1: Introduction to Competitive Programming

CPCFI

UNAM's School of Engineering

2021

Based on: Halim S., Halim F. *Competitive Programming 3*. Handbook for ACM ICPC and IOI Contestants. 2013

Table of Contents

Competitive Programming

- **CP?**:
  Given a well-known Computer Science problem, solve it as quickly as possible!
- **Objective**:
  The true end goal is to produce all-rounder computer scientists/programmers who are much readier to produce better software and to face harder CS research problems in the future

CPCFI

CP Problems

No	Category	In This Book	Frequency
1.	Ad Hoc	Section 1.4	1-2
2.	Complete Search (Iterative/Recursive)	Section 3.2	1-2
3.	Divide and Conquer	Section 3.3	0-1
4.	Greedy (usually the original ones)	Section 3.4	0-1
5.	Dynamic Programming (usually the original ones)	Section 3.5	1-3
6.	Graph	Chapter 4	1-2
7.	Mathematics	Chapter 5	1-2
8.	String Processing	Chapter 6	1
9.	Computational Geometry	Chapter 7	1
10.	Some Harder/Rare Problems	Chapter 8-9	1-2
		Total in Set	8-17 ($\approx\leq 12$)

Figure: ACM ICPC (Asia) Problem Types. See [Halim]

CP Programmer POV

No	Category	Confidence and Expected Solving Speed
A.	I have solved this type before	I am sure that I can re-solve it again (and fast)
B.	I have seen this type before	But that time I know I cannot solve it yet
C.	I have not seen this type before	See discussion below

Figure: Problem Types. See [Halim]

CPCFI

Algorithm Analysis

Given the maximum input bound, can the currently developed algorithm, with its time/space complexity, pass the time/memory limit given for that particular problem?

Code Testing

Guidelines for designing good test cases:

1. Your test cases should include the sample test cases since the sample output is guaranteed to be correct
2. For problems with multiple test cases in a single run, you should include two identical sample test cases consecutively in the same run. This helps to determine if you have forgotten to initialize any variables
3. Your test cases should include tricky corner cases. Corner cases typically occur at extreme values such as $N = 0$, $N = 1$, negative values, large final (and/or intermediate) values that does not fit 32-bit signed integer, etc.

Code Testing

4. Your test cases should include large cases. Increase the input size incrementally up to the maximum input bounds stated in the problem description. Sometimes your program may work for small test cases, but produces wrong answer, crashes, or exceeds the time limit when the input size increases.
5. Do not assume that the input will always be nicely formatted if the problem description does not explicitly state it. Try inserting additional whitespace (spaces, tabs) in the input and test if your code is still able to obtain the values correctly without crashing.

CPCFI

Practice

Figure: https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=604

Figure: https://icpc.global/compete/problems

CPCFI

Getting Started: Anatomy of a problem

- Background story/problem description
- Input and Output description
- Sample Input and Sample Output
- Hints or Footnotes

Table of Contents

C++

The C++ Programming Language fourth edition. Bjarne Stroustrup

- Part I: Chapters 2-5
- Part IV: The Standard Library (STL)

I/O

Problem: output the sum of numbers on each case

- Case 1: n input cases

```
Sample Input  | Sample Output
------------------------------
3             | 3
1 2           | 12
5 7           | 9
6 3           |
------------------------------
```

- Case 2: Stop until 0 0

```
Sample Input  | Sample Output
------------------------------
1 2           | 3
5 7           | 12
6 3           | 9
0 0           |
------------------------------
```

I/O

- Case 3: Stop until `EOF`

```
Sample Input | Sample Output
-----------------------------
1 2          | 3
5 7          | 12
6 3          | 9
-----------------------------
```

- Case 4: Each case indicates the number of elements to sum

```
Sample Input | Sample Output
-----------------------------
1 1          | 1
2 3 4        | 7
3 8 1 1      | 10
4 7 2 9 3    | 21
5 1 1 1 1 1  | 5
-----------------------------
```

CPCFI

I/O C++ code

- Case 1

```
#include <stdio.h>
int main() {
  int TC, a, b;
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &a, &b);
    printf("%d\n", a + b);
  }
}
```

- Case 2

```
#include <stdio.h>
int main() {
  int a, b;
  while (scanf("%d %d", &a, &b), (a || b)) {
    printf("%d\n", a + b);
  }
}
```

CPCFI

I/O C++ code

- Case 3

```c
#include <stdio.h>
int main() {
  int a, b;
  while (scanf("%d %d", &a, &b) != EOF) {
    printf("%d\n", a + b);
  }
}
```

- Case 4

```c
#include <stdio.h>
int main() {
  int k;
  while (scanf("%d", &k) != EOF) {
    int v, ans = 0;
    while (k--) {
      scanf("%d", &v);
      ans += v;
    }
    printf("%d\n", ans);
  }
}
```

Table of Contents

Time to Start the Journey

What we have to learn to do, we learn by doing.

- Aristotle, Nichomachean Ethics

UVa - Competitive Programming 3

University of Valladolid's Online Judge:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=604

- Most of these problems only need a basic computer science knowledge to code a solution
- UVa: **CP3** > **Introduction** > **Getting Started: The Easy Problems**

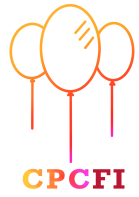


Table of Contents

Ad Hoc Problems

Ad Hoc problems are problems that 'cannot be classified anywhere else' since each problem description and its corresponding solution are 'unique'.

