constructing product DFA…..why ?
  - Provides an algorithm to solve the new problem

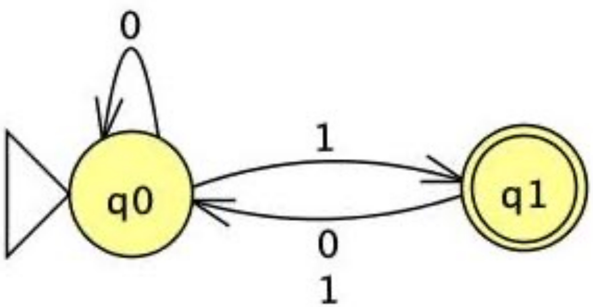
- What is a product DFA = we simulate both DFAs simultaneously in a new DFA
  - The states are the cartesian product of the states of the two machines
  - The transition function simulates both transition functions

Definition: Product DFA

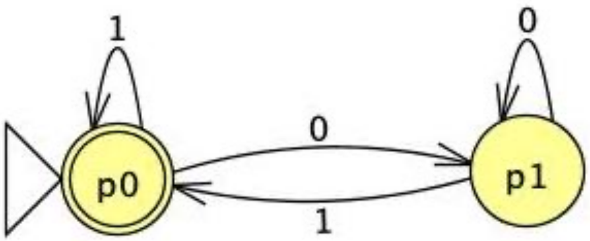
- "compose" two DFAs using cartesian product of their states
- Let M_1 and M_2 be two DFAs with states Q and R
  - $M_1 = (Q, \boldsymbol{\Sigma}, \delta_{1,}\ q_0, F_1)$ and $M_1 = (R, \boldsymbol{\Sigma}, \delta_{2,}\ r_0, F_2)$
- Product DFA M_p:
- Product DFA has set of states $Q \times R$
  - i.e., pairs *[q,r]* with q in Q and r in R
- Start state = $[q_0, r_0]$ (the start states of the two DFA's).
- Transitions: $\delta([q,r], a) = [\delta_1(q,a), \delta_2(r,a)]$
  - δ_1, δ_2 are the transition functions for the DFA's of M_1, M_2
  - That is, ***we simulate the two DFA's in the two state components of the product DFA.***
- Note: we have not yet defined the final states of the product DFA

