DATA STRUCTURES

- A **data structure** is a particular way of organizing data in a computer so that it can be used effectively
  - Arrays (Fixed size)
  - Dynamic Arrays
  - TreeSet / TreeMap (ordered)
  - HashSet / HashMap (unordered)
  - Heap and Priority Queue
  - Stack / Queue / Deque
  - Linked list
  - Pair / Tuple
  - Customized data structure
- Different DS suits different problems
  - use C++ STL, Java API and/or standard libraries included with Python 3

1

1

DATA STRUCTURES – CONT.

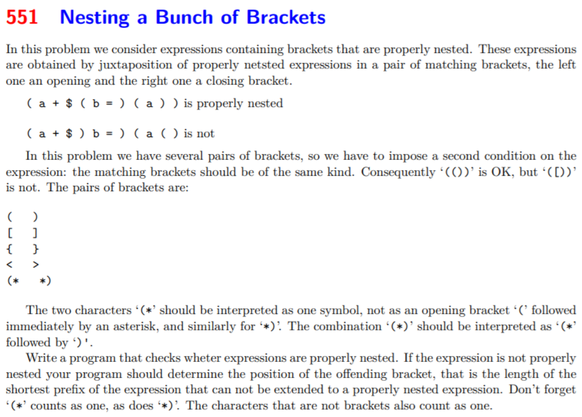
	C++	Java
Dynamic Arrays	vector	ArrayList (slightly faster than Vector)
TreeSet (ordered) TreeMap (ordered)	set map	TreeSet TreeMap
HashSet (unordered) HashMap (unordered)	unordered_set unordered_map	HashSet HashMap
Heap and Priority Queue	priority_queue	PriorityQueue
Stack	stack	Stack
Queue / Deque	queue deque	Queue Deque
Linked list	list	LinkedList
Pair / Tuple	pair<a, b> tuple<a, b, c>	AbstractMap.SimpleEntry

2

2

STACK

- Last In First Out (LIFO)
  - O(1) for insertion (push) and O(1) deletion (pop) from the top
  - Used in Recursion, Evaluation of Postfix Expressions, Bracket Matching
- Example:
  - Bracket matching: UVA 551 - Nesting a Bunch of Brackets
    - Read the brackets one by one from left to right. Every time we encounter a close bracket, we need to match it with the latest open bracket
      - (* *)



551 Nesting a Bunch of Brackets

In this problem we consider expressions containing brackets that are properly nested. These expressions are obtained by juxtaposition of properly netsted expressions in a pair of matching brackets, the left one an opening and the right one a closing bracket.

`( a + $ ( b = ) ( a ) )` is properly nested

`( a + $ ) b = ) ( a ( )` is not

In this problem we have several pairs of brackets, so we have to impose a second condition on the expression: the matching brackets should be of the same kind. Consequently '(())' is OK, but '([))' is not. The pairs of brackets are:

```
(   )
[   ]
{   }
<   >
(*   *)
```

The two characters '(*' should be interpreted as one symbol, not as an opening bracket '(' followed immediately by an asterisk, and similarly for '*)'. The combination '(*)' should be interpreted as '(*' followed by ')'.

Write a program that checks wheter expressions are properly nested. If the expression is not properly nested your program should determine the position of the offending bracket, that is the length of the shortest prefix of the expression that can not be extended to a properly nested expression. Don't forget '(*' counts as one, as does '*)'. The characters that are not brackets also count as one.

POSTFIX NOTATION

- Postfix notation is a notation for writing arithmetic expressions in which the operands appear before their operators

```
1. Read postfix expression from left to right till the end
     • If number
          • push it onto Stack
     • If operator
          i. Pop two items
               • A <- Top item
               • B <- Next to Top item
          ii. Evaluate B operator A
               • push calculation result onto Stack
2. result <- Top element
3. END
```

923*-

3

- Extend: Infix to Postfix Conversion Using Stacks

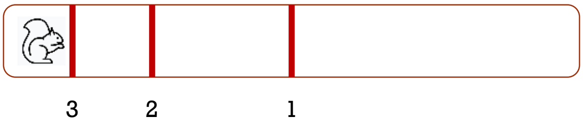
DYNAMIC ARRAYS

- Vector / ArrayList
  - Efficiently add/remove elements at the end of the structure. In general, vector is almost as fast as using an ordinary array in O(1) time
- Deque
  - Dynamic array that can be efficiently manipulated at both ends of the structure
  - O(1) average time with a larger constant factor
  - Used in algorithms for 'Sliding Window' problem, etc
- Queue
  - Used in Breadth First Search, Topological Sort, etc
- Example:
  - https://codeforces.com/problemset/problem/879/B

EXAMPLE - ESCAPE FROM STONES

- Question
  - https://codeforces.com/problemset/problem/264/A

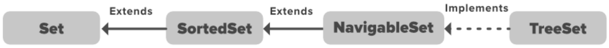
l l r l r



3	5	4	2	1

SET/TREESET

- In C++, set is ordered and based on a balanced search tree and its operations work in O(log n) time
- unsorted_set: in C++, unsorted_set is based on a hash table and its operations work on O(1) on average.
- C++ STL set, similar to C++ STL map
  - map stores a (key, data) pair
  - set stores just the key
- In Java: TreeSet based on a self-balancing tree



