CS 101 Computer Programming and Utilization

Practice Problems

Param Rathour
https://paramrathour.github.io/CS101

Autumn Semester 2020-21

Last update: 2021-01-19 19:58:11+05:30

Disclaimer

These are **optional** problems. As these problems are pretty involving, my advice to you would be to first solve exercises given in slides and get comfortable with the course content. The taught methods will suffice to solve these problems. (You are free to use 'other' stuff but not recommended)

Good Programming Practices

- Clearly writing documentation explaining what the program does, how to use it, what quantities it takes as input, and what quantities it returns as output.
- Using appropriate variable/function names.
- Extensive internal comments explaining how the program works.
- Complete error handling with informative error messages.
  For example, if $a = b = 0$, then the $\gcd(a, b)$ routine should return the error message "$\gcd(0, 0)$ is undefined" instead of going into an infinite loop or returning a "division by zero" error.

Contents

§1. Practice Problems 1 - Introduction

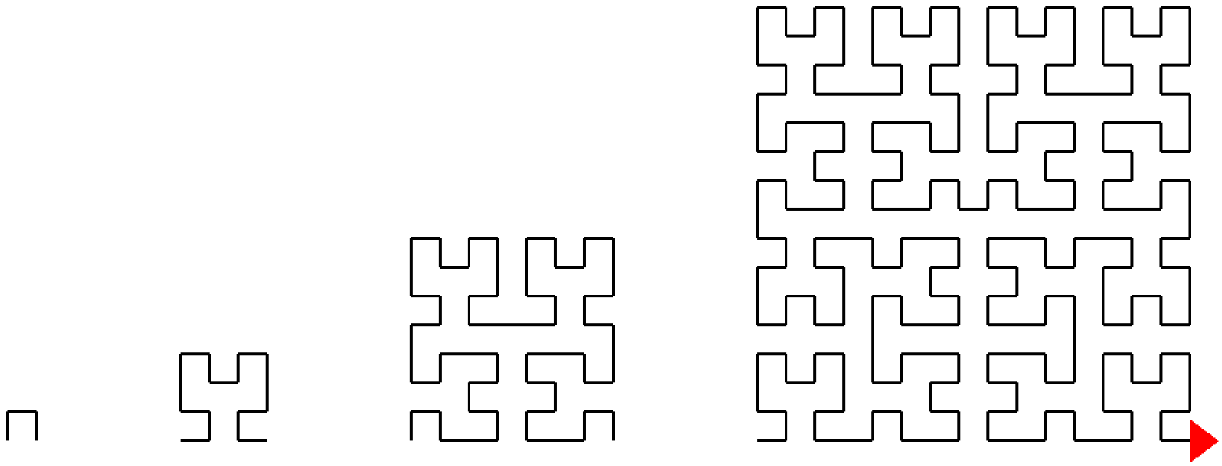


Figure 1: Hilbert Curve

1. **Problem Statement:**
   Take an integer as input and draw the corresponding iteration of this fractal using turtleSim
   You may think along these lines

   Step 1 Find a simple pattern in these iterations

   Step 2 Think how can you implement this pattern in an efficient way (here think in the number of lines of code you have to write. **Word of caution**: this is just one of the possible definitions of efficient code)

   Step 3 Do you think that you need something that will implement/shorten your code?
   How will it look like? (it's a feature)

   Feel free to discuss your thoughts on this.

   Note. *For people comfortable with the basics of C++, this shouldn't be difficult. You may try this*

Fun Videos

Hilbert's Curve: Is infinite math useful?
Recursive PowerPoint Presentations [Gone Fractal!]

Book Chapters for Graphics

Additional chapters of the book on Simplecpp graphics demonstrating its power
(It is just a list, you are not expected to understand/study things, CS101 is for a reason :P)

Chapter 1 Turtle graphics

Chapter 5 Coordinate based graphics, shapes besides turtles

Chapter 15.2.3 Polygons

Chapter 19 Gravitational simulation

Chapter 20 Events, Frames, Snake game

Chapter 24.2 Layout of math formulae

Chapter 26 Composite class

Chapter 28 Airport simulation

§2. Practice Problems 2 - Loops

1. You probably heard about Fibonacci Numbers!

   The Fibonacci numbers are the numbers in the integer sequence: (defined by the recurrence relation)

   $$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \quad n \in \mathbb{Z} \quad \text{(They can be extended to negative numbers)} \end{aligned} \tag{1}$$

   For any integer n, the sequence of Fibonacci numbers F_i taken modulo n is periodic.

