pp. 285-291. The Class P (Sec. 7.2)

- (p. 286) **Definition:** Class P = class of all languages **decidable by 1-tape TM in polynomial time**
  - \( P = \bigcup_k \text{TIME}(n^k) \) for all \( k \)
  - Key: all TM models eqvt to 1-tape TM same complexity
  - Key: close match to problems solvable on real computers

- **Approach to analyzing algorithms for membership in P**
  - See if polynomial upper bound on number of stages
  - See if each stage solvable by polynomial time TM

- All the following are in P
  - (p. 287) **PATH** = \{<G,s,t>| G is directed graph (V,E), with path from s to t\}
    - \( O(N) \): Place mark on vertex s
    - \( O(|V||E|) \): Repeat until no more marked (at most \( |V| \) times)
      - If edge (a,b) leads from marked a to unmarked b, then mark b (at most \( |E| \) times per vertex)
    - \( O(|V|) \): If t is marked, accept, else reject
    - At most \( |V|+2 \) stages, totaling \( O(|V||E|) \) steps
  - (p. 289) **RELPRIME** = \{<x,y>|x,y relatively prime\}
  - (p. 323) Other languages in P: Ex. 7.8, 7.9, 7.10, 7.11, 7.13, 7.14, 7.17
Theorem 7.16. Every CFL has a decider is in P

- i.e. if L expressible by a CFG, then there exists polynomial time decider
- Leads to (p. 322, Ex. 7.4) closure of P under union, concatenation, and complement
- And Ex. 7.15 P closed under star
- Consider following as first notional proof of Theorem:
  - L = \{w \mid w \text{ in a CFL from some CFG } G\}
  - Express G in Chomsky Normal Form (p. 109)
    - All rules of form A->BC or A->t
  - If w in L, |w|=n, any derivation has at most 2n-1 steps
  - Notionally, for particular w, decider for L tries all derivations with 2n-1 steps
  - But this is potentially exponential not polynomial
Better algorithm uses **dynamic programming**:
- Record solution to smaller problems in an nxn table (n=|w|) so don’t need some terms to be repeated
- Table(i,j) = set of G variables that generate $w_i w_{i+1} \ldots w_j$
  - Fill in for string lengths in order 1, 2, ...
    - For length 1, look at A->b rules & record A in cell
    - Use entries for shorter strings in longer ones
  - To generate substring of length k-i+1, split $w_i \ldots w_{k+1}$ into 2 pieces in k different ways
    - $(w_i, w_{i+1} \ldots w_{k+1})$,
    - $(w_i w_{i+1}, w_{i+2} \ldots w_{k+1})$,
    - $(w_i \ldots w_{i+2}, w_{i+3} \ldots w_{k+1})$,
    - ...
    - $(w_i \ldots w_k, w_{k+1})$
  - For each split, examine each rule A->BC to see if B is a generator for 1st part, & C a generator for 2nd part
    - If both, add A to Table(i,j)
  - If S is in Table(1,n) then accept, else reject
- See page 291 for algorithm
- Algorithm executes in $O(n^3)$ time!
- Try Problem 7.4 on p. 322