- Ad Hoc problems frequently appear in programming contests. In ICPC, $\approx 1-2$ problems out of every ≈ 10 problems are Ad Hoc problems
- If the Ad Hoc problem is easy, it will usually be the first problem solved by the teams in a programming contest
- In an ICPC regional contest with about 60 teams, your team would rank in the lower half (rank 30-60) if you can only solve Ad Hoc problems
- Most Ad Hoc problems can be solved without using advanced data structures or algorithms

CPCFI

Ad Hoc Categories

- Cards
- Chess
- Other games
- Palindromes
- Anagrams
- Interesting real life problems
- Dates and time
- Time Wasters

UVa - Competitive Programming 3

University of Valladolid's Online Judge:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=604

- Most of these problems only need a basic computer science knowledge to code a solution
- UVa: **CP3** > **Introduction** > **Ad Hoc Problems**

CPCFI

Table of Contents

Importance of Data Structures

- Most problems can be solved in many ways, however, using an efficient or the most proper data structure can be the difference between an accepted solution or a time limit exceeded solution
- The knowledge required must be such that given a problem, one should identify the correct DS to use
- Aim to understand the following concepts from each DS:
  - Strengths/Weaknesses
  - Time/Space complexities

CPCFI

Linear Data Structures

- **Def:** a data structure in which its elements form a linear sequence (from left to right or top to bottom)
- LDS:
  - Static Array
  - Dynamically-Resizeable Array
  - Linked List
  - Doubly Linked List
  - Stack
  - Queue
  - Doubly Linked List
  - Double-ended Queue (Deque)
  - Bitset
  - Bitmasks (lighweight small sets of Booleans)
- https://visualgo.net/en

CPCFI

But first...common background for most LDS

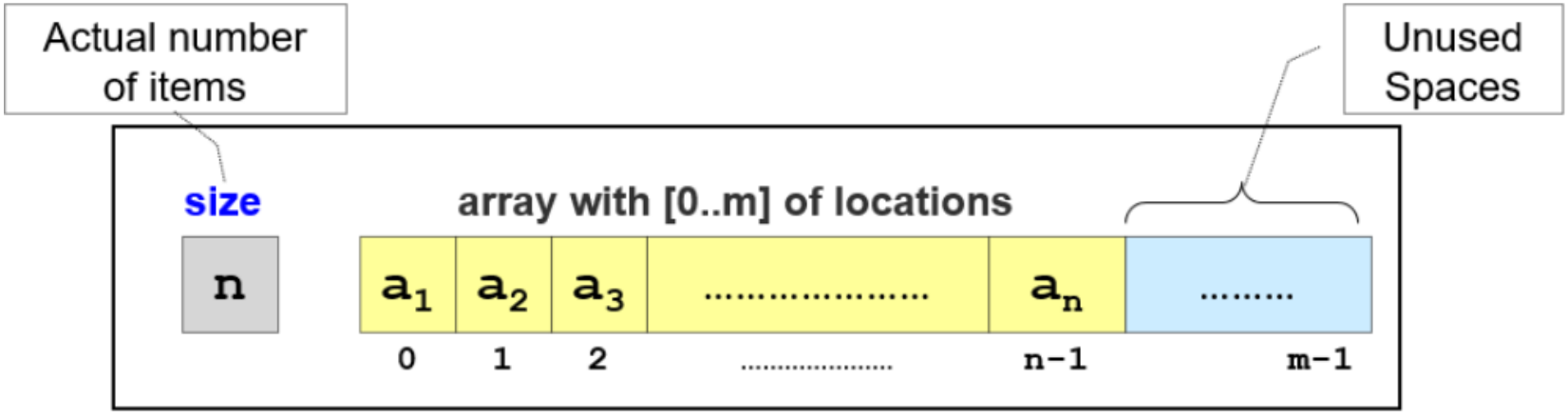
List Abstract Data Type (ADT) is a sequence of items where positional order is important:

$$a_0, a_1, \ldots, a_{n-2}, a_{n-1}$$

Common operations among lists:

1. `get(i)`: returns a_i (0-based indexed)
2. `search(v)`: decide if item v exists in the list. If true, also report its index, otherwise, return `-1`
3. `insert(i, v)`: inserts item v at index i; possibly shifting elements in the range $[i, \ldots, n-1]$ to the right one position
4. `remove(i)`: removes item at index i in the list; possibly shifting elements in the range $[i+1, \ldots, n-1]$ to the left one position

Static Array



Static Array

Great candidate for implementing a List ADT !

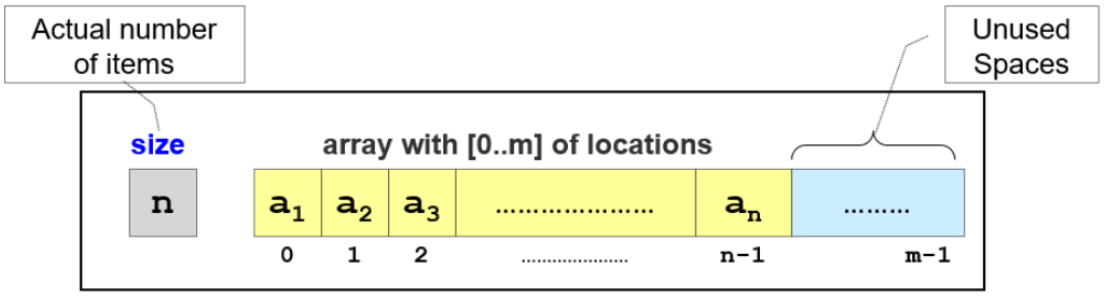


Figure: List ADT implementation using Array

Compact array:

- $[0, \ldots, n-1]$ indices are occupied
- $[n, \ldots, m-1]$ indices are empty

CPCFI

Static Array: Operations

Let A be a static array with indices `[0,...,n-1]` occupied:

- `get(i)`: return `A[i]`
- `search(v)`: check each index one by one to see if `A[i] == v`. Return `i` if `true`, otherwise, return `-1`
- `insert(i, v)`: shift elements in range `[i,...,n-1` one place to the right and set `A[i] = v`
- `remove(i)`: shift elements in range `[i+1,...,n-1]` one place to the left

