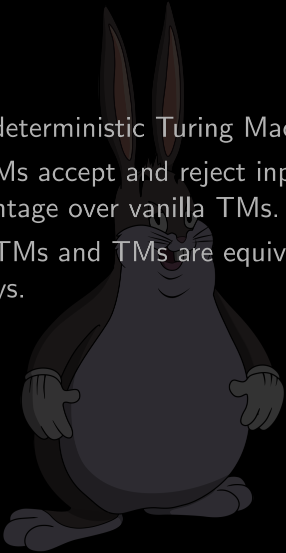
CSC363 Tutorial #6

NTMs. Guess the meaning of this acronym.

March 2, 2022

Learning objectives this tutorial

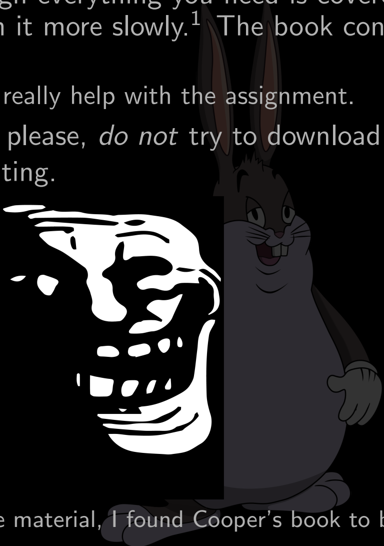


- Learn the formal definition of a Nondeterministic Turing Machine.
- Understand how Nondeterministic TMs accept and reject inputs, and why this gives NTMs an unfair advantage over vanilla TMs.
- Understand how, in the end, both NTMs and TMs are equivalent in some ways, but different in other ways.

Assignment tips

- Come to office hours! We may be able to drop some hints there.
- Read Cooper's book! Although everything you need is covered in the slides, the book goes through it more slowly.[1] The book contains many more examples.
  - Specifically, chapter 7 will really help with the assignment.

  If you do not have the book, please, *do not* try to download this illegally, such as through pirating.



[1] Personally, when learning the course material, I found Cooper's book to be extremely helpful.

(Deterministic) Turing Machines

Task: Fill in . . . .

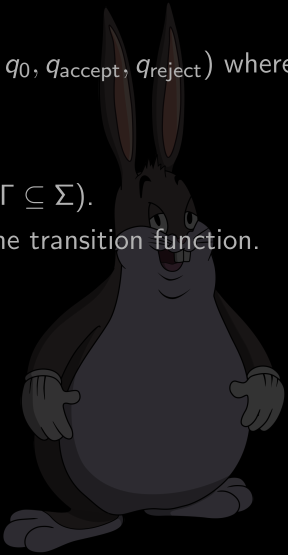
A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q is . . .
- Σ is . . .
- Γ is . . .
- $\delta : \ldots \to \ldots$ is the transition function.
- q_0 is . . .
- q_{accept} is . . .
- q_{reject} is . . .

(Deterministic) Turing Machines

A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q is the set of states.
- Σ is the *input alphabet*.
- Γ is the *tape alphabet* (and satisfies $\Gamma \subseteq \Sigma$).
- $\delta : (Q \times \Gamma) \to (Q \times \Gamma \times \{L, R\})$ is the transition function.
- q_0 is the starting state.
- q_{accept} is the accept state.
- q_{reject} is the reject state.

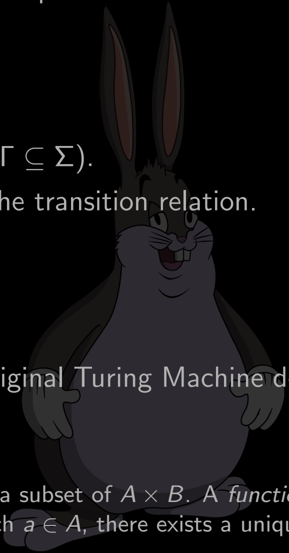


(Nondeterministic) Turing Machines

A Nondeterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q is the set of states.
- Σ is the *input alphabet*.
- Γ is the *tape alphabet* (and satisfies $\Gamma \subseteq \Sigma$).
- $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ is the transition relation.
- q_0 is the starting state.
- q_{accept} is the accept state.
- q_{reject} is the reject state.

