Math 251, Wed 29-Sep-2021 -- Wed 29-Sep-2021
Discrete Mathematics
Fall 2021

Wednesday, September 29th 2021

Wk 5, We
Topic:: Hilbert Hotel
Topic:: Algorithms

Terms: finite set, countably infinite set

Cantor's proof of an uncountable infinity

Hilbert's Grand Hotel

A set is countable if
- it is finite, or
- it can be put in 1-1 correspondence with $\mathbb{N}$.

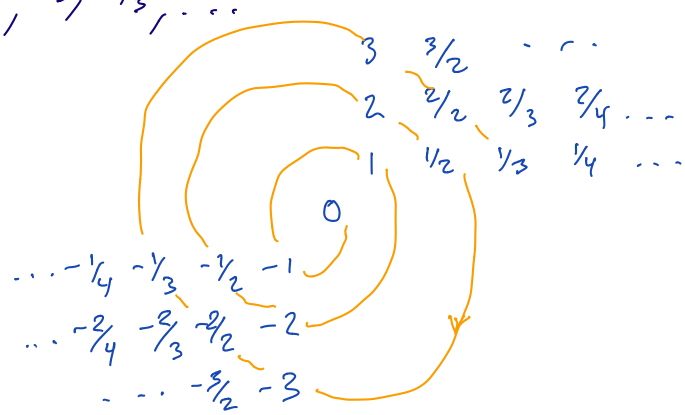
countably infinite

Examples:

evens: $2, 4, 6, 8, 10, \ldots$

$\mathbb{Z}$: $0, 1, -1, 2, -2, 3, -3, 4, -4, \ldots$

$\mathbb{Q}$: $0, 1, -1, 2, 1/2, -2, -1/2, 3, 1/3, -3, -1/3, \ldots$



Cantor claimed: $[0, 1]$ are not countably infinite proof by contradiction :

Suppose $[0,1]$ is countable.

Then it can be enumerated

$$\begin{array}{l} [0,1] \\ \hline a_1 = 0.a_{11}\,a_{12}\,a_{13}\,a_{14}\cdots \\ a_2 = 0.a_{21}\,a_{22}\,a_{23}\,a_{24}\cdots \\ a_3 = 0.a_{31}\,a_{32}\,a_{33}\,a_{34}\cdots \\ a_4 = 0.a_{41}\,a_{42}\,a_{43}\,a_{44}\cdots \\ \vdots \qquad\quad \vdots \end{array}$$

Now construct a number $b = 0.b_1 b_2 b_3 b_4 \cdots$

At each stage, make $b_i = \begin{cases} 3 & \text{if } a_{ii} = 2 \\ 2 & \text{if } a_{ii} \neq 2 \end{cases}$

So we have a number $b \in [0,1]$ but doesn't match any of those in enumerated list.

This ~~is~~ contradicts our original assumption that $[0,1]$ is countably infinite.

Player A

1	X	O	X	O	X	O
2	X	X	O	X	O	O
3	X	X	X	X	O	O
4	X	X	O	O	X	X
5	X	X	O	O	O	O
6	X	X	O	O	X	O

Player B

X	X	O	O	X	

Player A

1	O	X	O	X	O	X
2	X	X	O	X	X	O
3	X	O	X	X	O	X
4	X	O	O	X	O	X
5	X	O	O	O	X	O
6	X	O	O	O	O	X

Player B

X	O	O	O	O	O

Player A

1						
2						
3						
4						
5						
6						

Player B

Player A

1						
2						
3						
4						
5						
6						

Player B

Algorithms

Properties

- specified set of inputs (domain)
- every input produces output from codomain
- definiteness: clear process to follow
- correctness: accurately finds correct output for each input
- finiteness: desired output is produced after finite number of steps
- effectiveness: possible to do each step in finite amount of time
- generality: applicable to all problems of desired form

Note: Not all problems are solvable in the sense of having an algorithm as described above.

Example: Halting problem. At least one problem is unsolvable.

What is sought in the halting problem: An algorithm that can decide, given any computer program and set of inputs, whether the program halts in finitely many steps. Suppose such a procedure exists, and write

$$H\colon \{\text{programs}\} \times \{\text{inputs}\} \to \{\text{"halts"}, \text{"DNH"}\}.$$

Note: $H(P,P)$ is defined and will have either the value "halts" or "DNH". Define a procedure K which takes programs P as inputs, and

$K(P)$

$H(P, I)$

loops forever ("DNH") if $H(P,P) =$ "halts".
"halts" if $H(P,P) =$ "DNH".

Notice that $H(K,K)$ can produce either of two values, but both contradict the behavior of $K(K)$.

like the liar's paradox

Specific algorithms

Binary Search

```
import numpy as np

~~Binary search~~:
def binSrch(key, inList):
  i = 1
  j = len(inList)
  while (i < j):
    m = int(np.floor( (i+j) / 2 ))
    if (key > inList[m−1]):
      i = m+1
    else:
      j = m
  if (key == inList[i−1]):
    index = i
  else:
    index = 0
  return(index−1)
```

Bubble sort

```
def bubbleSort(inList):
  n = len(inList) - 1
  for i in range(n):
    for j in range(n-i):
      if ( inList[j] > inList[j+1] ):
        temp = inList[j+1]
        inList[j+1] = inList[j]
        inList[j] = temp

  return(inList)
```

Insertion sort

```
def insertionSort(inList):
  n = len(inList)
  for j in range(1,n):
    i = 0
    while inList[j] > inList[i]:
      i += 1
    m = inList[j]
    for k in range(j-i):
      inList[j-k] = inList[j-k-1]

    inList[i] = m

  return(inList)
```