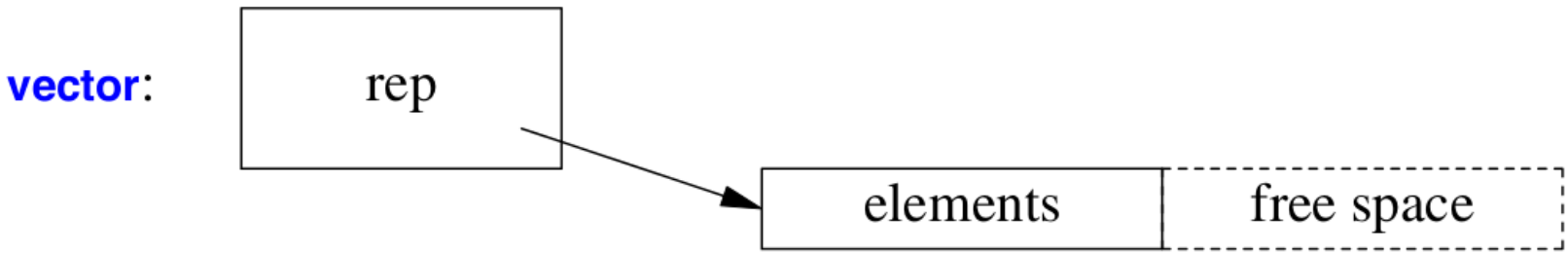
Static Array: Time complexity

- `get(i)`: $O(1)$
- `search(v)`: $O(n)$
- `insert(i, v)`: $O(n)$
- `remove(i)`: $O(n)$

Static Array: Problems

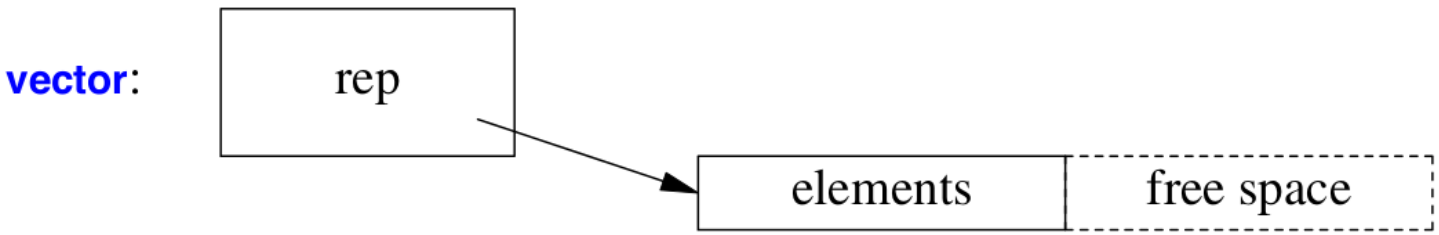
- Size is static and it may be a problem if the number of elements to be stored is not known in advance
- `insert` and `remove` operations are too expensive because elements are contiguously stored in memory

Dynamically-Resizeable Array



Dynamically-Resizeable Array

- Solves the problems encountered with `array`
- Handles runtinme resizing
- Better to use a **vector** if the input size is unknown.
  Use it unless you have a good reason not to



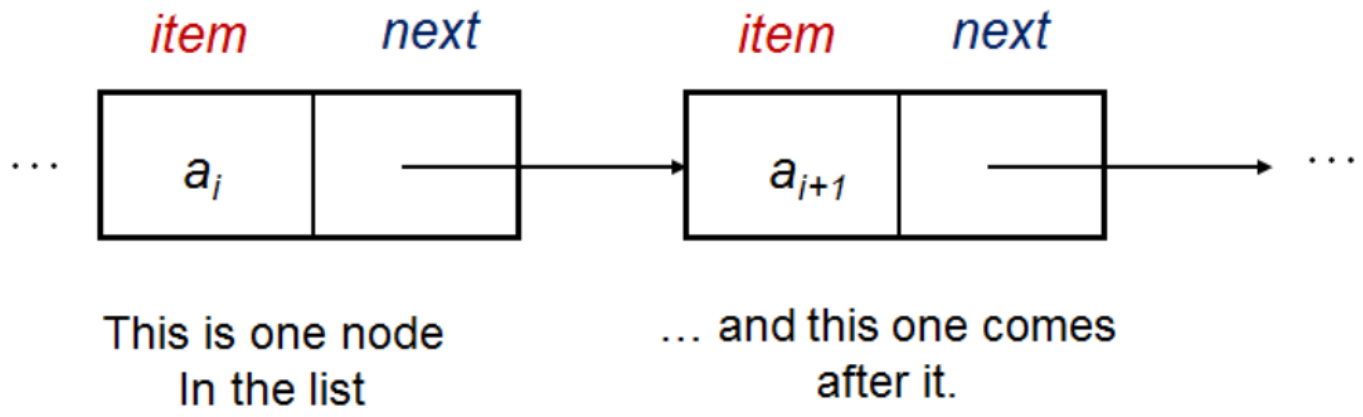
- **C++**: `STL::vector`
- `C++` code: `ch2_01_array_vector.cpp`

CPCFI

Dynamically-Resizeable Array: Operations

- **Sort** : https://visualgo.net/en/sorting
  - $O(n^2)$: slow, bad idea !
  - $O(n \log n)$: default choice in programming contests. Use `stable_sort` or `partial_sort` from C++ `STL algorithm` library
  - $O(n)$: can be used if the data has special properties
- **Search**
  - $O(n)$ Linear Search: avoid whenever as possible
  - $O(\log n)$ Binary Search: data must be sorted first in order to use this algorithm. Consider using C++ `STL algorithm` `binary_search()`, `lower_bound()` or `upper_bound()`
  - $O(1)$ using Hashes
- C++ code: `ch2_02_algorithm_collections.cpp`

Linked List



CPCFI

Linked List

- Uses pointers to store items non-contiguously in memory

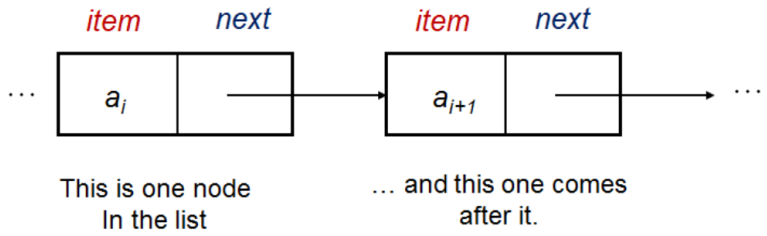


Figure: Linked list

```
1 struct Vertex {
2   int item;
3   Vertex* next;
4 };
```