   The Pisano period, denoted $\pi(n)$, is the length of the period of this sequence.

   For example, the sequence of Fibonacci numbers modulo 3 begins:

   $$0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, \ldots \text{(A082115)}$$

   This sequence has period 8, so $\pi(3) = 8$. (Basically, the remainder when these numbers are divided by n is a repeating sequence. You have to find the length of sequence)

   Problem Statement:

   (a) Find Pisano period of n numbers $k_1, k_2, \ldots, k_n$

   Input Format
   n
   $k_1, k_2, \ldots, k_n$
   Output Format
   $\pi(k_i)$ (each on a newline)
   Sample Input
   3
   3 10 25
   Sample Output
   8
   60
   100

   (b) For n numbers $k_1, k_2, \ldots, k_n$, find $\max(\pi(i))$ for $i = 1, 2, \ldots, k$ and corresponding i
   If there are 2 (or more) such i's, output smallest of them

   Input Format
   n
   $k_1, k_2, \ldots, k_n$
   Output Format
   k $\pi(k)$ (each pair on a newline)
   Here k is smallest possible integer satisfying $\pi(k) = \max(\pi(i))$ for possible i
   Sample Input
   5
   20 40 60 80 100
   Sample Output
   10 60
   30 120
   50 300
   50 300
   98 336

2.

$$\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{k!}{(2k+1)!!} = \sum_{k=0}^{\infty} \frac{2^k k!^2}{(2k+1)!} \tag{2}$$

Note. *$n!!$ is called double factorial. $n!! \neq (n!)!$.*

Problem Statement:
Calculate π using first $k_i + 1$ terms [1] of Equation (2) for n different natural numbers $k_1, k_2, \ldots, k_n$.
Give your answers correct to 10 decimal places

Input Format
n
$k_1, k_2, \ldots, k_n$
Output Format
Calculated π for k_i (each on a newline)
Sample Input
3
10 20 30
Sample Output
3.1411060216
3.1415922987
3.1415926533

3. **Simpson's Rule**: a method for numerical integration

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 4f(x_{n-1}) + f(x_n)\right) \tag{3}$$

Note. *Simpson's rule can only be applied when an odd number of ordinates is chosen.*

Problem Statement:
Solve Equation (4) giving the answers correct to 7 decimal places (Use 101 ordinates)

$$\int_{0.5}^{1} \frac{\sin\theta}{\theta}\,\mathrm{d}\theta \tag{4}$$

Correct Answer = 0.4529756

[1] i.e calculate π till $\dfrac{k_i!}{(2k_i+1)!!}$ term

§3. Practice Problems 3 - Functions

1. Write a function that calculates the day of the week for any particular date in the past or future. Consider Gregorian calendar (AD)

   Task 1: As a programming exercise, try the naive approach:
   Starting from 1 Jan 0001 (Saturday) and calculate day after day till you reach the given date
   Use `switch-case` statement

   Task 2: Try to make more efficent algorithm (reduce completion time) than Task 1
   Implement it, and discuss your approach with me.

   Also check for invalid dates (Write another function for this)
   If dates are invalid, output **-1**

   Input Format
   n
   Followed by n dates in **Date Month Year** format
   Output Format
   Day of the Week
   Sample Input
   5
   19 2 1627
   29 2 1700
   15 4 1707
   22 12 1887
   23 6 1912
   Sample Output
   Monday
   -1
   Friday
   Thursday
   Sunday

   Note. *Use your Task 1 program to check Task 2 implementation.*

2. **Farey Sequence**
   This sequence has all rational numbers in range $[0/1 \text{ to } 1/1]$ sorted *in increasing order* such that the denominators are less than or equal to n and all numbers are in *reduced forms* i.e., 2/4 does not belong to this sequence as it can be reduced to 1/2.