Example: Product DFA

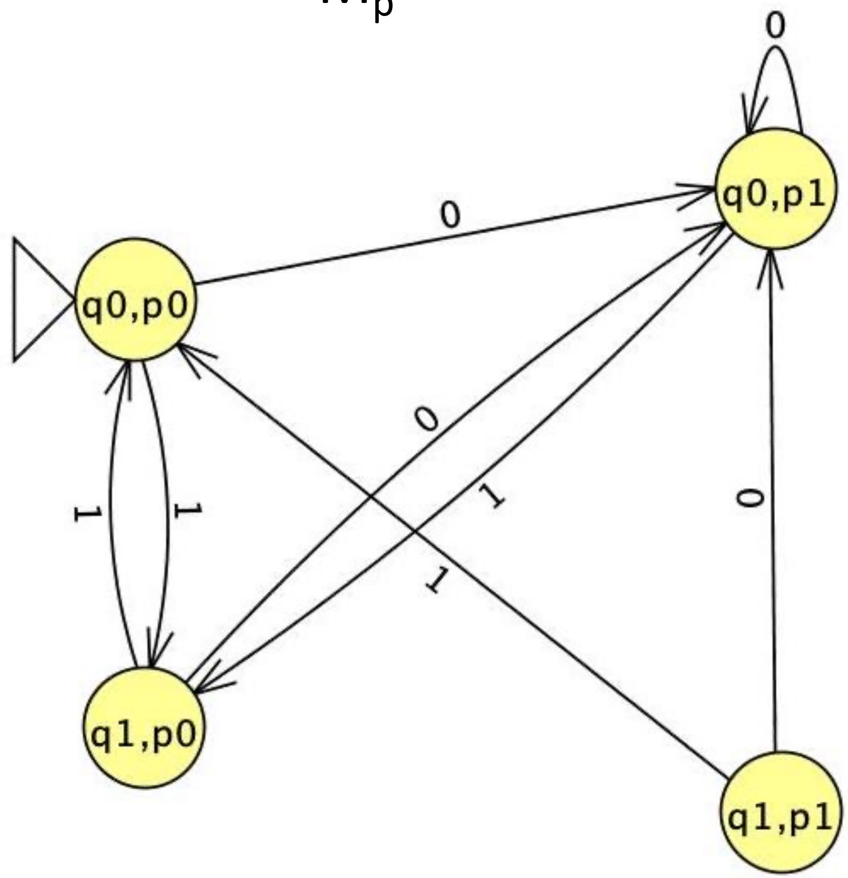
M_1



M_2



M_p



Today's Lab Topics

- Proof of a decision property – Subset property
  - Is L_1 a subset of L_2
- Graph algorithms –because a lot of our proofs/algos about decision properties used graph theory (and graph algorithms)…
  - Quick look at a simple algorithm

Decision Properties of Regular Languages

- L is a regular language iff there is a DFA M such that L(M)=L
- Theorem: Testing emptiness of regular languages is decidable
  - Is $L(M) = \emptyset$
- Theorem: Testing equivalence of regular languages is decidable
  - Is $L(M_1) = L(M_2)$
- Theorem: Testing membership of regular language is decidable
  - Is $w \in L(M_1)$
- Theorem: Testing finiteness of regular language is decidable
  - Is $L(M_1)$ finite ?

- How do we prove a property is decidable = provide an algorithm

Decision Property: Containment

- Given regular languages L_1 and L_2, is $L_1 \subseteq L_2$?
- **Theorem**: Containment property is decidable.
- Proof: will illustrate how we can combine several theorems and properties
- If L_1 and L_2 are regular then we have DFAs M_1 and M_2 such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$
- First question: from definition of subset property

  if $A \subseteq B$ then:
- Next, if A is not a subset of B then:
- In terms of $L(M_1)$ and $L(M_2)$:

Decision Property: Containment

- Given regular languages L_1 and L_2, is $L_1 \subseteq L_2$?
- **Theorem**: Containment property is decidable.
- Proof: will illustrate how we can combine several theorems and properties
- If L_1 and L_2 are regular then we have DFAs M_1 and M_2 such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$
- First question if $A \subseteq B$ then: for every $w \in A$, we have $w \in B$
- Next, if A is not a subset of B then: there is at least one w such that $w \in A$ but w is not in B
- In terms of $L(M_1)$ and $L(M_2)$: there is at least one input w such that w is accepted by M_1 and w is not accepted by M_2

Decision Property: Containment

- Observe: “A is subset of B” is equivalent to NOT (A is not a subset of B)
  - Therefore can we design an algorithm that can check ”(A is not a subset of B)” – if algorithm returns NO then A is a subset of B
- Can you design a DFA M such that w is accepted by M iff w is accepted by M_1 and w is not accepted by M_2 ?
  - This is same as $L(M) = L(M_1) – L(M_2)$

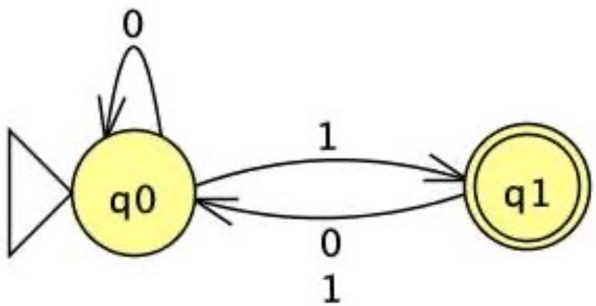
- Question: What is the implication if L(M) is empty ?
- Final question: Is there an algorithm to test if L(M) is empty for a DFA M ?