Task: What changed, compared to the original Turing Machine definition?

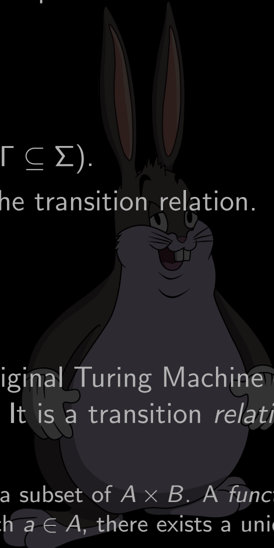


[2] Recall that a *relation* R between A and B is a subset of $A \times B$. A *function* from A to B is a relation between A and B where for each $a \in A$, there exists a unique $b \in B$ such that (a, b) is in the relation.

(Nondeterministic) Turing Machines

A Nondeterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where:

- Q is the set of states.
- Σ is the *input alphabet*.
- Γ is the *tape alphabet* (and satisfies $\Gamma \subseteq \Sigma$).
- $\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ is the transition relation.
- q_0 is the starting state.
- q_{accept} is the accept state.
- q_{reject} is the reject state.



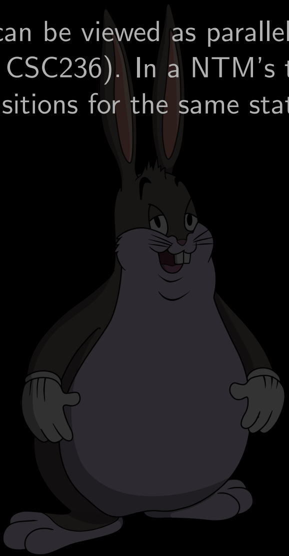
Task: What changed, compared to the original Turing Machine definition?
Ans: δ is no longer a transition *function*! It is a transition *relation*.[2]

---

[2] Recall that a *relation* R between A and B is a subset of $A \times B$. A *function* from A to B is a relation between A and B where for each $a \in A$, there exists a unique $b \in B$ such that (a, b) is in the relation.

(Nondeterministic) Turing Machines

The difference between a NTM and TM can be viewed as parallel to the difference between a DFA and NFA (from CSC236). In a NTM's transition table, we may have multiple different transitions for the same state and same character.

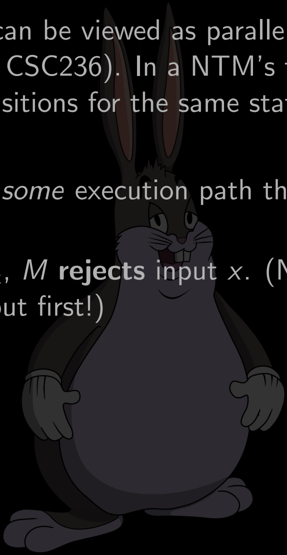


(Nondeterministic) Turing Machines

The difference between a NTM and TM can be viewed as parallel to the difference between a DFA and NFA (from CSC236). In a NTM's transition table, we may have multiple different transitions for the same state and same character.

A NTM M **accepts** input x when M has *some* execution path that ends in q_{accept}. Otherwise:

- If *some* execution path ends in q_{reject}, M **rejects** input x. (Note: Check that M doesn't accept the input first!)
- Otherwise, it loops.



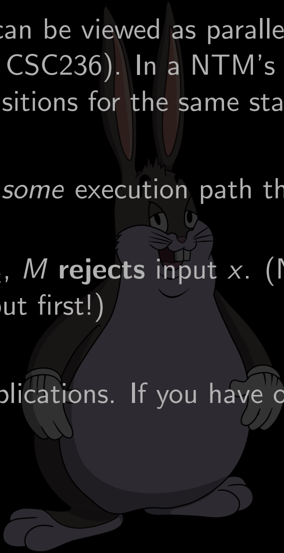
(Nondeterministic) Turing Machines

The difference between a NTM and TM can be viewed as parallel to the difference between a DFA and NFA (from CSC236). In a NTM's transition table, we may have multiple different transitions for the same state and same character.

A NTM M **accepts** input x when M has *some* execution path that ends in q_{accept}. Otherwise:

- If *some* execution path ends in q_{reject}, M **rejects** input x. (Note: Check that M doesn't accept the input first!)
- Otherwise, it loops.

In some sense, think of NTMs like job applications. If you have one job offer, you're good! Otherwise :(

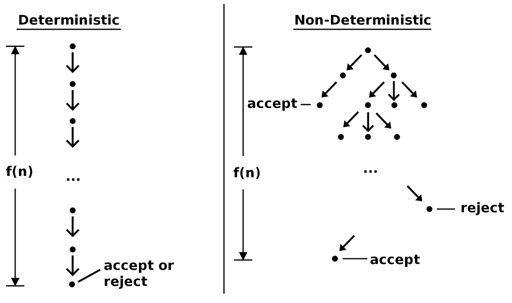


(Nondeterministic) Turing Machines

A NTM M **accepts** input x when M has *some* execution path that ends in q_{accept}. Otherwise:

- If *some* execution path ends in q_{reject}, M **rejects** input x. (Note: Check that M doesn't accept the input first!)
- Otherwise, it loops.

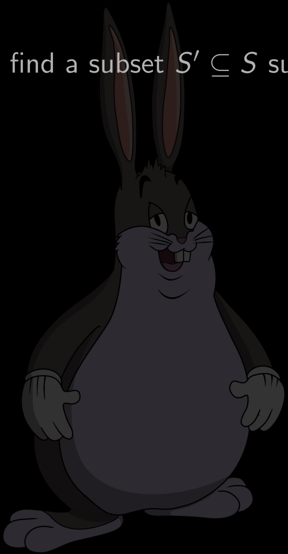
In some sense, think of NTMs like job applications. If you have one job offer, you're good! Otherwise :(



NTMs have multiple different "execution paths".

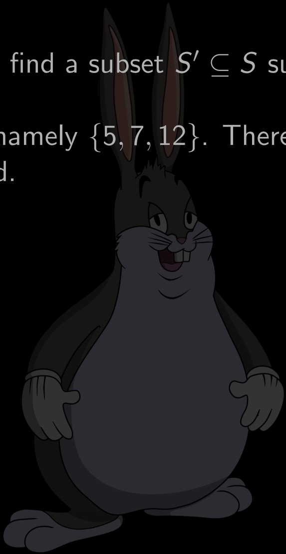
Subset Sum

Task: Let $S = \{5, 7, 11, 12, 14\}$. Can you find a subset $S' \subseteq S$ such that S' sums to 24? What about 27?



Subset Sum

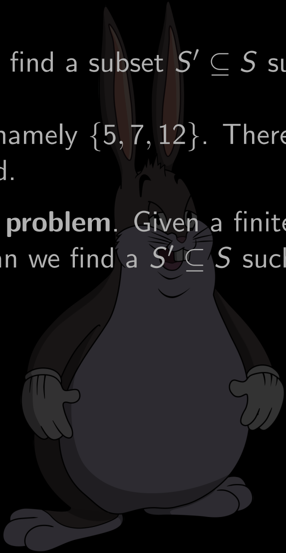
Task: Let $S = \{5, 7, 11, 12, 14\}$. Can you find a subset $S' \subseteq S$ such that S' sums to 24? What about 27?
Ans: There is a subset that sums to 24, namely $\{5, 7, 12\}$. There are no subsets that sum to 27, on the other hand.



Subset Sum

Task: Let $S = \{5, 7, 11, 12, 14\}$. Can you find a subset $S' \subseteq S$ such that S' sums to 24? What about 27?
Ans: There is a subset that sums to 24, namely $\{5, 7, 12\}$. There are no subsets that sum to 27, on the other hand.

This is a specific case of the **subset sum problem**. Given a finite set of natural numbers S and a *target* $t \in \mathbb{N}$, can we find a $S' \subseteq S$ such that S' sums to t?