   Input Format
   n
   Output Format
   Corresponding numbers in sequence in p/q format
   Sample Input
   7
   Sample Output
   0/1 1/7 1/6 1/5 1/4 2/7 1/3 2/5 3/7 1/2 4/7 3/5 2/3 5/7 3/4 4/5 5/6 6/7 1/1

   Can you find efficient solution?

Fun Video

Funny Fractions and Ford Circles

3. **Thue-Morse Sequence aka Fair Share Sequence**
Thue-Morse Sequence is the infinite binary sequence obtained by starting with 0 and successively appending the Boolean complement of the sequence obtained thus far (called prefixes of the sequence).
First few steps :

- Start with 0
- Append complement of 0, we get 01
- Append complement of 01, we get 0110
- Append complement of 0110, we get 01101001

Problem Statement:
Consider appending complement of a prefix to itself as one iteration
Define a function to take a positive integer n as input then iterate n times to print the first 2^n digits

Input Format
n
Output Format
Corresponding digits in sequence
Sample Input
6
Sample Output
0110100110010110100101100110100110010110011010010110100110010110

Again, can you find better solution?

Fun Video

The Fairest Sharing Sequence Ever

4. **Collatz Conjecture**
Consider the following operation on an arbitrary positive integer:

- If the number is even, divide it by two.
- If the number is odd, triple it and add one.

Collatz Conjecture states that no matter which positive integer we start with; we always end up with 1.

Problem Statement:
Define a function which performs this operation repeatedly on the result at each step; beginning with a given input n $(n < 10^6)$, returns the number of operations required to reach 1[2]

Input Format
Arbitrary number of testcases (each space separated)
Stop when input is negative
Output Format
Count of operations for each number (each on a newline)
Sample Input
1 3 7 9 27 871 77031 -1
Sample Output
0
7
16
19
111
178
350

[2]As of 2020, the conjecture has been checked by computer for all starting values up to $2^{68} \approx 2.95 \times 10^{20}$, so sequence from n will reach 1

§4. Practice Problems 4 - Recursion

Practice Problem 4 are inspired from the following video do watch till 6:10 to get clear understanding of recursion
5 Simple Steps for Solving Any Recursive Problem

- What's the simplest possible input?
- Play around with examples and visualize!
- Relate hard cases to simpler cases
- Generalize the pattern
- Write code by combining recursive pattern with base case

1. **Ackermann function**
   Write a recursive function $\mathrm{A}()$ that takes two inputs n and m and outputs the number $\mathrm{A}(m, n)$ where $\mathrm{A}(m, n)$ is defined as

$$\mathrm{A}(0, n) = n + 1 \tag{5}$$

$$\mathrm{A}(m + 1, 0) = \mathrm{A}(m, 1) \tag{6}$$

$$\mathrm{A}(m + 1, n + 1) = \mathrm{A}(m, \mathrm{A}(m + 1, n)) \tag{7}$$

   Problem Statement:
   For k pairs $(m_1, n_1), (m_2, n_2), \ldots, (m_k, n_k)$, find $\mathrm{A}(m, n)$ for such m, n

Input Format
k
m_i n_i (each pair on a newline)
Output Format
$\mathrm{A}(m_i, n_i)$ (each result on a newline)
Sample Input
7
0 0
0 4
1 3
2 2
3 4
4 0
4 1
Sample Output
1
5
5
7
125
13
65533

2. **GridPaths**
Write a recursive function $\mathrm{NumberOfGridPaths}()$ that takes two inputs n and m and outputs the number of unique paths from the top left corner to bottom right corner of a $n \times m$ grid.

Constraints: $n, m \geq 1$ and you can only move down or right 1 unit at a time.
Examples given in Figure 2

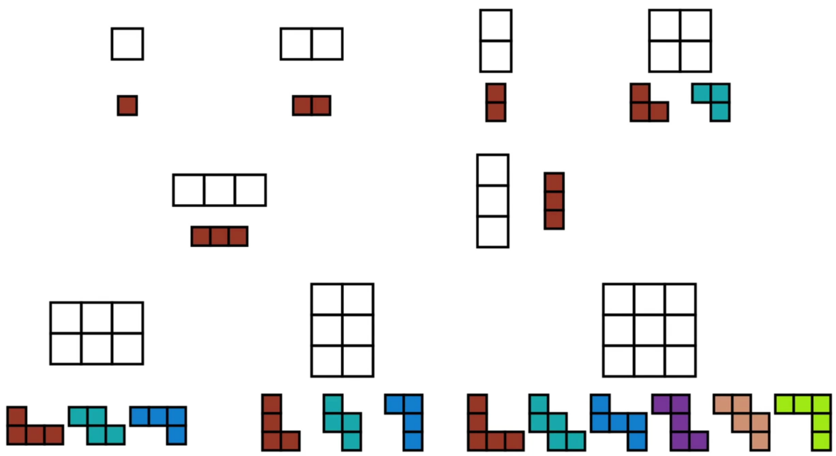


Figure 2: Number of Grid Paths for $m, n \in \{1, 2, 3\}$

Problem Statement:
For k pairs $(n_1, m_1), (n_2, m_2), \ldots, (n_k, m_k)$, find $\mathrm{NumberOfGridPaths}(n, m)$ for such n, m

Input Format
k
n_i m_i (each pair on a newline)
Output Format
$\mathrm{NumberOfGridPaths}(n_i, m_i)$ (each result on a newline)
Sample Input
6
1 1
2 5
3 3
6 3
7 10
17 8
Sample Output
1
5
6
21
5005
245157

Can you find efficient solution?

3. **Delannoy Numbers**
Delannoy numbers describes the number of paths from the southwest corner (0, 0) of a rectangular grid to the northeast corner (m, n), using only single steps north, northeast, or east.
Write a recursive function $\text{DelannoyNumber}()$ to count number of these paths Constraints: $n, m \geq 0$
Examples given in Figure 3

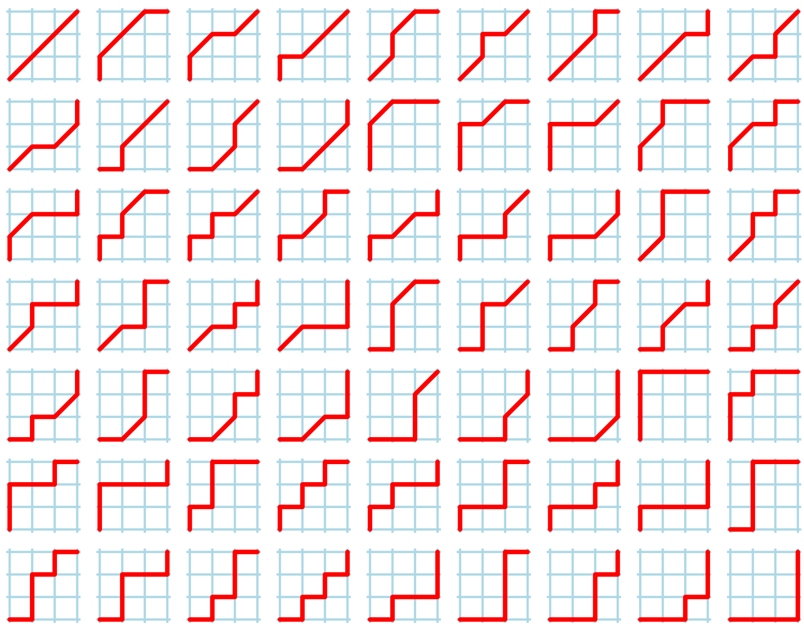


Figure 3: DelannoyNumber(3,3) = 63

Problem Statement:
For k pairs $(n_1, m_1), (n_2, m_2), \ldots, (n_k, m_k)$, find $\text{DelannoyNumbers}(n, m)$ for such n, m

Input Format
k
n_i m_i (each pair on a newline)
Output Format
$\text{DelannoyNumber}(n_i, m_i)$ (each result on a newline)
Sample Input
6
1 1
2 5
3 3
6 3
7 10
17 8
Sample Output
3
61
63
377
433905
245157
62390545

Can you find efficient solution?

4. **Partitions**

(a) Write a recursive function $\mathrm{NumberOfPartitions}()$ that counts the number of ways you can partition n objects using parts up to m

Constraints: $n, m > 0$
Examples given in Figure 4

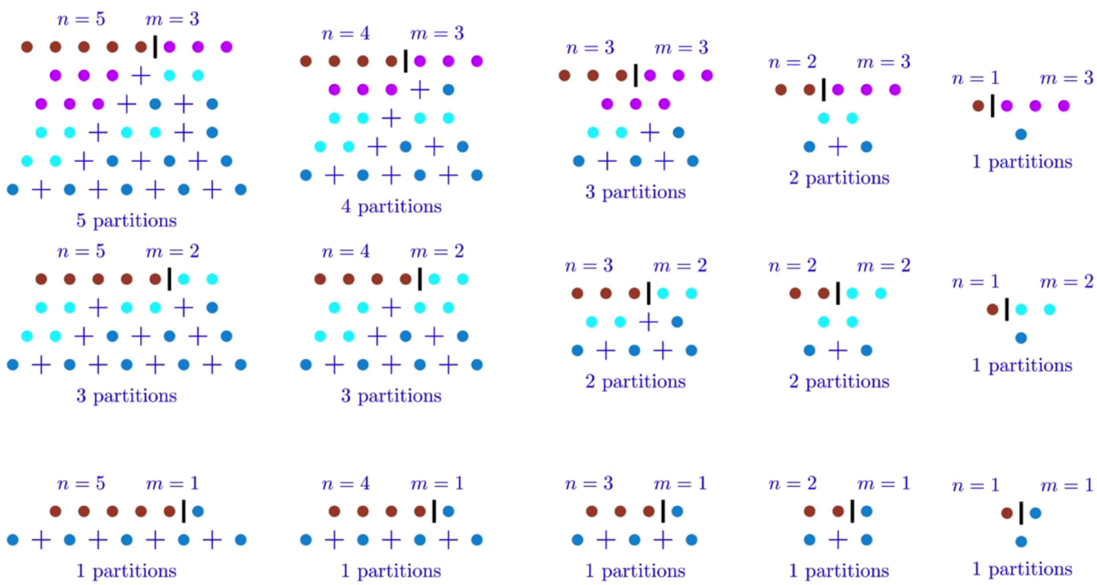


Figure 4: Partition n objects using parts up to m

Can you find efficient solution?
Problem Statement:
For k pairs $(n_1, m_1), (n_2, m_2), \ldots, (n_k, m_k)$, find $\mathrm{NumberOfPartitions}(n, m)$ for such n, m

Input Format
k
n_i m_i (each pair on a newline)
Output Format
$\mathrm{NumberOfPartitions}(n_i, m_i)$ (each result on a newline)
Sample Input
```
7
1 1
2 5
3 3
6 3
7 10
17 8
20 12
```
Sample Output
```
1
2
3
7
15
230
582
```

(b) Number of partitions $\mathrm{P}(n)$ of an integer n is same as $\mathrm{NumberOfPartitions}(n, n)$

Theorem 1 (Pentagonal Number Theorem)**.** *This theorem relates the product and series representations of the Euler function*

$$\prod_{n=1}^{\infty}(1-x^n) = \sum_{k=-\infty}^{\infty}(-1)^k x^{k(3k-1)/2} = 1 + \sum_{k=1}^{\infty}(-1)^k \left(x^{k(3k+1)/2} + x^{k(3k-1)/2}\right) \tag{8}$$

In other words,

$$(1-x)(1-x^2)(1-x^3)\cdots = 1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + x^{22} + x^{26} - \cdots \tag{9}$$

The exponents $1, 2, 5, 7, 12, \ldots$ on the right hand side are called (generalized) pentagonal numbers (A001318) and are given by the formula $g_k = k(3k-1)/2$ for $k = 1, -1, 2, -2, 3, -3, \ldots$

The equation (9) implies a recurrence for calculating $\mathrm{P}(n)$, the number of partitions of n:

$$\mathrm{P}(n) = \mathrm{P}(n-1) + \mathrm{P}(n-2) - \mathrm{P}(n-5) - \mathrm{P}(n-7) + \cdots \tag{10}$$

or more formally,

$$\mathrm{P}(n) = \sum_{k \neq 0}(-1)^{k-1}\,\mathrm{P}(n-g_k) \tag{11}$$

Problem Statement:
Write a function $\mathrm{P}()$ which calculates number of partitions using equation (10)

Input Format
Arbitrary number of testcases (each space separated)
Stop when input is negative
Output Format
Number of partitions for each testcase (each on a newline)
Sample Input
1 2 4 8 16 32 64 128 -1
Sample Output
1
2
5
22
231
8349
1741630
4351078600

Crazy Video

The hardest What comes next (Euler's pentagonal formula)

Exciting Puzzle

Towers of Hanoi: A Complete Recursive Visualization

§5. Practice Problems 5 - Arrays

1. **Horner's method** an algorithm for polynomial evaluation

$$\begin{aligned} \mathrm{P}(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n \\ &= a_0 + x\Big(a_1 + x\Big(a_2 + x\big(a_3 + \cdots + x(a_{n-1} + x\, a_n)\cdots\big)\Big)\Big) \end{aligned} \tag{12}$$

   This allows the evaluation of a polynomial of degree n with only n multiplications and n additions. This is optimal, since there are polynomials of degree n that cannot be evaluated with fewer arithmetic operations

   Problem Statement:
   Write a function Horner() takes three inputs an array of coefficients P, degree of polynomial n & input x and outputs the number $\mathrm{P}(x)$ which is $\mathrm{Horner}(\mathrm{P}, n, x)$
   For k testcases with coefficients $P_1[\ ], P_2[\ ], \ldots, P_n[\ ]$, degree $n_1, n_2, \ldots, n_k$ and variable $x_1, x_2, \ldots, x_n$ find $P_i(x_i)$ for each i in $\{1, 2, \ldots, n\}$

   Input Format
   k (number of testcases)
   n_i (degree of polynomial) $P_i[]$ ($n_i + 1$ coefficients with i^{th} index coefficient of i^{th} power) (each testcase on a newline)
   Output Format
   $\mathrm{Horner}(\mathrm{P}, n, x)$ (each result on a newline)
   Sample Input
   3
   0 5 1
   1 -3 2 2
   3 2 -1 -3 4
   Sample Output
   5
   7
   80

2. **Remove Duplicates**
   Problem Statement:
   Take an integer array as input and output all unique elements of that array (for repeated elements keep only first such occurence)

   Input Format
   k (number of testcases)
   n_i (size of array) followed by n_i elements of array (each testcase on a newline)
   Output Format
   Each (each result on a newline)
   Sample Input
   2
   15 1 4 9 10 18 1 4 9 16 17 6 10 11 13 17
   20 1 3 4 8 16 1 11 15 17 20 1 3 14 15 18 4 5 17 19 20
   Sample Output
   1 4 9 10 18 16 17 6 11 13
   1 3 4 8 16 11 15 17 20 14 18 5 19

3. **Large Factorials**

   Compute factorial of large numbers $n > 20$

   Note. `long long int` *can not store more than 20 digits. Maximum digits in testcases* $= 150$

   Input Format
   n (number of testcases)
   k_i for $i = 1, 2, \ldots, n$

   Output Format
   $\text{factorial}(k_i)$ (each result on a newline)

   Sample Input
   6
   21 33 57 69 77 93

   Sample Output
   51090942171709440000
   8683317618811886495518194401280000000
   40526919504877216755680601905432322134980384796226602145184481280000000000000
   171122452428141311372468338881272839092270544893520369393648040923257279754140647424000000000000000
   145183092028285869634070784086308284983740379224208358846781574688061991349156420080065207861248000000000000000000
   1156772507081641574759205162306240436214753229576413535186142281213246807121467315215203289516844845303838996289387078090752000000000000000000000

