CS 3313
Foundations of Computing:

More Turing Machine Examples:
Non-determinism, Multitape

http://gw-cs3313-2021.github.io

Nondeterministic TM's

- Allow the TM to have a choice of move at each step.
  - Each choice is a state-symbol-direction triple, as for the deterministic TM.
- The TM accepts its input if a sequence of choices leads to an accepting state.
- Transition function: $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$
  - Set of choices
  - Each choice: goes to a state, writes to tape, moves L or R
  - $\delta(q_0, a) = \{(q_1, b, R), (q_2, c, L)\}$
  - Ex: $aq_0ab \vdash abq_1b$ OR $aq_0ab \vdash q_2acb$

Why use non-determinism

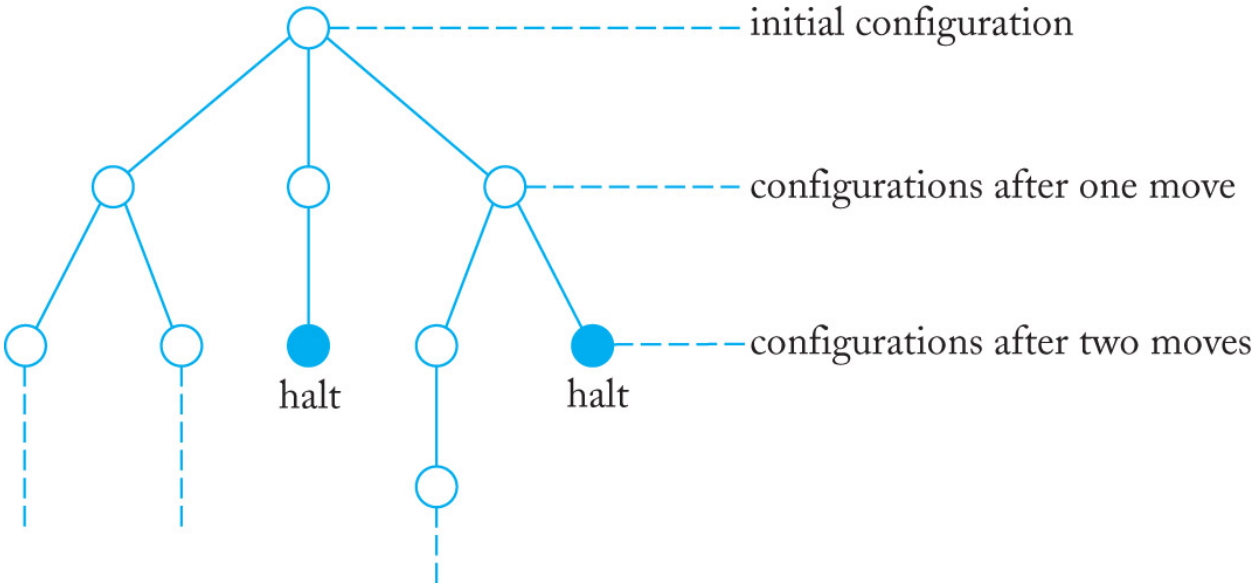
- Powerful expressive model to describe a solution
  - If we are only interested in showing there is a solution
  - So first focus on what non-deterministic machines can solve and how to construct solution
- Can simplify the solution in some cases
  - Ex: “guess” the mid point of ww non-deterministically
- Coding Analogy: imagine you spawn multiple threads, as many as you want, and then wait for one of the threads to complete.
  - Since multiple “transitions” may be applied at each step:

    the program (i.e., machine) may have multiple active simultaneous threads,

    any of which may accept the input string when the thread halts

Behavior of NDTMs

- From a configuration, i.e., ID, $\alpha p a \beta$ after one move the TM goes to another ID $\alpha c q_i \beta$ if $\delta(p, a) = (q_i, c, R)$ is contained in $\delta(p, a)$
  - (q_i, c, R) is one of the choices of moves the machine can make from (p,a)
- Machine starts in ID $q_0 w$



Simulation of a NDTM on a TM

- Theorem: For any Non-deterministic Turing Machine, there is an equivalent deterministic Turing Machine.
  - If a language is accepted (or a function is computed) by a non-deterministic TM then there is a deterministic turing machine that accepts that language (or computes the function).
  - *NDTM and TM are equally powerful and solve the same class of problems !*
- Multi-tape Deterministic TM can simulate a non-deterministic TM
- Simulation time: If NDTM accepts in n moves, and it has at most k choices from each configuration, then Deterministic TM accepts in $O(k^n)$ moves
  - *Polynomial time on NDTM but exponential time on NDTM*

