STMC coding team Training

Lesson 7: Data Structure I

Tsai Yun Chen

April 12, 2024

Goal today

Today we will introduce a series of types of data structure that will become useful tools for us to solve harder problem.

- A brief introduction to class
- Why data structure?
- Linked list
- Linked-list vs array

Class, a new approach

- Up to now, our program are `imperative`
- one command, one operation
- it's an effective paradigm for designing simple program
- hard to organize when scale become large

An example of large scale program

Consider you are working in a game company and you need to work with your colleague to create a game, how will you distribute the work?
Let's take Minecraft as an example, what should be done?

- Player Character (PC): movement, crafting, mining, interaction with NPC,...
- Non-Player Character (NPC): movement (self-petrol), interaction with PC, generation and distribution,...
- Block: texture, interaction, generation and distribution, ...
- Sandbox: Generation of biome, Manage of block and NPC, size and scope, ...
- and a lot more.....

An example of large scale program

Let's assign one person for each job, but the problem is they are actually depending on each other, so the thing have to be done in order so that we can ensure there is no bug, e.g.

1. Implement of basic block that forms biome
2. Implement sandbox that generate the biome
3. Implement PC and NPC movement, then their interaction
4. Back to sandbox and implement manage of block and NPC
5. Back to basic block and implement their interaction with PC/NPC
6. implement PC's possible action (crafting, mining etc.)

An example of large scale program

If we follow such procedure, it will take decades to finish one game. Moreover, imagine the number of function needed to code such a large software, things will spread around and become messy. We need some other idea.

- Idea from real life: Simulate how we understand and live in this world
- We recognize things as different **classes** of **objects**, each having their own functionality
- We only need to know what the thing can do, but not what behind.
- We need to able to create similar things, but keeping certain order of freedom

Class and Object

There are plenty of examples in real life that share the same paradigm,

- class Animal (a class of things)
- sub-class Human (a class of things, but it is a kind of Animal)
- Human Vincent (an instance/object of Human class)
- class character (a class of things)
- sub-class NPC (a kind of character, but not controllable by player)
- sub-class PC (a kind of character, controllable by player)

Idea of class

- The main idea of class is to abstractlize properties and functionalities of things, e.g.
  - apple –> fruit –> food
  - button –> interactable UI object –> UI object
- It is a revolutional framework that its used in most of the complicated system nowadays
- We will introduce the very basic usage of it for implementing today's main topic.

Syntax of class

```
class ClassName:
  def __init__(_paramerter_here):
    self.member1=sth
    self.member2=sth
    ...
  def method1(_param_):
    ...
```

A class mainly contains two type of things, members and methods, members are properties of the object and methods are the thing that the object can do.

An example

```
class Cup:
  def __init__(self,volume):
    self.volume=volume
    self.contains=0
  def fill(self):
    self.contains=self.volume
    return
  def drink(self):
    self.contians=0
    return

Mycup=Cup(100) #pay extra attentions to indentation here
print(Mycup.volume,Mycup.contains)
Mycup.fill()
print(Mycup.volume,Mycup.contains)
Mycup.drink()
print(Mycup.volume,Mycup.contains)
```

Data structure

- data structure is a way to allow users efficiently store, organize and access data
- No data structure are perfect, one need to decide which to use for best efficiency
- example:
  - array
  - linked-list (array-like, but different structure)
  - queue and stack (structure from real life)
  - map, hashmap (function-like storage)
  - tree, graph (advanced structure for handling data with intra-relation)

Why data structure?

- Indeed Array is an powerful data structure that in most “common” circumstances, array is efficent and useful.
- why do we need other structure?
- Problems of array:
  - costly to insert item (array is usally fixed size and inserting need to reallocate and moving elements)
  - hard to remove item while maintaining certain structure
  - slow for searching in general

Linked List

- A data structure that shares similar structure of array
- different method of construction –> different benefit and Problems

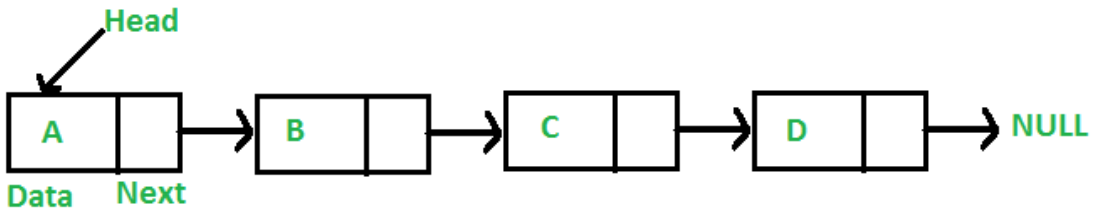

Figure 1: Linked List(Source)

Implementation

```
1  class LinkedList:
2    def __init__(self):
3      self.data=None
4      self.next=None
5    def insert(self,data):
6      if self.data==None:
7        self.data=data
8        self.next=LinkedList()
9      else:
10       self.next.insert(data)
11   def find(self,data):
12     if self.data==data:
13       return True
14     elif self.data==None:
15       return False
16     else:
17       return self.next.find(data)
```

Pros and Cons of Linked List

- Pros:
  - Fast to insert: In practice we have more complicated implementation to allow direct jump to the end and insert
  - memory efficient: memory usage proportion to data stored without special care
  - easy to remove, rearrange elements
- Cons:
  - searching is still slow
  - slow to access data by position
- Usage:
  - Not particularly useful alone
  - Usually use as base of other structure
  - useful when frequently update data but have very few query