Example: Product DFA for Subset Checking

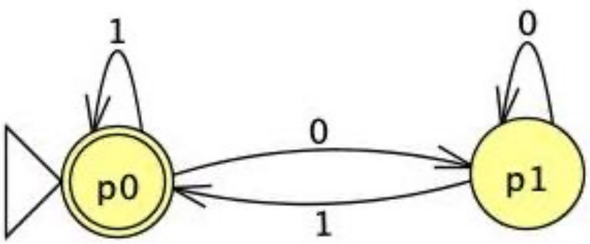
L_1 is not a subset of L_2 iff there is a *w* such that *w* accepted by L_1 and not accepted by L_2

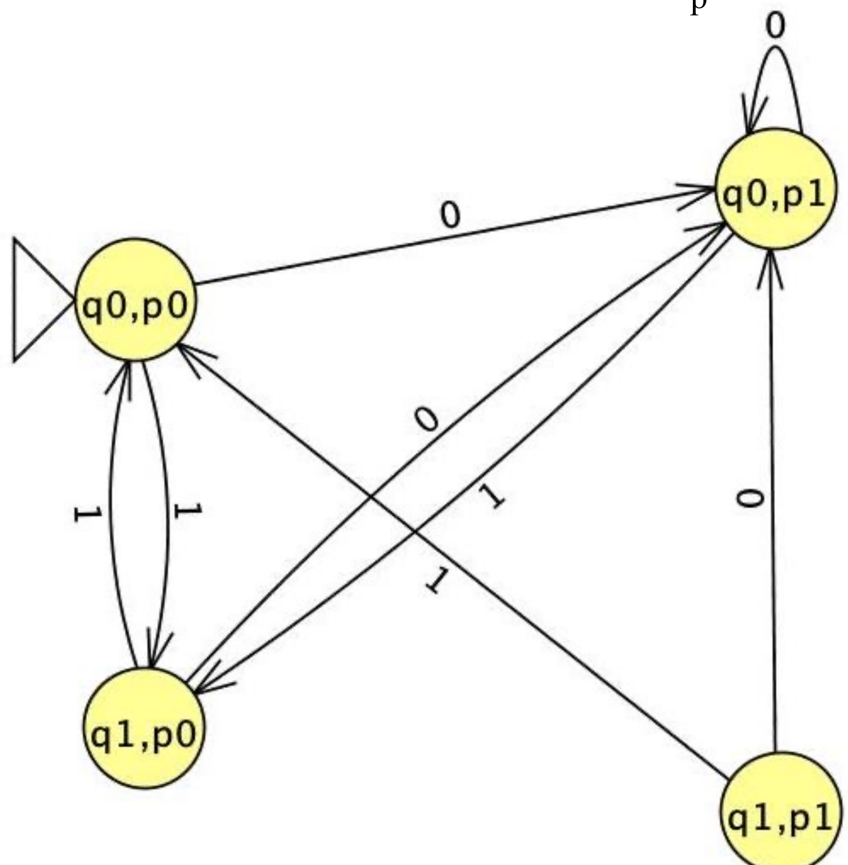
So what are the final states of M_p?

$L_1 = L(M_1)$



$L_2 = L(M_2)$





$L_p = L(M_1) - L(M_2)$

Proof: Containment Property in Regular Languages is decidable

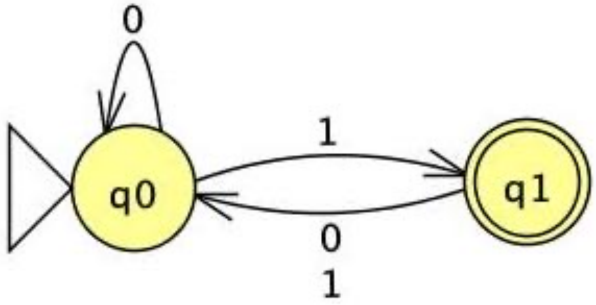
- Given regular languages L_1 and L_2, is $L_1 \subseteq L_2$?
- Theorem: Containment property is decidable.
- Proof:
- Construct product DFA M_p
- How do you define the final states $[q, r]$ of the product so its language is empty iff $L_1 \subseteq L_2$?
  - i.e., there is no string w, such that $w \in L_1$ and $w \notin L_2$
  - [q,r] is final state if q is final and r is not
- Algorithm: Construct this product DFA and call the emptiness testing algorithm