§6. Dynamic Programming

Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

Dynamic Programming Methods

- **Top-down with Memoization** - In this approach, we try to solve the bigger problem by recursively finding the solution to smaller sub-problems. Whenever we solve a sub-problem, we cache its result so that we don't end up solving it repeatedly if it's called multiple times. Instead, we can just return the saved result. This technique of storing the results of already solved subproblems is called Memoization.
- **Bottom-up with Tabulation** - Tabulation is the opposite of the top-down approach and avoids recursion. In this approach, we solve the problem "bottom-up" (i.e. by solving all the related sub-problems first). This is typically done by filling up an n-dimensional table. Based on the results in the table, the solution to the top/original problem is then computed.

Recursive Formula

```
int fibonacci(int n){
    if(n <= 1) {
        return n;
    }
    else {
        fibonacci(n-1)+fibonacci(n-2);
    }
}
```

Bottom-up with Tabulation

```
int fibonacci[n];
fibonacci[0] = 0;
fibonacci[1] = 1;
for (int i = 2; i < n; ++i)
{
    fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
}
cout<<fibonacci[n];
```

As a practice solve question 2, 3, 4 of practice problems 4 using dynamic programming

1. **Currency sums** India's currency consists of 1,2,5,10,20,50,100,200,500,2000.
   Problem Statement:
   Write a function to get number of different ways n can be made using any number of given coins/notes.

   Input Format
   n
   $k_1, k_2, \ldots, k_n$
   Output Format
   Correct answer (each on a newline)
   Sample Input
   9
   1 2 5 10 20 50 100 200 500 2000
   Sample Output
   1
   4
   11
   41
   451
   4563
   73682
   6295435
   27984272287