The classes P and NP

- A problem is in class NP if there is a polynomial time non-deterministic algorithm to solve the problem
- A problem is in class P if there is a polynomial time deterministic algorithm to solve the problem
- P = NP ? Open problem….(our simulation suggests not equal)
- NP-Complete: A problem is NPC if it can be reduced to the SAT problem….
  - If this problem can be solved then all problems in NPC can be solved in polynomial deterministic time

Non-deterministic Algorithms: Example

- Non-determinism is a powerful expressive tool to design solutions
  - However, implementation on computer requires deterministic algorithms

- **Example**: *Graph Coloring Problem*
  - Known to be NP-complete (see previous page).
  - How can we solve it in polynomial time using a non-deterministic algorithm

- *Graph Coloring*: Given a graph $G=(V,E)$, find the minimum number of colors k such that no two adjacent (connected) vertices have the same color
  - Recall: definition and properties from Discrete 2 !!

- We outline a simple polynomial time non-deterministic algorithm (there are better solutions, but still NP)

Non-deterministic Algorithm for Graph Coloring (1)

- First, design `Function Check_Coloring(G,C):` given a mapping of colors to vertices, check if it is a valid coloring of G.
- A coloring C assigns one of k colors to each vertex,

$$C: \{1,..,k\} \rightarrow V = \{v_1, v_2 ..., v_n\}$$

- `Function Check-Coloring(G,C)`
  - ➢ `Input is G=(V,E) and a coloring C`
  - `For each vertex` v_i `check if the coloring is valid`
    - `Check if` $C(v_i) <> C(v_j)$ `for all` v_j `where` (v_i, v_j) `in E (edge b/w` v_i `and` v_j`)`
    - `If not valid then return "No"`
  - `Return "Yes"`

- This is a deterministic algo…takes $O(n^2)$
  - For each of the n vertices v_i , check all edges (bounded by $|V| = n$)

Non-deterministic Algo for Coloring – (2)

- Next step generate a coloring C using k colors, starting with $k=1$
- How to represent a coloring: a string of length n where each symbol is from the set $\{1,2,...k\}$!
  - Ex: For $n=7$ and $k=4$ the string **2134142** is a coloring (may not be valid)
  - $C(v_1)= 2$, $C(v_2)= 1$, $C(v_3)= 3$, $C(v_4)= 4$, $C(v_5)= 1$, $C(v_6)= 4$, $C(v_7)= 2$
- Given n (number of vertices in graph) and k (number of colors), we can generate all strings of length n – i.e., all n digits base k numbers
- How ?....discussion of algorithm from lecture
  - Generate in lexicographic order (or smallest numeric value first)
- For each such coloring (n digit base k number) branch non-deterministically and call function `Check_Coloring`

Generating Sequences/Enumeration

- Recall from Lab review of countable sets, enumeration….
- **Canonical Ordering**: list words in order of size with words of same size in numerical (dictionary) order
- How to generate (i.e., enumerate) strings (numbers?) of increasing length of radix/base k, alphabet $\{1,2,\ldots k\}$
- Define: w_i is set of strings of length i
  - w_0 = { empty string }
  - /* generate strings of length $i+1$ */
  - for each string w_i of length i /* $i=0$ to n. */

    for (radix) $j=1$ *to* k

    print $j.w_i$ /* concatenation */

Non-deterministic Algo for Coloring – (3)