  if product DFA is empty then L_1 is a subset of L_2

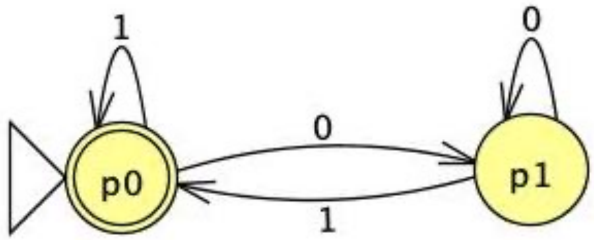
Answer: Product DFA for Subset Checking

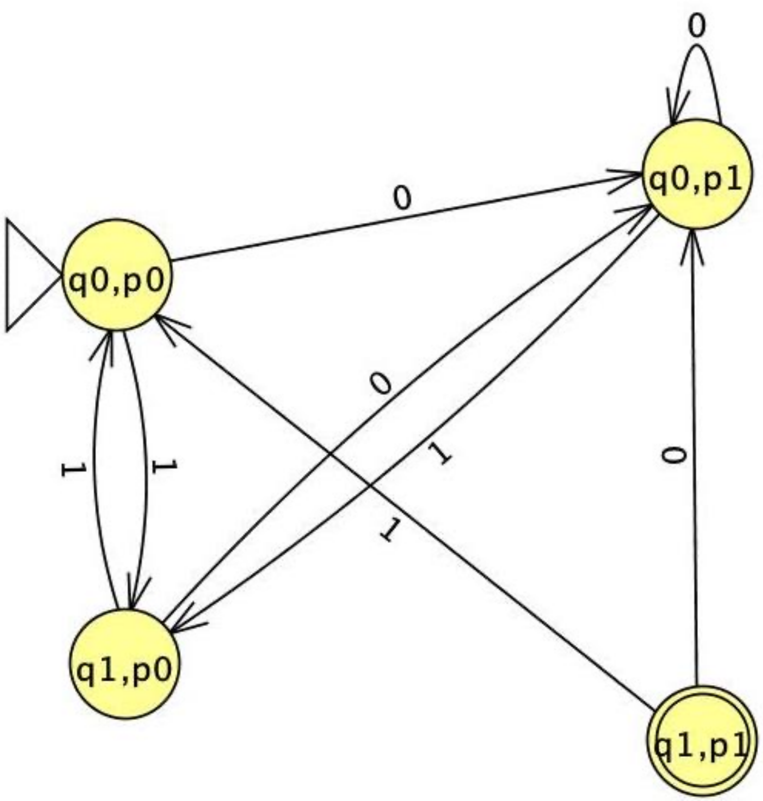
L_1 is subset of L_2 iff no string w such that w accepted by L_1 and not accepted by L_2

$L_1 = L(M_2)$



$L_2 = L(M_2)$





Questions ?

Graph Algorithms

- Very important and useful body of knowledge (i.e., algorithms/solutions) in Computer Science
  - Graphs are everywhere: network is a graph, social media analytics, even code optimization performed by compilers
  - Lot of useful problems can be formulated as a graph problem
    - Ex: How does a compiler assign variables to registers with the goal of maximizing performance (by minimizing memory accesses)….The register allocation problem = graph coloring problem!
    - Ex: How do I route a message from one network node (computer/ IP address) to another = shortest path in a graph
    - Ex: How do we find all twitter users who follow (or are followed by) Ed Sheeran = connected components in a graph

Graph Algorithms – Path finding algorithms

- Path finding algorithms
  - We used these to construct solutions to decision problems about regular languages
- Given graph $G=(V,E)$, find a path from p to q
- Decision version: Is there a path from p to q
- Other questions: is there a cycle in the graph ?
- Today: a simple solution to the graph path finding problem
  - You will cover more efficient algorithms in the algorithms course

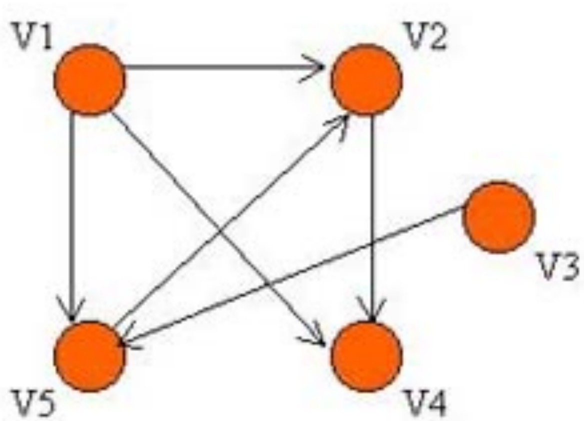
How to represent a graph -- Data Structures