In addition to this, we need to add two more pointers: `head` and `tail` that should point to the first and last elements of the linked list respectively

Linked List: Operations

- `get(i)`: starting from `head` pointer, iterate until index `i`
- `search(v)`: starting from `head` pointer, iterate until finding `item == v`
- Insertion `insert(i, v)`
  - At an empty LL
  - At the head of the LL (`i=0`)
  - In between the head and the tail (`i=[1,...,n-1]`)
  - At the tail of the LL (`i=n`)
- Removal `remove(i)`
  - At the head of the LL (`i=0`)
  - In between the head and the tail (`i=[1,...,n-2]`)
  - At the tail of the LL (`i=n-1`)

CPCFI

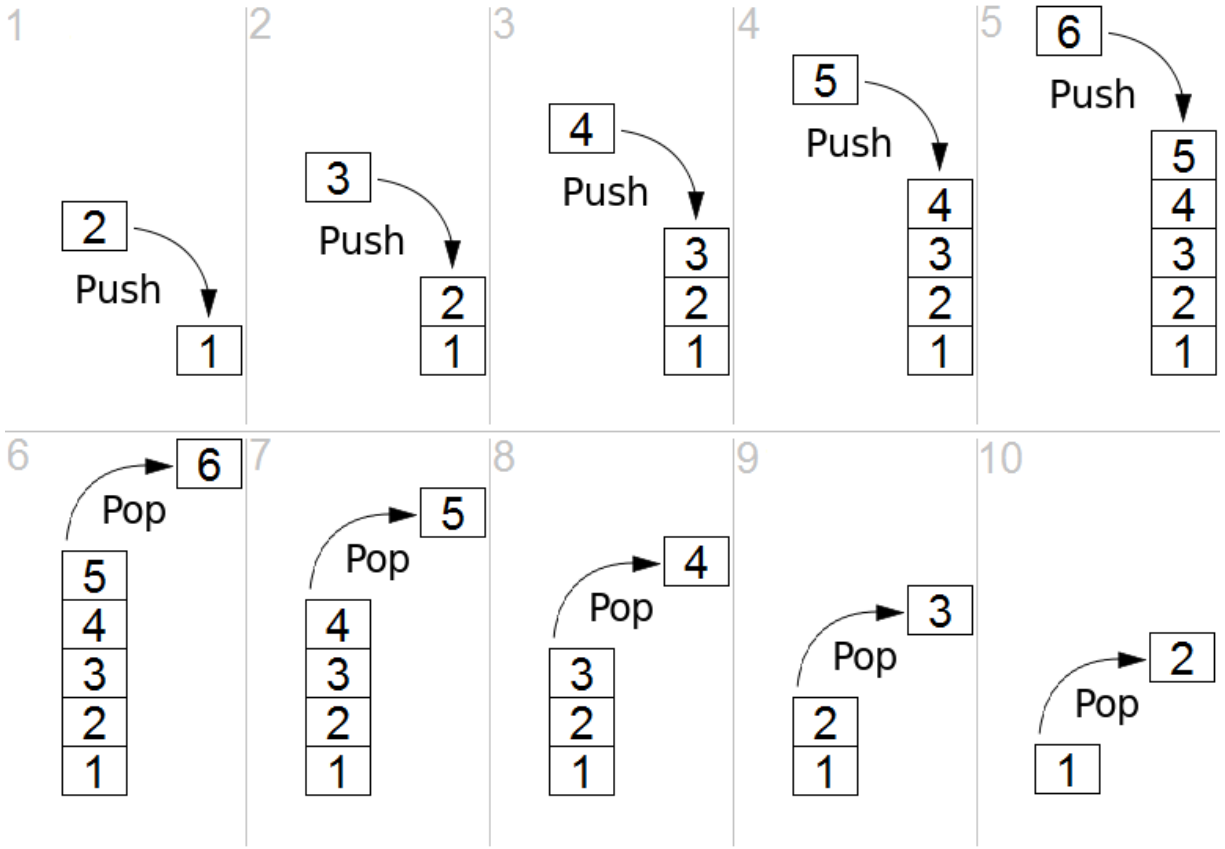
Linked List: Time Complexity

- `get(i)`: $O(n)$ since it should traverse through all the elements until item `i`
- `search(v)`: $O(n)$
- `insert(i, v)`: $O(n)$
- `remove(i)`: $O(n)$

Linked List: Final comments

- Inefficient to access elements
- Rare to use since a simple resizable compact array (vector) does the job better
- Excellent resizable data structure since it stores elements non contigously

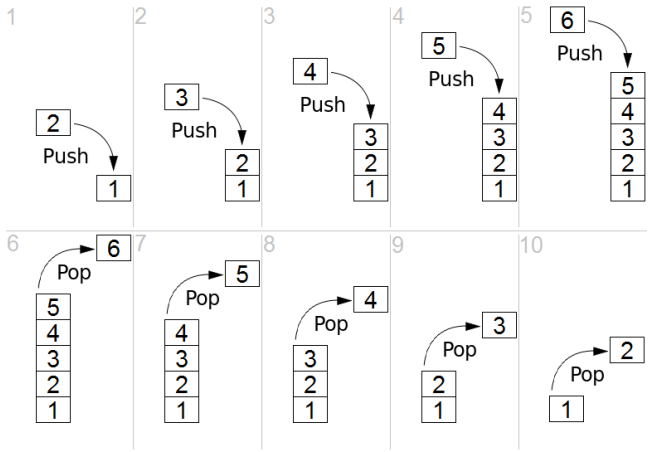
Stack


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
Pop
6
5
4
3
2
1
7
Pop
5
4
3
2
1
8
Pop
4
3
2
1
9
Pop
3
2
1
10
Pop
2
1

CPCFI

Stack

- ADT where the main operations are insertion (**push**) and removal (**pop**) from the top of the stack
  - Top: accessible
  - Bottom: inaccessible



- Last-In-First-Out (LIFO)
- Single Linked List where we can only perform insertions and deletions from the top

CPCFI

Stack: Applications

- Bracket Matching:

$$\{\ [\ (\ (\ )\ )\ )\ ]\ \}$$

$$(\ \{\ [\ \{\ [\ (\ )\ ]\ ]\ \}\ )$$

  Are the previous strings valid?