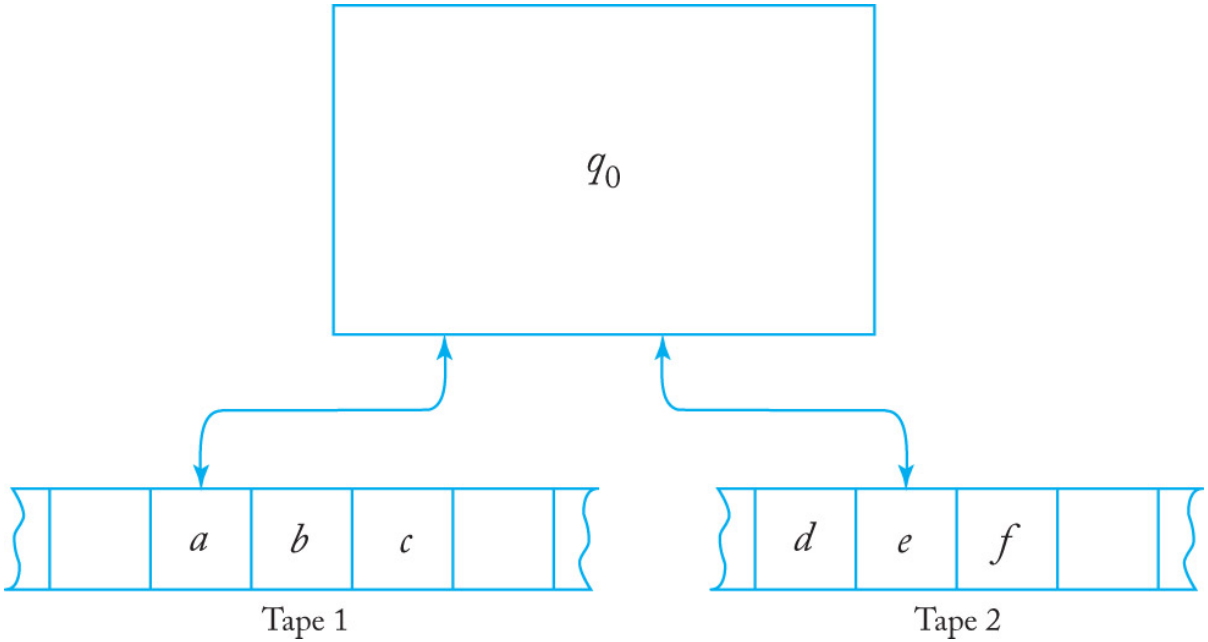
1. *DONE*=**False** *k*=1
2. while (not *DONE*) {
   1. Generate all colorings C_j of length *n* using *k* colors(/symbols)
   2. For each coloring C_j simultaneously (non-deterministically) start function Check_Color(C_j,*G*)
   - If any of these return "Yes" then *DONE*=**True** else *k++* }
3. Return *k* (minimum number of colors needed)

- Note that the algorithm will eventually halt – since we know that *k*=*n* is a valid coloring (i.e., each vertex colored with its own color)
- For *n* and *k*, there're k^n different colorings we have to check
  - One concurrent branch in a NDTM…but no concurrent branches in DTM!!

Multitape Turing Machines

- a *multitape Turing machine* has several tapes, each with its own independent read-write head
- A sample transition rule for a two-tape machine must consider the current symbols on both tapes:

$$\delta(q_0, a, e) = (q_1, x, y, L, R)$$



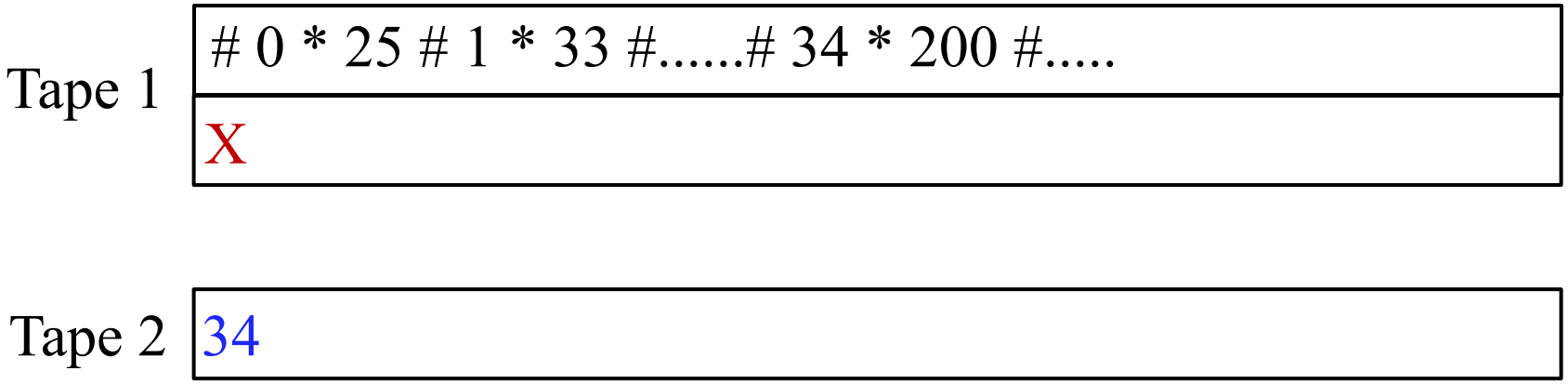
Name-Value (key-value) Stores

- Name-Value or Key-value is a pair (n_i, v_i)
  - n_i is name/key/address
  - v_i is value associated with the name/key, i.e., value at that address
- Given key n_k find the value v_k
- Why bother with this…..
  1. Data(base) storage….from DB course a NoSQL DB model of Key-Value Databases
  2. model of Computer memory…address of a location and content at that location!

Key (or Address)	Value (content)
0	25
…	…
34	200
…	…

Simulating a Name-Value (key-value) Store by a 2-tape TM

- Key-value is a pair (n*v); use separator symbols # between pairs
- The TM uses one of several tapes to hold an arbitrarily large sequence of name-value pairs in the format ***#name*value#…***
- Mark (X), using a second track, the left end of the sequence.
- A second tape can hold a name whose value we want to look up.



Find value at “address” 34

Lookup

- Starting at the left end of the store (Tape 1), compare the lookup name with each name in the store.
  - Search for #n_k*
  - Ex: search for *#34**
- When we find a match, take what follows between the * and the next # as the value.
  - #n_k* $\boldsymbol{v_k}$ #
  - Ex: 200 is the value with n=34
- Insert and Delete operations can be similarly supported.

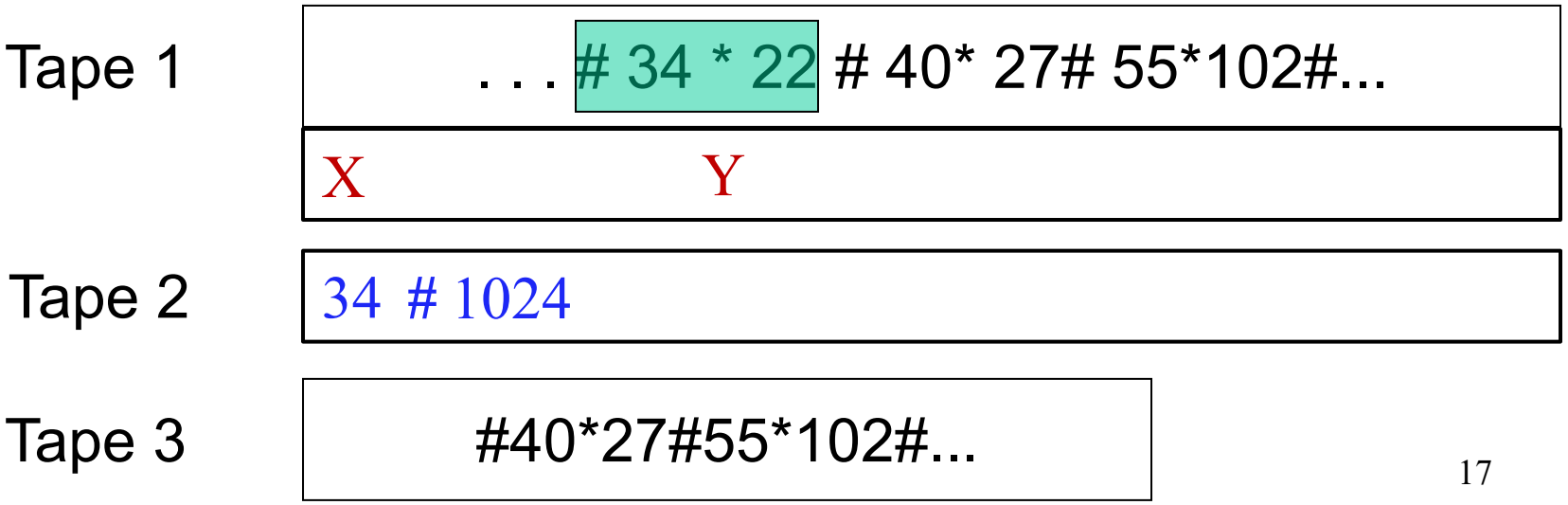
Insertion

- Suppose we want to insert name-value pair *(n, v),* or replace the current value associated with name *n* by *v*.
- Perform lookup for name *n*.
- If not found, add *n***v*# at the end of the store (Tape 1)
- If we find #*n***v'*#, we need to replace *v'* by *v*.
  - If *v* is shorter than *v'*, you can leave blanks to fill out the replacement.
  - **But** if *v* is longer than *v',* you need to make room.
    - *Use the shifting over "subroutine"*