- Adjacency matrix for path: generalize to

  $A[\ i,j] = 1$ if there is a path from v_i to v_j

  $A[\ i,j] = 0$ if no path

Recall: If there is a path then there is a path of length <= $(n-1)$



	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	1
v_2	0	0	0	1	0
v_3	0	0	0	0	1
v_4	0	0	0	0	0
v_5	0	1	0	0	0

How to represent a graph -- Data Structures

- How can we compute paths of length 2 ?
- If there is a path of length 2 from v_i to v_j then there must be an edge from v_i to v_k and an edge from v_k to v_j for some node v_k
- => in initial adjacency matrix: $A^1[i,k] = 1$ and $A^1[k,j]=1$ for some $1 <= k <= n$
- To compute this, check for all *k* – this is multiplication of row *i* with column *j*
- To compute paths of length = 2 for all pairs of nodes, this is matrix multiplication: compute $A^2 = A^1 \times A^1$
- What about paths of length <=2 (length 1 or length 2)

  $$A^{2'} = A^1 + (A^1 \times A^1)$$

- What about paths of length <= k, for any k < n

  $$A^k = A^{k-1} \times A^1 \quad \& \quad A^{k'} = A^{k-1'} + A^k$$

- Algorithm: for k=2 to n

  at each iteration, multiply A^{k-1} with A^1 and add to $A^{k-1'}$

  time = $n\,(O(n^3)) = O(n^4)$

Paths of length = 2

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	1
v_2	0	0	0	1	0
v_3	0	0	0	0	1
v_4	0	0	0	0	0
v_5	0	1	0	0	0

×

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	1
v_2	0	0	0	1	0
v_3	0	0	0	0	1
v_4	0	0	0	0	0
v_5	0	1	0	0	0

=

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	0
v_2	0	0	0	0	0
v_3	0	1	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	1	0

```
A[3,2]=
A[3,k]*A[k,2]
```

Paths of length <= 2

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	0
v_2	0	0	0	0	0
v_3	0	1	0	0	0
v_4	0	0	0	0	0
v_5	0	0	0	1	0

+

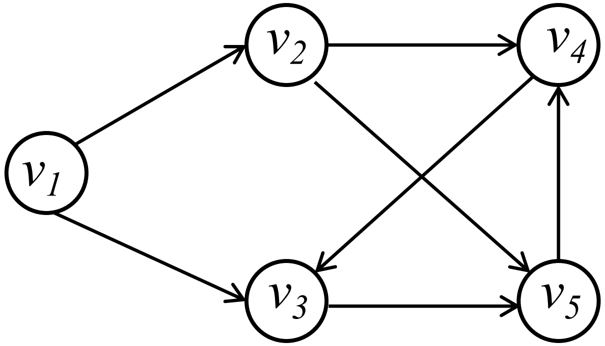
	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	1
v_2	0	0	0	1	0
v_3	0	0	0	0	1
v_4	0	0	0	0	0
v_5	0	1	0	0	0

=

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	1
v_2	0	0	0	1	0
v_3	0	1	0	0	1
v_4	0	0	0	0	0
v_5	0	1	0	1	0

Example

G=(V,E)

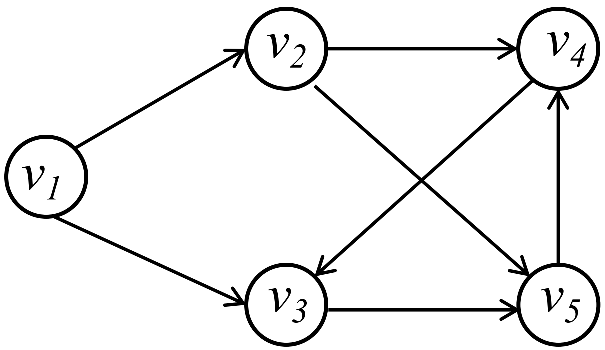


Adjacency matrix $A^1[i,j]$

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	0	0	0	1	1
v_3	0	0	0	0	1
v_4	0	0	1	0	0
v_5	0	0	0	1	0

