CSC363 Tutorial #7

Runtime of Turing Machines

March 8, 2023

Things covered in this tutorial

- ⋆ How can I recognize the language $\{0^n 1^n : n \in \mathbb{N}\}$?
- ⋆ How can I recognize the language $\{0^n 1^n : n \in \mathbb{N}\}$, but faster?
- ⋆ What is "pseudo-polynomial time", and why do I need to be be aware of this?

You are still enrolled in this course.

We need to go back to the basics!

Task: Construct a decider for the language $\{0^n 1^n : n \in \mathbb{N}\}$. *What is the runtime?*

We need to go back to the basics!

Task: Construct a decider for the language $\{0^n 1^n : n \in \mathbb{N}\}$. *What is the runtime?*

Ans:

```
is_in_0n1n(x):
  if len(x) is odd:
    reject
  for i in range(len(x)/2): # not including len(x)/2
    if x[i] != 0 or x[len(x)/2 + i - 1] != 1:
      reject
  accept
```

Since there is only one for loop, the runtime is $O(n)$.

We need to go back to the basics!

Task: Construct a decider for the language $\{0^n 1^n : n \in \mathbb{N}\}$. *What is the runtime?*

Ans:

```
is_in_0n1n(x):
  if len(x) is odd:
    reject
  for i in range(len(x)/2): # not including len(x)/2
    if x[i] != 0 or x[len(x)/2 + i - 1] != 1:
      reject
  accept
```

~~Since there is only one for loop, the runtime is $O(n)$.~~

We need to be more careful regarding what we mean by “runtime”…

Definition of "runtime"

Let M be a Turing machine, and $f : \mathbb{N} \to \mathbb{N}$ a function. We say that M is "$O(f(n))$-time" if: given any input x of **size n**, $M(x)$ halts in $O(f(n))$ steps or less.

Definition of "runtime"

Let M be a Turing machine, and $f : \mathbb{N} \to \mathbb{N}$ a function. We say that M is "$O(f(n))$-time" if: given any input x of **size n**, $M(x)$ halts in $O(f(n))$ steps or less.

```
is_in_0n1n(x):
  if len(x) is odd:
    reject
  for i in range(len(x)/2): # not including len(x)/2
    if x[i] != 0 or x[len(x)/2 + i - 1] != 1:
      reject
  accept
```

Church-Turing implies that there is a Turing machine M that computes `is_in_0n1n()`.

Definition of "runtime"

Let M be a Turing machine, and $f : \mathbb{N} \to \mathbb{N}$ a function. We say that M is "$O(f(n))$-time" if: given any input x of **size n**, $M(x)$ halts in $O(f(n))$ steps or less.

```
is_in_0n1n(x):
  if len(x) is odd:
    reject
  for i in range(len(x)/2): # not including len(x)/2
    if x[i] != 0 or x[len(x)/2 + i - 1] != 1:
      reject
  accept
```

Church-Turing implies that there is a Turing machine M that computes `is_in_0n1n()`. Church-Turing does not guarantee that M has the same runtime as `is_in_0n1n()`.

0^n1^n **Turing Machine**

At a lower level, how would one construct a Turing machine that decides $\{0^n1^n\}$?

0^n1^n **Turing Machine**

At a lower level, how would one construct a Turing machine that decides $\{0^n1^n\}$?

Task: Try to construct a Turing machine that decides $\{0^n1^n\}$ in $O(n^2)$-time.

Please don't look ahead in my slides!

0^n1^n **Turing Machine**

How to decide whether something is in 0^n1^n, using a Turing machine:

1. Attempt to “cross out” a 0 at the left end of the string.
2. Move to the right end of the string.
3. Attempt to “cross out” a 1 at the right end of the string.
4. Move to the left end of the string.
5. Go to step 1.

0^n1^n Turing Machine

How to decide whether something is in 0^n1^n, using a Turing machine:

1. Attempt to "cross out" a 0 at the left end of the string.
2. Move to the right end of the string.
3. Attempt to "cross out" a 1 at the right end of the string.
4. Move to the left end of the string.
5. Go to step 1.

$O(n^2)$ steps!

$0^n 1^n$ Turing Machine, but faster!

We can actually decide $\{0^n 1^n\}$ in $O(n \log n)$ steps!

0^n1^n **Turing Machine, but faster!**

We can actually decide $\{0^n1^n\}$ in $O(n \log n)$ steps!