Insertion – (2): Overwriting & Shifting Right

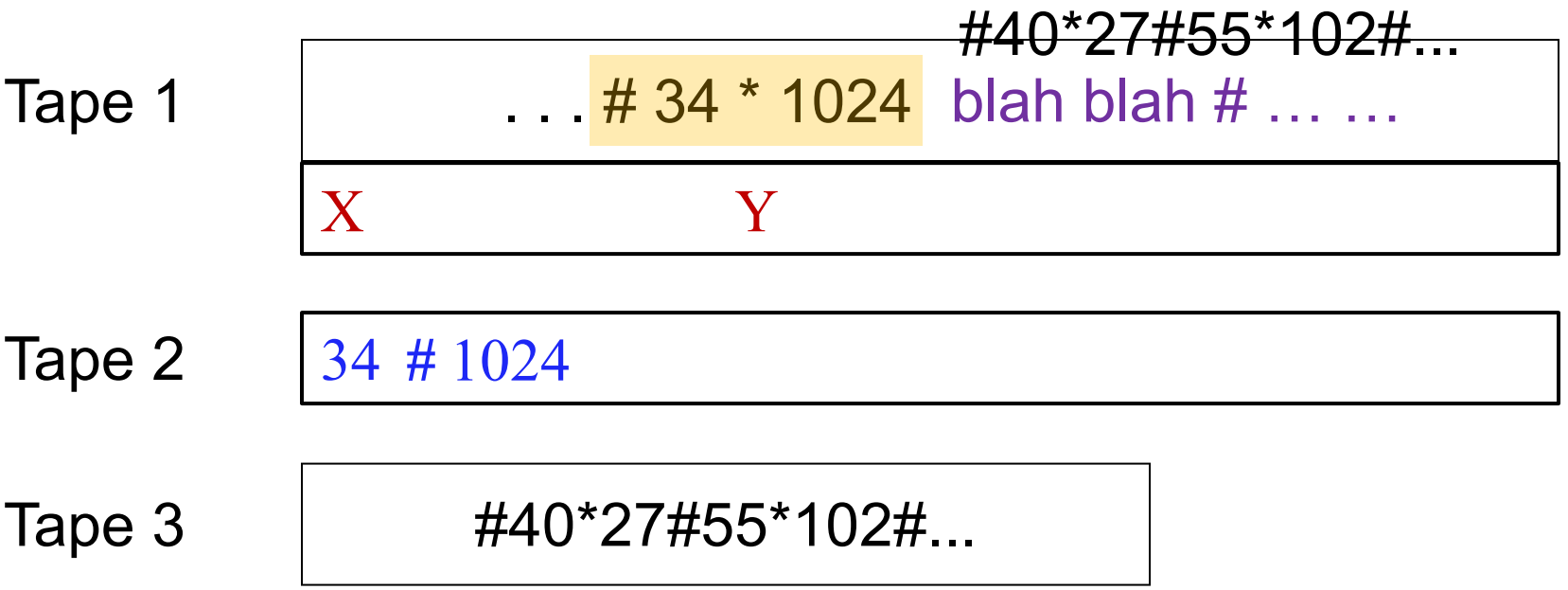
1. Search for *n*.
2. Mark the position of the * to the left of *v'*.
3. Use a third tape: copy everything from the first tape to the right of *v'* to Tape 3.

Example: n=34, v' =22
Replace v'=22 by v=1024



Insertion – (3): Overwriting & Shifting Right

4. On the first tape, write *v* just to the left of that star.
5. Copy from the third tape to the first, leaving enough room for *v*.



Questions on HW8 ?…Discussions

- Reminder: Review the "math review" notes/lab discussion
  - Lectures this week will refer to concepts of
    - Enumeration (canonical ordering)…
    - Diagonalization
    - Encodings…