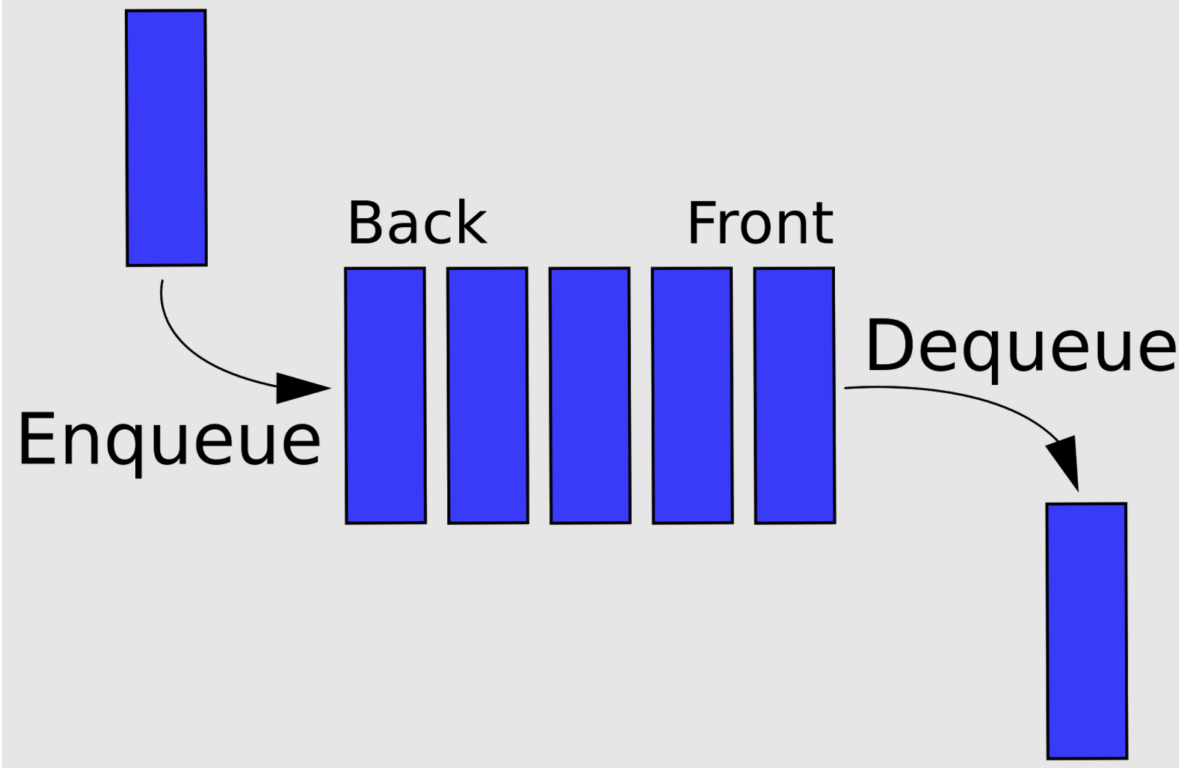
- Postfix calculator

$$2\ 3\ +\ 4\ *$$

  Is this string a valid sentence ? $\rightarrow (2+3)*4$

CPCFI

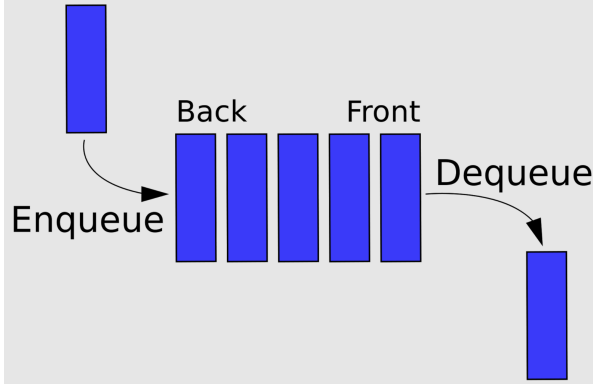
Queue



CPCFI

Queue

- ADT where items in the collection are kept in order and the main operations are addition of items to the back of the queue (**enqueue**) and removal of items from the front (**dequeue**) - FIFO



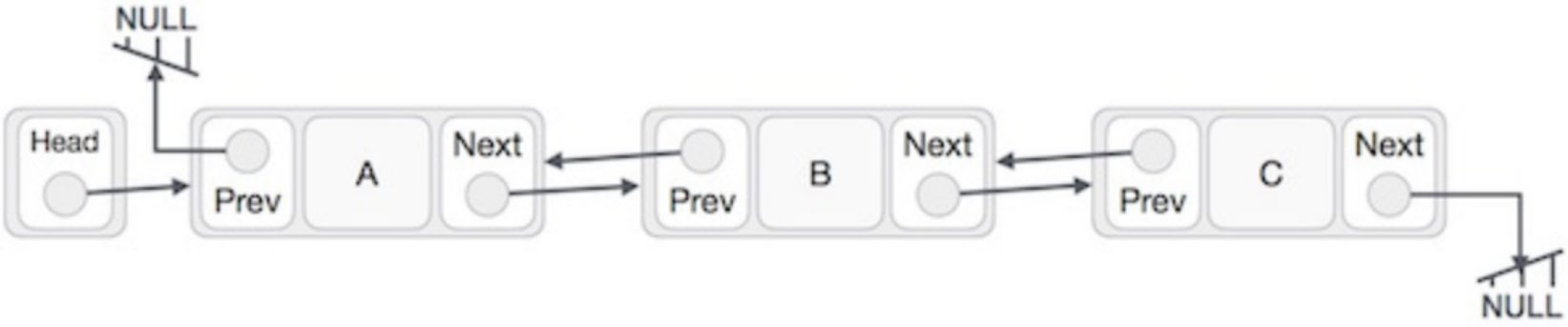
- Common implementations of a Queue are with a protected Singly Linked List where both enqueue and dequeue have $O(1)$ complexities
- `C++: STL queue`