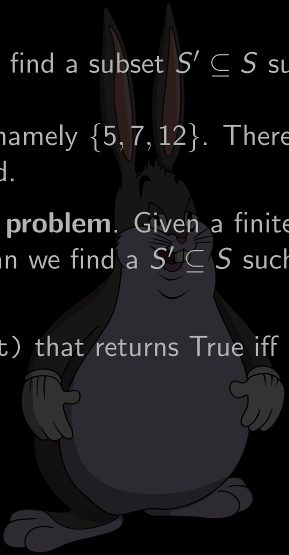


Subset Sum

Task: Let $S = \{5, 7, 11, 12, 14\}$. Can you find a subset $S' \subseteq S$ such that S' sums to 24? What about 27?
Ans: There is a subset that sums to 24, namely $\{5, 7, 12\}$. There are no subsets that sum to 27, on the other hand.

This is a specific case of the **subset sum problem**. Given a finite set of natural numbers S and a *target* $t \in \mathbb{N}$, can we find a $S' \subseteq S$ such that S' sums to t?

Task: Write a function `subset_sum(S, t)` that returns True iff S has a subset that sums to t, using pseudocode.

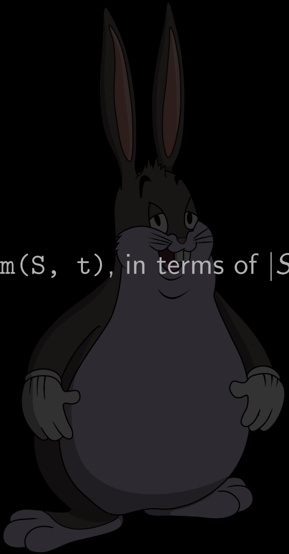


Subset Sum

Task: Write a function `subset_sum(S, t)` that returns True iff S has a subset that sums to t, using pseudocode.

Ans:

```
def subset_sum(S, t):
  for every subset S' of S:
    if S' sums to t:
      return True
  return False
```



Task: What is the runtime of `subset_sum(S, t)`, in terms of $|S|$ (the size of S)?

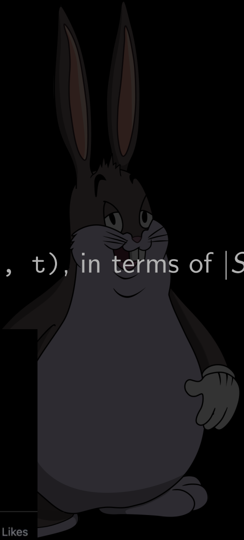
Subset Sum

Task: Write a function `subset_sum(S, t)` that returns True iff S has a subset that sums to t, using pseudocode.

Ans:

```
def subset_sum(S, t):
  for every subset S’ of S:
    if S’ sums to t:
      return True
  return False
```

Task: What is the runtime of `subset_sum(S, t)`, in terms of $|S|$ (the size of S)? **Ans:** It is $O(2^{|S|})$.



Subset Sum

Task: Write a function `subset_sum(S, t)` that returns True iff S has a subset that sums to t, using pseudocode. Make sure to do it in polynomial time!

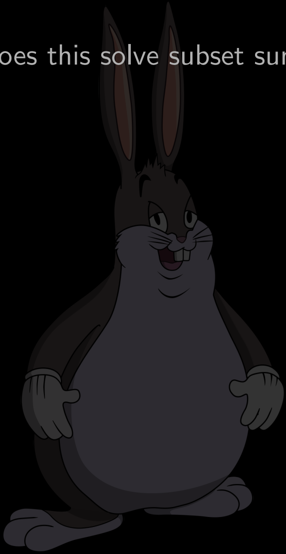
Ans:

(we don't know if it's possible or not)

Subset Sum

Wait! What about the following code? Does this solve subset sum?