References

5 Simple Steps for Solving Dynamic Programming Problems
What is Dynamic Programming
Coin sums: Project Euler

§7. Hints

§§7.1. Practice Problems 1

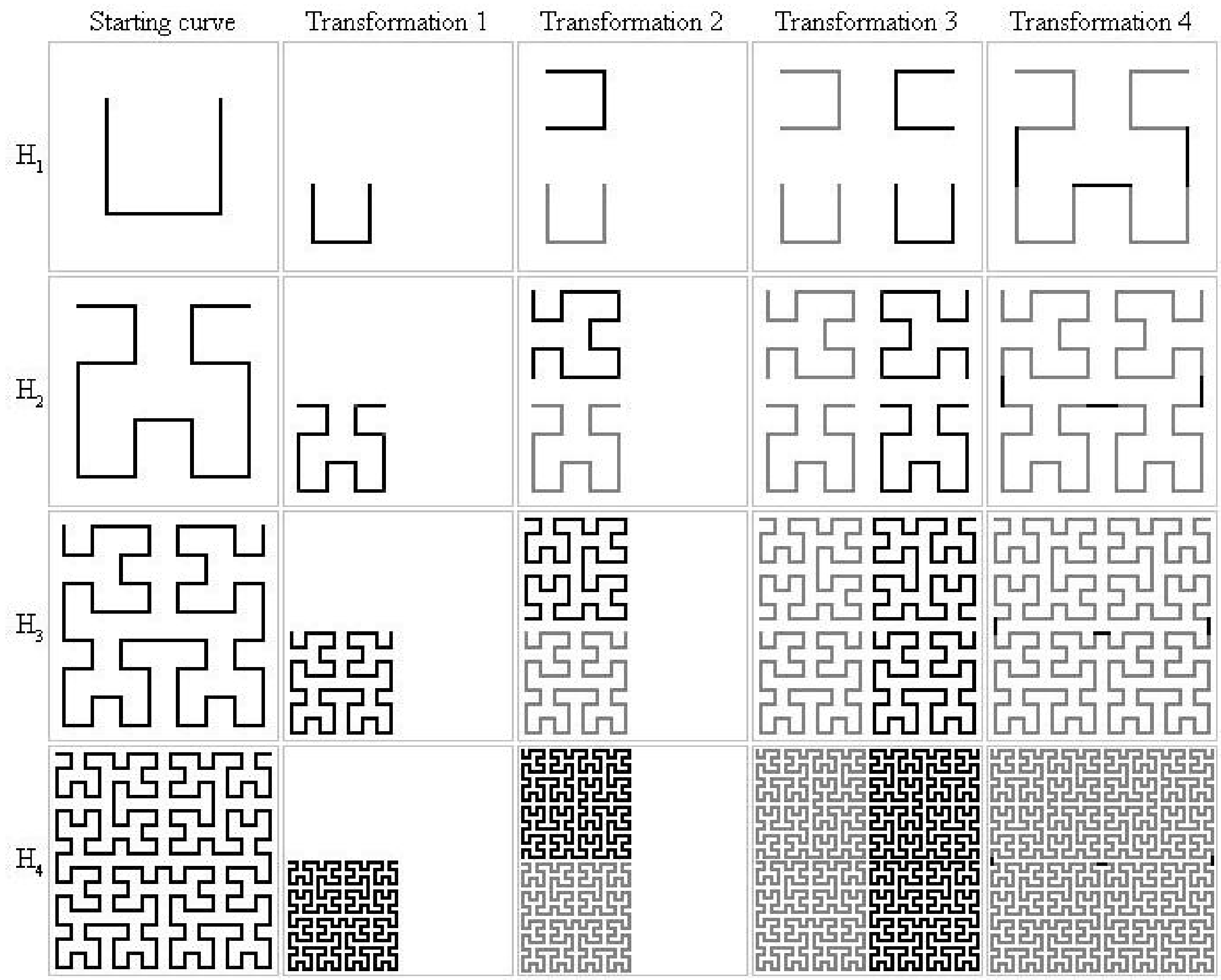


Figure 5: Hilbert Curve Transformations

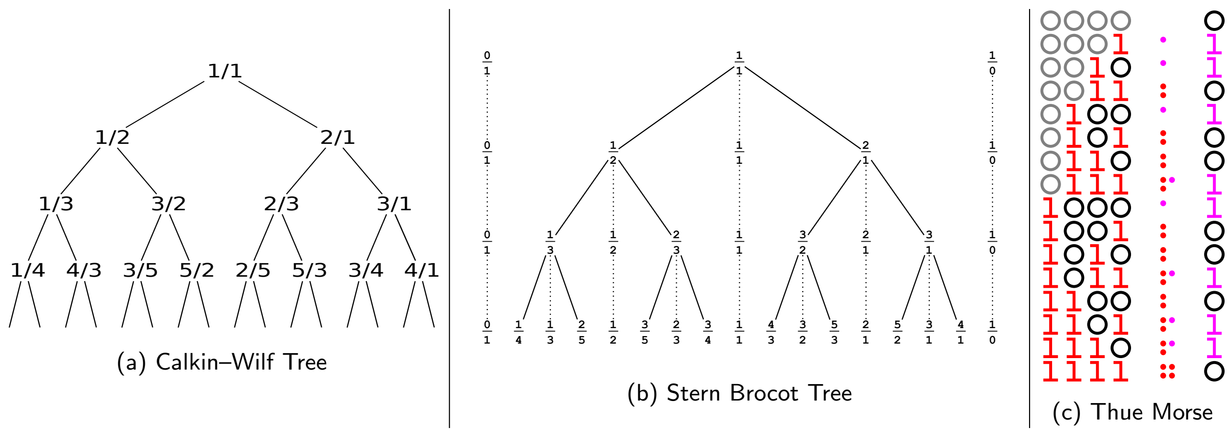
§§7.2. Practice Problems 2

1. The sequence repeats when we encounter '0' then '1'
2. Similar to calculating e (problem given in slides)

§§7.3. Practice Problems 3

1. Try skipping days, months, years and centuries :D

   Is there a formula?
2. Similar pattern as in Stern Brocot Tree. Also, exactly same pattern between 3 consecutive terms
3. How will you calculate n^{th} term?



(a) Calkin–Wilf Tree

(b) Stern Brocot Tree

(c) Thue Morse

Figure 6