CPCFI

Queue - Applications

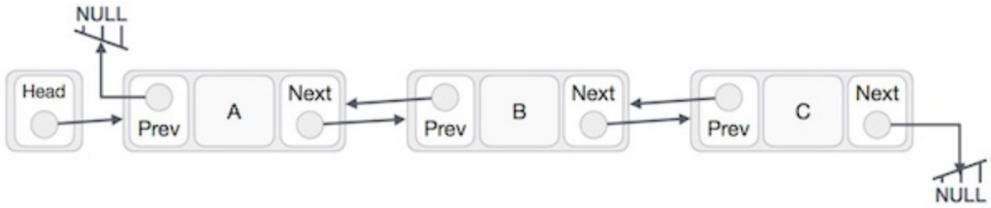
- Simulation of real queues
- Breadth-First Search (BFS) algorithm for graph traversal

Doubly Linked List



Doubly Linked List

- Each vertex contains two pointers:
  - Next
  - Prev



- The usage of **prev** makes it easier to move backwards but DLL occupies two times the memory of a singly LL
- `C++ STL: list`

CPCFI

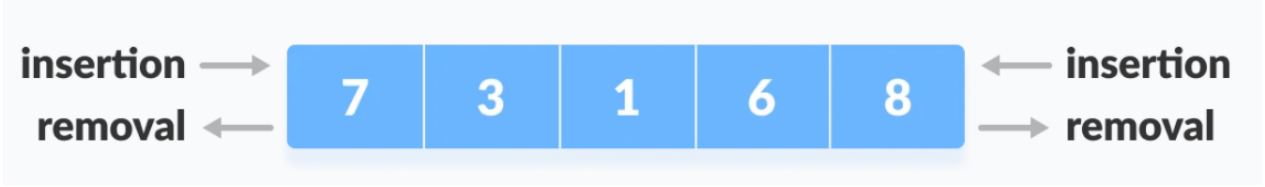
Doubly Linked List - Remove at tail

- One benefit of using **prev** pointer is the ability to move backwards, which makes the problem of removing the tail element $O(1)$. In a singly LL removing the tail element used linear time since we needed to iterate over the LL until finding the tail
- With **prev** pointer we have direct access to the tail element via the **tail** pointer and to the previous item via the **prev** pointer

```
1 Vertex *temp = tail;
2 tail = tail->prev;
3 tail->next = null;
4 delete temp;
```

CPCFI

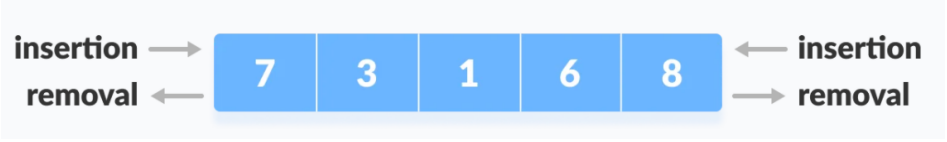
Deque



CPCFI

Deque

- Doubled Ended Queue. Pronounced "deck"
- ADT that generalizes a Queue for which elements can be added to or removed from either the front or the back



- Can be implemented as a protected DLL
- C++ code: `ch2_04_stack_queue.cpp`

CPCFI

Deque - Operations

- Query the head or tail
- Insert a new item to the head or tail
- Remove an existing item from the head or tail
- All operations are $O(1)$

Deque - Applications

- Finding the shortest paths 0/1-weighted graphs
- Sliding window techniques

Bitset

- Array that holds only boolean values
- `C++ STL bitset`

Bitmask

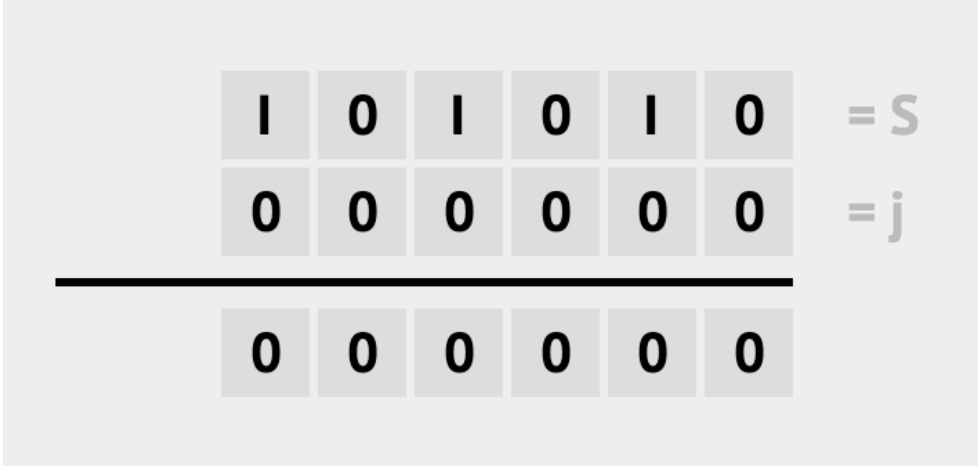


Figure: Input integer **S**. Bits of the mask **j**

CPCFI

Bitmask

- Provide an efficient way to manipulate a small set of Booleans
- S: Input; j: bitmask; O is a logical operator applied to S and j
- Some common operations $O(1)$:
  - Multiply and divide by i where i is a power of 2
  - Turn on the j_{th} item of the set
  - Turn off the j_{th} item of the set
  - Toggle the j_{th} item of the set
  - Turn on all the bits in a set of size n
  - Check if the j_{th} item is turned on
  - Get the value of the least significant bit (LSB)
  - Modulo
  - Determine if number m is power of 2
  - Find nearest power of 2
  - Turn off/on last bit/zero
  - Turn off/on last consecutive bits/zeroes
- C++ code: `ch2_03_bit_manipulation.cpp`

Summary

- Create operation is the same for LL, Stack, Queue, DLL, Deque
- Minor differences for the search/insert/remove operations
  - **Stack**: query/restricted-search, push/restricted-insert and pop/restricted-remove from the top
  - **Queue**: query/restricted-search and pop/restricted-remove from the front. Push/restricted-insert from the back
  - **Deque**: query/restricted-search, enqueue/restricted-insert, dequeue/restricted-remove from the front and back but not the middle
  - **SLL and DLL**: do not have these restrictions

CPCFI

Summary

Standard Container Operation Complexity					
	[] §31.2.2	List §31.3.7	Front §31.4.2	Back §31.3.6	Iterators §33.1.2
vector	const	O(n)+		const+	Ran
list		const	const	const	Bi
forward_list		const	const		For
deque	const	O(n)	const	const	Ran
stack				const	
queue			const	const	
priority_queue			O(log(n))	O(log(n))	
map	O(log(n))	O(log(n))+			Bi
multimap		O(log(n))+			Bi
set		O(log(n))+			Bi
multiset		O(log(n))+			Bi
unordered_map	const+	const+			For
unordered_multimap		const+			For
unordered_set		const+			For
unordered_multiset		const+			For
string	const	O(n)+	O(n)+	const+	Ran
array	const				Ran
built-in array	const				Ran
valarray	const				Ran
bitset	const				

Figure: *Front*: insertion/deletion before the first element. *Back*: insertion/deletion after the last element. *List*: insertions/deletion between front and back

Logarithm examples

Logarithm Examples					
n	16	128	1,024	16,384	1,048,576
log(n)	4	7	10	14	20
n*n	256	802,816	1,048,576	268,435,456	1.1e+12

Figure: "Don't mess with quadratic algorithms for larger values of n"

CPCFI

C++ LDS Implementations

Containers		
<vector>	One-dimensional resizable array	§31.4.2
<deque>	Double-ended queue	§31.4.2
<forward_list>	Singly-linked list	§31.4.2
<list>	Doubly-linked list	§31.4.2
<map>	Associative array	§31.4.3
<set>	Set	§31.4.3
<unordered_map>	Hashed associative array	§31.4.3.2
<unordered_set>	Hashed set	§31.4.3.2
<queue>	Queue	§31.5.2
<stack>	Stack	§31.5.1
<array>	One-dimensional fixed-size array	§34.2.1
<bitset>	Array of bool	§34.2.2

Figure: Part IV: STL, pg. 863 [Stroustrup]

UVa - Competitive Programming 3

University of Valladolid's Online Judge:
https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=622

- Linear DS with Built In Libraries

CPCFI

Table of Contents

Mathematics

- Most ICPC contests now include problems that need a math insight to be able to find a solution within the given time
- The next problems only need basic programming skills and some math fundamentals

Ad Hoc Math Problems

- Simpler Ones
- Simulation (Brute Force)
- Finding Pattern or Formula
- Grid
- Number Systems or Sequences
- Logarithm, Exponentiation, Power
- Polynomial
- Base Number Variants
- Ad Hoc

UVa - Competitive Programming 3

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=696

- Ad Hoc Mathematics Problems

Table of Contents

String Processing - Motivation

As the strings (e.g. DNA strings) that researchers deal with are usually (very) long, efficient string-specific data structures and algorithms are necessary

Basic String Processing Skills - #1

1. Given a text file that contains only alphabet characters [A-Za-z], digits [0-9], space, and period ('.'), write a program to read this text file line by line until we encounter a line that starts with seven periods ("....."). Concatenate (combine) each line into one long string `T`. When two lines are combined, give one space between them so that the last word of the previous line is separated from the first word of the current line

BSPS - #2

2. Suppose that we have one long string `T`. We want to check if another string `P` can be found in `T`. Report all the indices where `P` appears in `T` or report `-1` if `P` cannot be found in `T`

BSPS - #3

3. Suppose we want to do some simple analysis of the characters in `T` and also to transform each character in `T` into lowercase. The required analysis are: How many digits, vowels `[aeiouAEIOU]`, and consonants (other alphabets that are not vowels) are there in `T`? Can you do all these in $O(n)$ where n is the length of the string `T`?

CPCFI

BSPS - #4

4. Next, we want to break this one long string `T` into tokens (substrings) and store them into an array of strings called `tokens`. For this mini task, the delimiters of these tokens are spaces and periods (thus breaking sentences into words). Then, we want to sort this array of strings lexicographically and then find the lexicographically smallest string

CPCFI

BSPS - #5

5. Now, identify which word appears the most in `T`. In order to answer this query, we need to count the frequency of each word. Which data structure should be used for this mini task?

BSPS - #6

6. The given text file has one more line after a line that starts with '.....' but the length of this last line is not constrained. Your task is to count how many characters there are in the last line. How to read a string if its length is not known in advance?

- C++ code: `ch6_01_basic_string.cpp`

CPCFI

Table of Contents

Ad Hoc String Processing Problems

- Cipher/Encode/Encrypt/Decode/Decrypt
- Frequency Counting
- Input Parsing
- Solvable with Java String/Pattern class (Regular Expression)
- Output Formatting
- String Comparison
- Just Ad Hoc

UVa - Competitive Programming

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=737

- Ad Hoc String Processing Problems

Table of Contents

Overview & Motivation

- At least one geometry problem in ICPC
- These problems are usually left at the end because are the most time expensive problems and most tedious to code
- Main issues with geometry problems:
  - Most of them have tricky corner cases
  - Floating point precision errors
  - Tedious coding
  - Contestants forget some basic formulas and are unable to derive more complex formulas
  - Contestants do not prepare well-written library functions before the contests

CPCFI

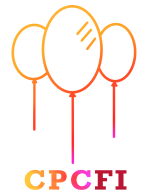
Basic Geometry Objects with Libraries

- $0D$ Objects: Points
- $1D$ Objects: Lines
- $2D$ Objects:
  - Circles
  - Triangles
  - Quadrilaterals

CPCFI

0D - Points

- Basic building block of higher dimensional geometry objects
- Represented with x and y coordinates with respect to the origin $(0, 0)$
- Features:
  - Sort points
  - Test if two points are equal
  - Measure the Euclidean distance between two points
  - Rotate a point by an angle θ counter clockwise around the origin