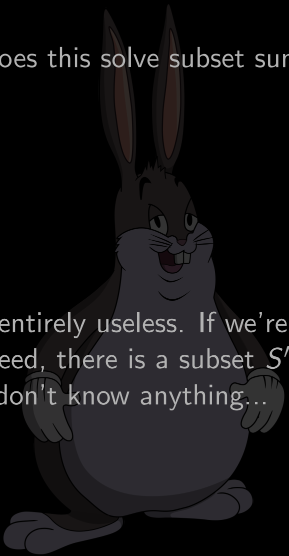
```
def subset_sum_2(S, t):
  choose a random subset S’ of S
  if S’ sums to t:
    return True
  return False
```



Subset Sum

Wait! What about the following code? Does this solve subset sum?

```
def subset_sum_2(S, t):
  choose a random subset S' of S
  if S' sums to t:
    return True
  return False
```



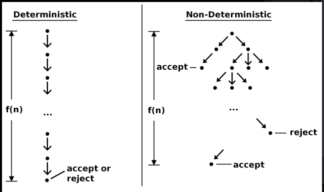
Of course not! But `subset_sum_2` is not entirely useless. If we're lucky and `subset_sum_2` returns True, then indeed, there is a subset S' that sums to t. If it returns False instead, we don't know anything...

Subset Sum

This process of choosing a “random” S' may be implemented in a NTM. The NTM will *simultaneously choose all* particular subsets S', and accept if and only if one execution path returns True.

```
def subset_sum_ntm(S, t):
  choose a particular subset S’ of S
  if S’ sums to t:
    return True
  return False
```

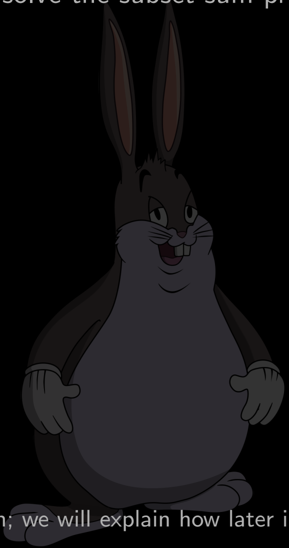
And it runs in linear time (in terms of the maximum execution length)! Unfortunately we can’t implement this in real life... :(



$f(n)$ is the execution length.

Subset Sum

Conclusion: The subset sum problem can be solved in linear time by a NTM. However, we don't know if we can solve the subset sum problem with a TM.[3]

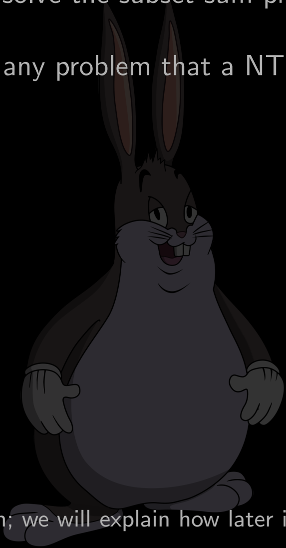


[3] This amounts to solving the P = NP problem; we will explain how later in this course.

Subset Sum

Conclusion: The subset sum problem can be solved in linear time by a NTM. However, we don't know if we can solve the subset sum problem with a TM.[3]

Question: Disregarding runtime, is there any problem that a NTM can solve, but a TM can't solve?



[3] This amounts to solving the P = NP problem; we will explain how later in this course.

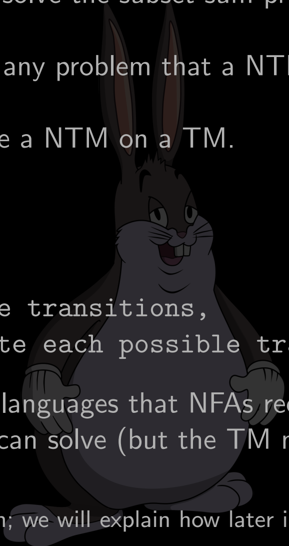
Subset Sum

Conclusion: The subset sum problem can be solved in linear time by a NTM. However, we don't know if we can solve the subset sum problem with a TM.[3]

Question: Disregarding runtime, is there any problem that a NTM can solve, but a TM can't solve?

Ans: No! This is because we can simulate a NTM on a TM.

```
def simulate_NTM(ntm, input):
  while True:
    execute ntm(input) one step.
    if there are multiple possible transitions,
    spawn a thread here to simulate each possible transition
```



In the same way DFAs can recognize any languages that NFAs recognize, TMs can solve any problem that a NTM can solve (but the TM may be much slower).

[3] This amounts to solving the P = NP problem; we will explain how later in this course.