Competitive Programming

Lesson 3 – Basic Data Structures

Written by Jason Sun

Edited by Puneet Bhat

Languages...

There will be examples for what each data structure is called in C++ and Python.

If you want to find it in your language, just do a quick google search! Really easy. Keep in mind that everyone will need to search up specific information for each data structure, since our examples aren't extremely detailed.

For C++ Examples

There will be two lines of code above the entire program.

1. import the entire C++ standard library.
2. Use namespace std to make writing faster

```
#include <bits/stdc++.h>
using namespace std;
```

1

Built In (Probably)

Your very basic data structures. Like an array! This is the **MOST USEFUL** lesson for general purpose programming! You will have to use these data structures A LOT, meaning that understanding when to use them is essential.

List

Information

A list is just a dynamic array. A dynamic array is just an **array with dynamic size**.

So if I make a list like so

- create [1, 2, 3]

I can add or remove an element (at the end of the list) in *(pretty much)* constant time.

- add 4 to the end [1, 2, 3, 4]

Although inserts take O(n) time because some elements are pushed to the next index.

Examples

Think about using lists when you want, well, any list of something, but you want to be able to add more things.

C++:

```
vector<int> myList = {1, 2, 3};
vector<int> myList(size, element);
```

Python:

```
list() or []
myList = [1, 2, 3]
```

2D List

Information

Remember that you can put a list in a list! (Or an array in an array.)

So I would use this if I, for example, needed to represent a chess board.

```
[
[2, 3, 4, 5, 6, 4, 3, 2],
[1, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[1, 1, 1, 1, 0, 1, 1, 1],
[2, 3, 4, 5, 6, 4, 3, 2]
]
```

Examples

This can help for any grid you want to store.

C++:

```
vector<vector<int>> grid;
```

Python:

```
grid = [[0, -4], [1, 3], [7, 2]]
```

Set

Information

Has O(1) add, remove, and look up.

- Put 4 in the set
- Is 4 in the set? -> True

Had **no order** and does **not** have **duplicates**.

eg. a list of {1, 3, 4, 2, 4, 6, 3}
when converted to a set
becomes {1, 3, 4, 2, 6}

Examples

Think about using sets when you want to quickly quick if something is in the set or not.

C++:
unordered_set<int> backpack;

Python:
set()
backpack = {“apple, “pencil”, “book”}

The time complexity to check if a specific element is in an array is O(n)

Map

Information

Wow it's a hashmap! It's the same as a set, except each element is called a key, and each key has a value!

The keys have to be unique and **immutable**, but the values can be anything!

Eg.
{
"mario": "password123",
"luigi": "131415"
}

or
{0: "Jason Sun", 1: "Puneet Bhat"}

Examples

Think about using maps when you need something similar to a login database.

- Check if username in map
- If username in map check if password == map[username]

C++: unordered_map
Python: dict() or {}

Immutable: Can not change the value. Examples include strings and numbers. However, an array, for example, is mutable.

2

Easy Data Structures

Very simple, easy data structures that you might not have heard of before. Your coding language might have these in their standard libraries.

Tuple

Information

A tuple is just a list that can not be changed. This means you can not assign values to indices.

This might be helpful if you want to put tuples into a set, since sets only allow immutable values (values that can not be changed).

Examples

C++:

```
tuple<int, int, int> myTuple(1, 2, 3);
```

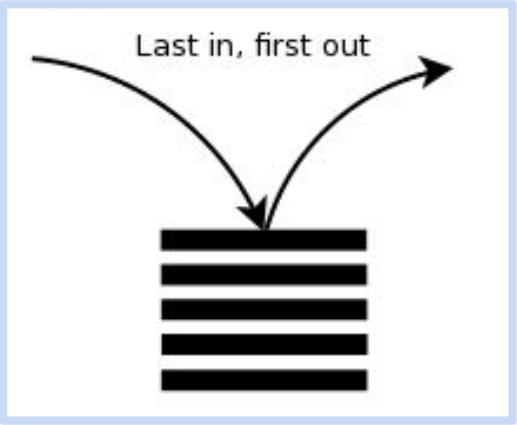
Python:

```
tuple() or ()
myTuple = (1, 2, 3)
```

Stack

Information

O(1) add and removal from the end/top of the stack. Think of it like a stack of clothes. You can only put more clothes on top, or take away the top clothing to wear.



Examples

C++:
stack<int> clothes;

Python:
list() or []

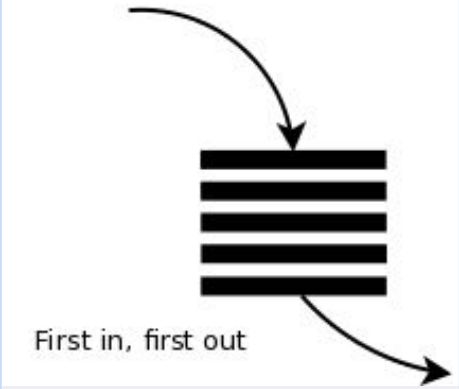
You can just use a dynamic array to represent a stack. As long as it has / you only use the below:

- What is the top element?
- Delete top element.
- Add an element to the top.

Queue

Information

O(1) add from one side (the end) of the queue and O(1) removal from the other side (the front). Think of it like an actual queue, for example, people waiting in line at the cashier.



Examples

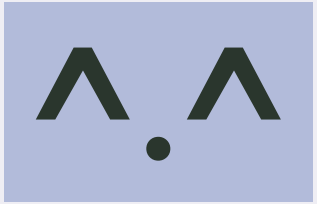
C++:

```
queue<int> lineUp;
```

Python: (python does not have queue)

```
from collections import deque
lineUp = deque()
lineUp = deque([1, 2, 3])
```

A **deque** can add and remove in O(1) time on **both sides**, so you can represent a queue with a deque (since for a queue you need a data structure with O(1) removal from one side, and O(1) add from the other side), and a deque has all of that, and a bit extra.



Practice

If you want to improve, practice! Do the homework as well as some DMOJ questions outside of class. Make it a fun hobby!

Editorials

If you get stuck on a DMOJ problem, some problems have an editorial where they give you **hints** and/or the **solution**.

This can be good to partially read whenever you get **stuck** for 30 minutes (you couldn't think of any new ideas of **30 minutes**). Try not to read the entire editorial, but only read it until you have a better idea and direction, then keep thinking and trying to solve the problem.

Keep in mind: **You aren't trying to solve the problem, you are trying to improve your ability to solve.** If you do need to read the entire solution, try to think about how the solution might have been made so that you can recreate it for similar problems.

Hints?

If you need a hint, and there is no editorial, you can always ask **ChatGPT** for a hint on any ideas you missed, or ask it for any prerequisite content you should know and **research** before further attempting the problem. You might need to explicitly ask ChatGPT **NOT** to give you the full solution.

Homework

Junior

1) Unique Elements – https://dmoj.ca/problem/set
2) Snow Calls – https://dmoj.ca/problem/ccc05s1
3) CCC '15 S2 – Jerseys – https://dmoj.ca/problem/ccc15s2
4) Hardcore Grinding – https://dmoj.ca/problem/grind
5) CCC '22 S2 – Good Groups – https://dmoj.ca/problem/ccc22s2

Senior

1) Postfix Notation – https://dmoj.ca/problem/postfix
2) The Geneva Confection – https://dmoj.ca/problem/ccc14s3
3) Snowflakes – https://dmoj.ca/problem/cco07p2 **(HARD)**

Thanks!

Do you have any questions?

https://github.com/CryoJS/DSA-Handbook