1D - Lines

- Set of points that satisfy the equation:

$$Ax + By + C = 0$$

- Vertical lines: $B = 1$
- Non-vertical lines: $B = 0$
- Features:
  - Compute the line equation given at least two points that pass through that line
  - Test whether two lines are parallel ($A == B$?)
  - Test whether two lines are the same (both lines must be parallel and their C coefficient must be equal)
  - Find the intersection point of two lines

CPCFI

1D - Lines

- **Line Segment**: line with two end points and finite length
- **Vector**: line segment with a direction
- More features:
  - Translate a point with respect to a vector
  - Compute the minimum distance from a point p and a line l
  - Compute the minimum distance from a point p and a line segment ab. We must consider the end points a and b
  - Compute the angle aob given three points
  - Determine whether a point r is on the left/right side of a line defined by two points p and q or whether the three points p, r, q are collinear
- C++ code: `ch7_01_points_lines.cpp`

2D - Circles

- **Circle**: centered at coordinate (a, b) with radius r is the set of all points (x, y) such that $(x - a)^2 + (y - b)^2 = r^2$
- Safest definition of π for a programming contest: $\text{pi} = \text{acos}(-1.0)$ or $\text{pi} = 2 * \text{acos}(0.0)$
- A circle with radius r:
  - Diameter $d = 2r$
  - Circumference $c = 2\pi r$
  - Area $A = \pi r^2$
- **Arc of a circle**: connected section of the circumference c of the circle.
  - Given a central angle α in degrees we can compute the length of the arc as:

$$\frac{\alpha}{360.0} \times c$$

2D - Circle

- **Chord of a circle**: line segment whose endpoints lie on the circle. Line of the chord:

$$\sqrt{2r^2 \times (1 - \cos(\alpha))}$$

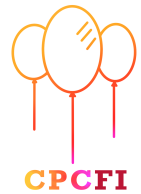
Another way:

$$2r \sin(\alpha/2)$$

- **Sector of a circle**: region of the circle enclosed by two radius and an arc lying between the two radius. Sector area:

$$\frac{\alpha}{360.0} \times A$$

- **Segment** of a circle: region of the circle enclosed by a chord and an arc lying between the chord's endpoints



2D - Circle: Features

- Check if a point is inside, outside or at the border of a circle
- Given two points p_1 and p_2 and radis r determine the location of the centers c_1 and c_2 of the two possible circles
- C++ code: `ch7_02_circles.cpp`

2D - Triangles

- **Triangle**: polygon with three vertices and three edges
  - **Equilateral**: three equal-length edges and all interior angles of 60 degrees
  - **Isosceles**: two edges have the same length and two interior angles are equal
  - **Scalene**: all edges have different lengths
  - **Right**: one of its interior angles has 90 degrees
- **Area**: $A = \frac{b \times h}{2}$
- **Perimeter**: $p = a + b + c$
- **Semiperimeter**: $s = \frac{p}{2}$
- **Heron's Formula**: $A = \sqrt{s \times (s-a) \times (s-b) \times (s-c)}$
- **Inscribed circle** of a triangle has radius $r = \frac{A}{s}$

CPCFI

2D - Triangles

- **Circumbrscribed circle**: radius $R = a \times b \times c/(4 \times A)$
- Check if three line segments a, b, c can form a triangle:

$$(a + b > c)\ \&\&(a + c > b)\ \&\&(b + c > a)$$

- If the three lengths are sorted, (a smallest and c the largest) we can simply check

$$(a + b > c)$$

- **Law of Cosines**: relates the lengths of its sides to the cosine of one of its angles

$$\gamma = a\cos(\frac{a^2 + b^2 - c^2}{2ab})$$

$$c^2 = a^2 + b^2 - 2ab\cos(\gamma)$$

CPCFI

2D - Triangles

- **Law of Sines**: relates the lengths of the sides of an arbitrary triangle to the sines of its angle.

$$\frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma} = 2R$$

  where R is the radius of the circumcircle
- **Pythagorean Theorem**: only applicable to the right triangles and is a specialization of the Law of Cosines where $\gamma = 90°$ and $\cos(\gamma) = 0$, thus $c^2 = a^2 + b^2$
- **Pythagoream Triple**: triple with three positive integers (a, b, c) such that $a^2 + b^2 = c^2$. This triple describes the integer lengths of the three sides of a right triangle.

2D - Triangles - Features

- Find the center of the incircle: meeting point between the triangle's angel bicectors
- Find the center of the circumcircle: meeting point between the triangle's perepdnicular bisectors
- C++ code: `ch7_03_triangles.cpp`

CPCFI

2D - Quadrilaterals

- **Quadrilateral**: polygon with four edges and four vertices
- **Rectangle**: polygon with four edges, four vertices and four right angles
  - Area: $A = w \times h$
  - Perimeter: $P = 2 \times (w + h)$
- **Square**: rectangle where $w = h$
- **Trapezium**: polygon with four edges, four vertices and one pair of parallel edges
  - Area: $A = \frac{1}{2} \times (w_1 + w_2) \times h$ where w_i is the length of the edge i from the parallel edges and h is the height between those parallel edges
  - **Isosceles Trapezium**: the two non-parallel sides have the same length

CPCFI

2D - Quadrilaterals

- **Parallelogram**: polygon with four edges and four vertices and the opposite sides must be parallel
- **Kite**: quadrilateral with two pairs of sides of the same length which are adjacent to each other. Area: $A = (D_1 \times D_2)/2$
- **Rhombus**: special parallelogram where each side has the same length and a special case of a kite where each side has the same length

CPCFI

UVa - Competitive Programming 3

https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=756

- Basic Geometry

References

- Halim S., Halim F., *Competitive Programming 3*, Handbook for ACM ICPC and IOI Contestants. 2013
- Stroustrup B. *The C++ Programming Language*. Fourth ed.
- Skiena S. *The Algorithm Design Manual*. Springer. 2020

CPCFI