Example

$G=(V,E)$



Adjacency matrix $A^2[i,j]$
for paths length 2

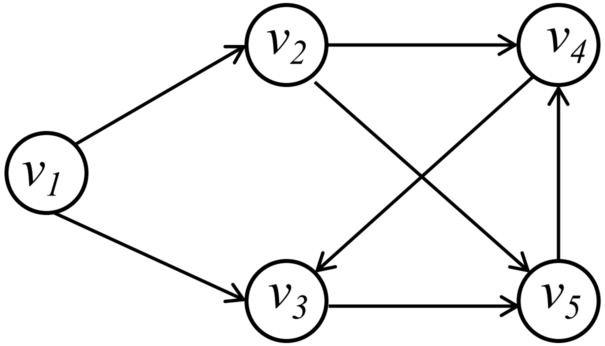
	v_1	v_2	v_3	v_4	v_5
v_1	0	0	0	1	2
v_2	0	0	1	1	0
v_3	0	0	0	1	0
v_4	0	0	0	0	1
v_5	0	0	1	0	0

Adjacency matrix $A^{2'}[i,j]$
for paths length <= 2

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	1	2
v_2	0	0	1	2	1
v_3	0	0	0	1	1
v_4	0	0	1	0	1
v_5	0	0	1	1	0

Example

G=(V,E)

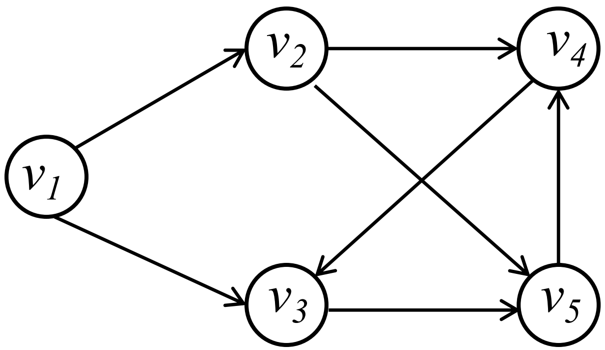


Adjacency matrix $A^{3'}[i,j]$
for paths length <= 3

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	2	3	2
v_2	0	0	2	2	2
v_3	0	0	1	1	1
v_4	0	0	1	1	1
v_5	0	0	1	1	1

Example

G=(V,E)



Adjacency matrix $A^{4'}[i,j]$
for paths length <= 4

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	4	3	3
v_2	0	0	2	3	3
v_3	0	0	1	1	2
v_4	0	0	2	1	1
v_5	0	0	1	2	1

Question 1: Is there a path from v_1 to v_5 ?
Question 2: Is there a path from v_3 to v_2 ?
Question 3: Is there a cycle in the graph ?

Graph Algorithms….

- So is this how graph algorithms are implemented…NO!
- Data structures options: represent graph as linked list
- Much faster algorithms exist G=(V,E) with n vertices
  - Djikstra's algorithm - shortest path between a pair of vertices: $O(n^2)$
    - $O((E+V) \log V)$ can be quicker if $|E|$ is much less than $O(n^2)$
  - All pairs shortest path – Floyd Warshall is $O(n^3)$
    - Very similar to the algorithm for generating Reg. Expr. from a DFA

Putting it all together: Closure Properties and Decision Properties

- Is the following problem decidable:

  "If L_1 and L_2 are regular, then is L_1 intersection L_2 empty ?"

- Proof: (we use multiple results)
  - From the definition of regular languages, If L_1 and L_2 are regular then they are accepted by DFAs M_1 and M_2
  - From closure properties, intersection is regular.
  - To design algorithm, for any two DFAs M_1 and M_2 we construct the product DFA ($M_1 \times M_2$) and set its final states = ($F_1 \times F_2$)
  - Since checking emptiness of a regular language is decidable, we check for emptiness of ($M_1 \times M_2$)
    - Input to algorithm is the product DFA
    - Intersection is empty iff product DFA is empty.