- Example: UVa10815 - Andy's First Dictionary

MAPS

- Associative containers that store elements in a mapped fashion
  - Each element has a key value and a mapped value
  - No two mapped values can have same key values
- map is based on a balanced binary search tree and its operations work in O(log n) time
  - C++ STL map (Java TreeMap)
- Example
  - Given an array, your task is to find the k-th occurrence (from left to right) of an integer v.

```
8 3
1 3 2 2 4 3 2 1
1 3                 2
2 4                 0
3 2                 7
```

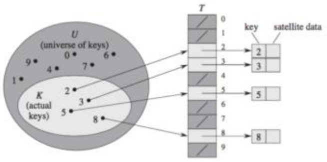
$1 \le n, m \le 100,000, 1 \le k \le n, 1 \le v \le 1,000,000)$.

PRIORITY QUEUE

- A multiset designed such that the first element of the queue is the greatest of all elements in the queue and elements are in non increasing order(hence we can see that each element of the queue has a priority{fixed order}).
  - Smaller constant factor than multiset
  - Based on heap structure, which is a special binary tree
  - Used when you only needs to find minimum or maximum value
  - Descending order
    - largest value
- UVA 1203 Argus

HASH TABLE

- Advertised O(1) for insert, search, and delete, but
  - The hash function must be good!
  - There is no Hash Table in C++ STL (Yes in Java API)
- Nevertheless, O(log n) using map is usually ok
- Direct Addressing Table (DAT)
  - Key values are distinct, and is drawn from a universe U = {0, 1, . . . , m - 1}
  - Store the items in an array, indexed by keys
- Example
  - UVa 11340 (Newspaper)



COMPARISONS – IN C++

- Count number of unique elements

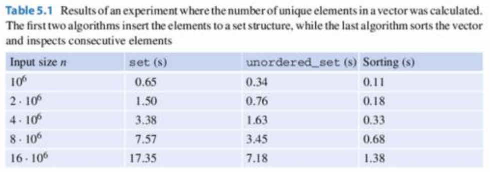


Table 5.1 Results of an experiment where the number of unique elements in a vector was calculated. The first two algorithms insert the elements to a set structure, while the last algorithm sorts the vector and inspects consecutive elements

Input size n	set (s)	unordered_set (s)	Sorting (s)
10^6	0.65	0.34	0.11
$2 \cdot 10^6$	1.50	0.76	0.18
$4 \cdot 10^6$	3.38	1.63	0.33
$8 \cdot 10^6$	7.57	3.45	0.68
$16 \cdot 10^6$	17.35	7.18	1.38

- Determine the most frequent value

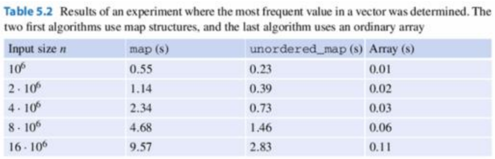


Table 5.2 Results of an experiment where the most frequent value in a vector was determined. The two first algorithms use map structures, and the last algorithm uses an ordinary array

Input size n	map (s)	unordered_map (s)	Array (s)
10^6	0.55	0.23	0.01
$2 \cdot 10^6$	1.14	0.39	0.02
$4 \cdot 10^6$	2.34	0.73	0.03
$8 \cdot 10^6$	4.68	1.46	0.06
$16 \cdot 10^6$	9.57	2.83	0.11

- Add/Remove elements

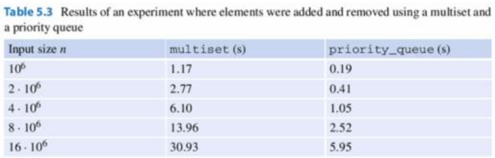


Table 5.3 Results of an experiment where elements were added and removed using a multiset and a priority queue

Input size n	multiset (s)	priority_queue (s)
10^6	1.17	0.19
$2 \cdot 10^6$	2.77	0.41
$4 \cdot 10^6$	6.10	1.05
$8 \cdot 10^6$	13.96	2.52
$16 \cdot 10^6$	30.93	5.95

PREFIX SUM ARRAY

- The sums of prefixes (running totals) of the input sequence:

input numbers	1	2	3	4	5	6	…
prefix sums	1	3	6	10	15	21	…

$\text{PreSum}_0 = a_0$
$\text{PreSum}_1 = a_0 + a_1 = \text{PreSum}_0 + a_1$
$\text{PreSum}_2 = a_0 + a_1 + a_2 = \text{PreSum}_1 + a_2$
. . .
$\text{PreSum}_n\text{=}\text{PreSum}_{n\text{-}1}\text{+}a_n$

- Example:
  - Stripe: https://codeforces.com/problemset/problem/18/C