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following as long as there is both a 0 and a 1 on the tape:
   2.1. Scan across the tape, and reject if the total number of 0s and 1s remaining is odd.
   2.2. Scan again across the tape, crossing off every other 0, and crossing off every other 1.
3. If the tape doesn't have any 0s or 1s, accept. Else, reject.

0^n1^n Turing Machine, but faster!

We can actually decide $\{0^n1^n\}$ in $O(n \log n)$ steps!

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following as long as there is both a 0 and a 1 on the tape:
   2.1. Scan across the tape, and reject if the total number of 0s and 1s remaining is odd.
   2.2. Scan again across the tape, crossing off every other 0, and crossing off every other 1.
3. If the tape doesn't have any 0s or 1s, accept. Else, reject.

Click for counter stirik free cheat $2000 skin (2003) sans gaming poggers!![240p] chikin farm recipe (FDA approvid) linkin chester alive???? [CLICK TO FIND OUT] nokia

Polynomial time Turing machines

Rejoice, CSC373 enjoyers!

Polynomial time Turing machines

Rejoice, CSC373 enjoyers!

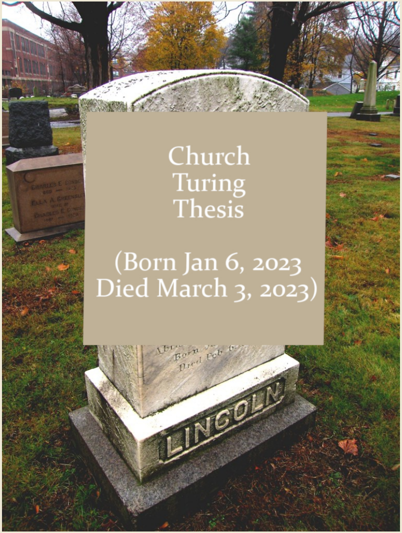
A Turing machine is polynomial-time if it runs in $O(n^k)$-time for some $k \in \mathbb{N}$.

Examples of polynomial runtimes: $O(n)$, $O(1)$, $O(n^5)$, $O(n^{999999})$, $O(n \log n)$, $\ldots$

Examples of non-polynomial runtimes: $O(2^n)$, $O(n!)$, $O(n^n)$, $O(\mathrm{Ackermann}(n, n))$, $\ldots$

Polynomial time Turing machines

We could decide $\{0^n1^n\}$ in $O(n)$ time on a modern computer, but we needed $O(n^2)$ time in a Turing machine implementation!



Church Turing Thesis 2: Electric Boogaloo

Let L be any language.

Church Turing Thesis 2: Electric Boogaloo

Let L be any language.

L is decidable by a polynomial-time Turing machine if and only if L is decidable by a computer in polynomial time.

Church Turing Thesis 2: Electric Boogaloo

Let L be any language.

L is decidable by a polynomial-time Turing machine if and only if L is decidable by a computer in polynomial time.

Note: not necessarily the same O-bound! $\{0^n 1^n\}$ is decidable in $O(n)$ time on a computer, but $O(n \log n)$ time on a Turing machine. Either way, both $O(n)$ and $O(n \log n)$ are polynomial runtimes.

Pseudo-polynomial runtime

(Please do not look this up on Wikipedia!)

Pseudo-polynomial runtime

(Please do not look this up on Wikipedia!)

Consider the following code to decide whether a number is prime:

```
is_prime(x):
  if x == 1:
    return False
  for i in range(2, x): # not including x
    if i divides x:
      return False
  return True
```

Question: Why is this code not *polynomial time*?

Hint: Let M be a Turing machine, and $f : \mathbb{N} \to \mathbb{N}$ a function. We say that M is "$O(f(n))$-time" if: given any input x of **size** n, $M(x)$ halts in $O(f(n))$ steps or less.

Pseudo-polynomial runtime

The set of prime numbers is actually decidable in polynomial time!

Pseudo-polynomial runtime

The set of prime numbers is actually decidable in polynomial time!

The AKS Primality Test can determine if an input number is prime in $O(n^{12})$ time (where n is the number of digits in the input, not the numerical value).

Pseudo-polynomial runtime

The set of prime numbers is actually decidable in polynomial time!

The AKS Primality Test can determine if an input number is prime in $O(n^{12})$ time (where n is the number of digits in the input, not the numerical value).

Proof that it works: