**Leetcode Note by Yang**

[**Section 01. Bit Manipulation** 1](#_Toc10970570)

[**136. Single number I (E) [bit]** 1](#_Toc10970571)

[**137. Single number II (M) [bit]** 1](#_Toc10970572)

[**190. Reverse bits (E) [bit]** 3](#_Toc10970573)

[**191. Number of 1 bits (E) [bit]** 4](#_Toc10970574)

[**201. Bit wise AND of number range (M) [bit]** 5](#_Toc10970575)

[**231. Power of two (E) [bit, math]** 6](#_Toc10970576)

[**260. Single Number III (M) [bit]** 7](#_Toc10970577)

[**268. Missing number (M) [bit, array, math]** 8](#_Toc10970578)

[**287. Find the duplicate number (M) [bit]** 9](#_Toc10970579)

[**318. Maximum product of word lengths (M) [bit, string]** 10](#_Toc10970580)

[**338. Counting bits (M) [bit]** 11](#_Toc10970581)

[**342. Power of four (E) [bit]** 12](#_Toc10970582)

[**371. Sum of two integers (E) [bit]** 13](#_Toc10970583)

[**389. Find the difference (E) [bit, string]** 13](#_Toc10970584)

[**401. Binary watch (E) [bit]** 14](#_Toc10970585)

[**421. Maximum XOR of two number in an array (M) [bit, array]** 16](#_Toc10970586)

[**461. Hamming distance (E) [bit]** 17](#_Toc10970587)

[**477. Total hamming distance (M) [bit]** 18](#_Toc10970588)

[**645. Set mismatch (E) [array]** 18](#_Toc10970589)

[**693. Binary number with alternating bits (E) [bit]** 20](#_Toc10970590)

[**Section 02. Array** 22](#_Toc10970591)

[**015. 3 sum (M) [array, two pointers]** 22](#_Toc10970592)

[**016. 3 sum closest (M) [array, two pointers]** 23](#_Toc10970593)

[**018. 4 sum (M) [array, two pointers]** 25](#_Toc10970594)

[**026. Remove duplicates from sorted array (E) [array]** 26](#_Toc10970595)

[**027. Remove element (E) [array]** 27](#_Toc10970596)

[**031. Next permutation (M) [array, math]** 28](#_Toc10970597)

[**036. Valid Sudoku (M) [array]** 29](#_Toc10970598)

[**041. First missing positive (H)** 31](#_Toc10970599)

[**048. Rotate image (M)** 32](#_Toc10970600)

[**054. Spiral matrix (M) [array]** 33](#_Toc10970601)

[**059. Spiral matrix II (M) [array]** 35](#_Toc10970602)

[**066. Plus one (E) [array]** 36](#_Toc10970603)

[**073. Set matrix zeros (M) [array]** 37](#_Toc10970604)

[**080. Remove Duplicates from Sorted Array II (M) [array]** 39](#_Toc10970605)

[**118. Pascal’s triangle (E) [array]** 40](#_Toc10970606)

[**119. Pascal’s triangle II (E) [array]** 41](#_Toc10970607)

[**121. Best time to buy and sell stock (E) [array]** 42](#_Toc10970608)

[**128. Longest consecutive sequence (H) [array, hash map]** 43](#_Toc10970609)

[**169. Majority element (E) [array, hash map]** 44](#_Toc10970610)

[**189. Rotate array (E) [array]** 45](#_Toc10970611)

[**Section 03. String** 47](#_Toc10970612)

[**005. Longest palindromic substring (M)** 47](#_Toc10970613)

[**006. ZigZag conversion (E) [string, math]** 48](#_Toc10970614)

[**008. String to integer (E) [string]** 50](#_Toc10970615)

[**014. Longest common prefix (E) [string]** 52](#_Toc10970616)

[**028. Implement strStr() (E) [string]** 53](#_Toc10970617)

[**038. Count and say (E) [string]** 54](#_Toc10970618)

[**043. Multiply strings (M)** 55](#_Toc10970619)

[**058. Length of last word (E) [string]** 56](#_Toc10970620)

[**067. Add binary (E) [string]** 57](#_Toc10970621)

[**068. Text justification (H)** 58](#_Toc10970622)

[**125. Valid palindrome (E) [string, two pointers]** 59](#_Toc10970623)

[**165. Compare version numbers (E) [string]** 59](#_Toc10970624)

[**242. Valid anagram (E) [string, hash map]** 61](#_Toc10970625)

[**383. Ransom note (E) [string, hash map]** 62](#_Toc10970626)

[**405. Convert a number to hexadecimal (E) [string, bit]** 63](#_Toc10970627)

[**408. Valid word abbreviation (E) [string]** 64](#_Toc10970628)

[**415. Add strings (E) [string]** 65](#_Toc10970629)

[**434. Number of segments in a string (E) [string]** 66](#_Toc10970630)

[**443. String compression (E) [string, double pointers]** 67](#_Toc10970631)

[**Section 04. Linked List** 69](#_Toc10970632)

[**002. Add two numbers (M) [linked list]** 69](#_Toc10970633)

[**021. Merge two sorted lists (E) [linked list]** 70](#_Toc10970634)

[**024. Swap nodes in pairs (E) [linked list]** 72](#_Toc10970635)

[**083. Remove duplicate from sorted list (E) [linked list]** 73](#_Toc10970636)

[**Section 05. Stack** 76](#_Toc10970637)

[**020. Valid parentheses (E) [stack]** 76](#_Toc10970638)

[**032. Longest valid parentheses (H) [stack]** 76](#_Toc10970639)

[**101. Symmetric tree (E) [tree, stack]** 77](#_Toc10970640)

[**150. Evaluate reverse polish notation (M) [stack]** 79](#_Toc10970641)

[**155. Min stack (E) [stack]** 80](#_Toc10970642)

[**224. Basic calculator (H) [stack]** 81](#_Toc10970643)

[**227. Basic calculator II (M) [stack]** 82](#_Toc10970644)

[**232. Implement queue using stack (E) [stack]** 84](#_Toc10970645)

[**716. Max stack (E) [stack]** 86](#_Toc10970646)

[**739. Daily temperatures (M) [stack]** 88](#_Toc10970647)

[**770. Basic calculator IV (E) [stack]** 89](#_Toc10970648)

[**772. Basic calculator III (H) [recursion, string]** 89](#_Toc10970649)

[**Section 06. Queue** 92](#_Toc10970650)

[**225. Implement stack using queue (H) [queue]** 92](#_Toc10970651)

[**239. Sliding window maximum (H) [queue]** 93](#_Toc10970652)

[**281. Zigzag iterator (M) [queue]** 94](#_Toc10970653)

[**346. Moving average from data stream (E) [queue]** 95](#_Toc10970654)

[**Section 09. Hash table** 97](#_Toc10970655)

[**001. Two sum (E) [hash table, array]** 97](#_Toc10970656)

[**003. Longest substring without repeating characters (M) [hash table, string]** 97](#_Toc10970657)

[**049. Group anagrams (M) [hash table, string]** 98](#_Toc10970658)

[**159. Longest substring with at most two distinct characters (H) [hash table, string]** 99](#_Toc10970659)

[**170. Two sum III (E) [hash table]** 100](#_Toc10970660)

[**202. Happy number (E) [hash table, math]** 101](#_Toc10970661)

[**205. Isomorphic string (E) [hash table]** 102](#_Toc10970662)

[**217. Contains duplicate (E) [hash table, array]** 103](#_Toc10970663)

[**219. Contains duplicate (E) [hash table, array]** 103](#_Toc10970664)

[**249. Group shifted strings (H) [hash table, string]** 104](#_Toc10970665)

[**290. Word pattern (E) [hash table]** 105](#_Toc10970666)

[**340. Longest substring with at most K distinct characters (H) [hash table, string]** 106](#_Toc10970667)

[**Section 10. Math** 108](#_Toc10970668)

[**007. Reverse integer (E) [math]** 108](#_Toc10970669)

[**009. Palindrome number (E) [math]** 108](#_Toc10970670)

[**012. Integer to roman (M) [math]** 110](#_Toc10970671)

[**013. Roman to integer (E)** 111](#_Toc10970672)

[**168. Excel sheet column title (E)** 112](#_Toc10970673)

[**171. Excel sheet column number (E)** 113](#_Toc10970674)

[**172. Factorial trailing zeros (E)** 114](#_Toc10970675)

[**223. Rectangle area (M)** 115](#_Toc10970676)

[**Section 11. Two Pointers** 116](#_Toc10970677)

[**141. Linked list cycle (E) [two pointers, linked list]** 116](#_Toc10970678)

[**142. Linked list cycle II (M) [two pointers, linked list]** 116](#_Toc10970679)

[**167. Two sum II (E) [two pointers]** 118](#_Toc10970680)

[**Section 12. Sort** 120](#_Toc10970681)

[**088. Merge sorted array (E)** 120](#_Toc10970682)

[**252. Meeting room (E)** 121](#_Toc10970683)

[**973. K closest points to origin (E)** 122](#_Toc10970684)

[**Section 18. Back Tracking** 123](#_Toc10970685)

[**017. Letter combination of a phone number (M) [Back Tracking, string]** 123](#_Toc10970686)

[**038. Sudoku solver (H) [Back Tracking, DFS]** 124](#_Toc10970687)

[**039. Combination sum (M) [Back Tracking, DFS]** 126](#_Toc10970688)

[**040. Combination sum II (M) [Back Tracking, DFS]** 127](#_Toc10970689)

[**046. Permutations (M) [Back Tracking, DFS]** 129](#_Toc10970690)

[**047. Permutations II (M) [Back Tracking, DFS]** 130](#_Toc10970691)

[**077. Combination (M) [Back Tracking, DFS]** 131](#_Toc10970692)

[**078. Subsets (M) [Back Tracking, DFS]** 133](#_Toc10970693)

[**079. Word search (M) [Back Tracking, DFS]** 134](#_Toc10970694)

[**090. Subsets II (M) [Back Tracking, DFS]** 135](#_Toc10970695)

[**212. Word Search II (H) [Back Tracking, DFS]** 136](#_Toc10970696)

[**216. Combination sum III (M) [Back Tracking, DFS]** 138](#_Toc10970697)

[**784. Letter case permutation (E) [Back Tracking, DFS]** 139](#_Toc10970698)

[**Section 20. Design** 141](#_Toc10970699)

[**146. LRU cache (H) [double linked list, hash map]** 141](#_Toc10970700)

[**355. Design twitter (M) [hash map, unordered set, map]** 143](#_Toc10970701)

[**460. LFU cache (H) [hash map, list iterator]** 146](#_Toc10970702)

**Section 01. Bit Manipulation**

**136. Single number I (E) [bit]**

**Problem**: Given a non-empty array of integers, every element appears twice except for one. Find that single one.

**Example:** Input: [4, 1, 2, 1, 2]

Output: 4

**Analysis:** Solution1 used C++ xor operation.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <numeric>
4. #include <functional>
6. **class** Solution
7. {
8. **public**:
9. **int** singleNumber(std::vector<**int**>& nums)
10. {
11. **return** accumulate(nums.cbegin(), nums.cend(),
12. 0, std::bit\_xor<**int**>());
13. }
14. };
16. **class** Solution1
17. {
18. **public**:
19. **int** singleNumber(std::vector<**int**>& nums)
20. {
21. **int** result = nums[0];
22. **for** (**int** i = 1; i < nums.size(); i++)
23. {
24. result ^= nums[i];
25. }
26. **return** result;
27. }
28. };
30. **int** main()
31. {
32. std::vector<**int**> v{ 4, 1, 2, 1, 2};
33. Solution1 a;
34. **int** result = a.singleNumber(v);
35. std::cout << result << std::endl;
36. std::cin.get();
37. **return** 0;
38. }

**137. Single number II (M) [bit]**

**Problem:** Given a non-empty array of integers, every element appears three times except for one, which appears exactly once. Find that single one.

**Example:** Input: [2, 2, 3, 2]

Output: 3

**Note:** Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

**Analysis:** Assume *int* is a 32-bit binary variable. Since all the other elements appear to three times, it is reasonable to use 1-bit binary variable (B) *Ones* to define that element appears one time; use 1-bit binary variable (A) *Twos* to define that element appears two times. A is upper bit and B is lower bit. Then a state transition table can be summarized as followed:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A (*Twos*) | B (*Ones*) | C (*Input*) | D (*Next\_Twos*) | E *(Next\_Ones*) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |

This is the flow chart:

C=0

C=1

C=1

C=1

C=0

C=0

Karnaugh map for D (*Twos*): Karnaugh map for E (*Ones*):

|  |  |  |  |
| --- | --- | --- | --- |
| C \ AB | 00 | 01 | 10 |
| 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| C \ AB | 00 | 01 | 10 |
| 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** singleNumber(vector<**int**>& nums)
10. {
11. **int** ones = 0, twos = 0;
12. **for** (**size\_t** i = 0; i < nums.size(); i++)
13. {
14. **int** ones\_next = ((~twos)&(~ones)&(nums[i])) | ((~twos)&(ones)&(~nums[i]));
15. **int** twos\_next = ((~twos)&(ones)&(nums[i])) | ((twos)&(~ones)&(~nums[i]));
17. ones = ones\_next;
18. twos = twos\_next;
19. }
20. **return** ones;
21. }
22. };
24. **int** main()
25. {
26. vector<**int**> v{ 2, 2, 3, 2 };
27. Solution a;
28. **int** result = a.singleNumber(v);
29. cout << result << endl;
30. cin.get();
31. **return** 1;
32. }

**190. Reverse bits (E) [bit]**

**Problem:** Reverse bits of a given 32 bits unsigned integer

**Example:** Input: 00000010100101000001111010011100

Output: 00111001011110000010100101000000

**Analysis:** first,get one lowest (right) bit from input. If it is 1, left shift one bit result and plus 1; if it is zero, left shift one bit result. Then right shift one bit input and repeat the process. Following is the flow chat:

Lowest bit == 1

No

Yes

Input (uint\_32t)

result left shift 1 bit

result + 1

result left shift 1 bit

result left shift 1 bit

**Solution:**

1. #include <stdint.h>
2. #include <iostream>
3. #include <bitset>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. uint32\_t reverseBits(uint32\_t input)
10. {
11. **size\_t** count = 32;
12. uint32\_t result = 0;
13. **for** (**size\_t** i = 0; i < count; i++)
14. {
15. **if** (input & 1 == 1)
16. {
17. result = result << 1;
18. result++;
19. }
20. **else**
21. {
22. result = result << 1;
23. }
24. input = input >> 1;
25. }
26. **return** result;
27. }
28. };
30. **int** main()
31. {
32. Solution a;
33. uint32\_t input = 55;
34. uint32\_t result = a.reverseBits(input);
35. cout << "Input is " << bitset<32>(input) << endl;
36. cout << "Output is " << bitset<32>(result) << endl;
37. cin.get();
38. **return** 1;
39. }

**Caution:**

* For result, first left shift 1 bit, then plus one (think about the last bit!).

**191. Number of 1 bits (E) [bit]**

**Problem:** Write a function that takes an unsigned integer and return the number of '1' bits it has (also known as the Hamming weight).

**Example:** Input: 00000000000000000000000000001011

Output: 3 **Analysis:** Determine the lowest bit of input: if it is 1, sum plus 1; if it is 0, nothing. Input right shift 1 bit. Then repeat this process for all bit of input.

Improvement would be consider n&(n-1). This operation will remove the rightmost 1 in n. This will reduce the iteration times from 32 to sum.

**Solution:**

1. #include <stdint.h>
2. #include <iostream>
3. #include <bitset>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. uint32\_t numberof1Bits(uint32\_t input)
11. {
12. uint32\_t result = 0;
13. **while** (input != 0)
14. {
15. input = input & (input - 1);
16. result++;
17. }
18. **return** result;
19. }
20. };
22. **void** main()
23. {
24. Solution a;
25. uint32\_t input = 55;
26. uint32\_t result = a.numberof1Bits(input);
28. cout << "Input is " << bitset<32>(input) << endl;
29. cout << "Number of 1 bits is: " << result << endl;
30. cin.get();
31. }

**201. Bit wise AND of number range (M) [bit]**

**Problem:** Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

**Example:** Input: [5,7]

Output: 4

**Analysis:** similar to 191…

**Solution:**

1. #include <iostream>
2. #include <stdint.h>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. uint32\_t bitWiseAndofNumberRange(uint32\_t m, uint32\_t n)
10. {
11. **while** (m < n)
12. {
13. n &= n - 1;
14. }
15. **return** n;
16. }
17. };
19. **void** main()
20. {
21. Solution a;
22. uint32\_t m = 5, n = 7;
23. uint32\_t result = a.bitWiseAndofNumberRange(m, n);
24. cout << "Result is: " << result << endl;
25. cin.get();
26. }

**231. Power of two (E) [bit, math]**

**Problem:** Given an integer, write a function to determine if it is a power of two.

**Example:** Input: 1

Output: true

Explanation: 12 = 1

**Analysis:** Similar to 191…

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **bool** isPowerofTwo(**int** input)
9. {
10. **return** (input > 0) && ((input&(input - 1)) == 0);
11. }
12. };
14. **int** main()
15. {
16. Solution a;
17. **int** input = 4;
18. **bool** result = a.isPowerofTwo(input);
19. cout << input << " is power of two: " << boolalpha << result << endl;
20. cin.get();
21. **return** 1;
22. }

**Caution:** line 10, (input&(input - 1)) == 0, ==higher priority than &

**260. Single Number III (M) [bit]**

**Problem:** Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

**Example:** Input: [1,2,1,3,2,5]

Output: [3,5]

**Note:** 1. The order of the result is not important. So in the above example, [5, 3] is also correct.

2. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

**Analysis:**

1. First, xor for all the elements;
2. Get the last bit where 1 occurs by “x & ~(x-1)”. Because –(x-1) = ~(x-1)+1 ⬄ -x = ~(x-1), the last bit can also calculated by x & -x;
3. Get one subset where the number has the bit and get the other subset where the number doesn’t have the bit
4. Xor the two subsets to get the two numbers

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<**int**> singleNumber(vector<**int**>& input)
10. {
11. // store xor result
12. **int** temp = 0;
13. **for** (**int** i = 0; i < input.size(); i++)
14. {
15. temp ^= input[i];
16. }
18. // get last different bit position
19. **int** diff = temp &(~(temp - 1));
21. vector<**int**> result(2, 0);
22. **for** (**int** j = 0; j < input.size(); j++)
23. {
24. **if** (diff&input[j])
25. result[0] ^= input[j];
26. **else**
27. result[1] ^= input[j];
28. }
29. **return** result;
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. vector <**int**> input{1, 2, 1, 3, 2, 5};
37. vector<**int**> result = a.singleNumber(input);
38. cout << "The results are: " << result[0] << ", " << result[1] << endl;
39. cin.get();
40. **return** 0;
41. }

**268. Missing number (M) [bit, array, math]**

**Problem:** Given an array containing n distinct numbers taken from 0, 1, 2, .., n, find the one that is missing from the array.

**Example:** Input: [3, 0, 1]

Output: 2

**Note:** Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

**Analysis:**

Solution first calculate the sum from 0-n; then subtract the sum of all elements from the result; the left would be the missing number.

Solution1 performs xor operation for all elements and 0 – n numbers; then the only one left would be the missing number.

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** missingNumber(vector<**int**>& input)
10. {
11. **int** sum = 0;
12. **for** (**int** i = 0; i < input.size(); i++)
13. {
14. sum += input[i];
15. }
16. **return** 0.5\*input.size()\*(input.size() + 1) - sum;
17. }
18. };
20. **class** Solution1
21. {
22. **public**:
23. **int** missingNumber(vector<**int**>& input)
24. {
25. **int** result = 0;
26. **for** (**int** i = 0; i < input.size(); i++)
27. {
28. result ^= (i + 1) ^ input[i];
29. }
30. **return** result;
31. }
32. };
34. **int** main()
35. {
36. Solution1 a;
37. vector<**int**> input{ 1, 0, 3 };
38. **int** result = a.missingNumber(input);
39. cout << "The missing number is: " << result << endl;
40. cin.get();
41. **return** 0;
42. }

**287. Find the duplicate number (M) [bit]**

**Problem:** Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

**Note:**

* You must not modify the array (assume the array is read only).
* You must use only constant, O(1) extra space.
* Your runtime complexity should be less than O(n2).
* There is only one duplicate number in the array, but it could be repeated more than once.

**Example:** Input: [1,3,4,2,2]

Output: 2

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** findDuplicate(vector<**int**>& nums)
6. {
7. **int** res = 0;
8. **for** (**int** i = 0; i < 32; i++)
9. {
10. **int** bit = (1 << i);
11. **int** cnt1 = 0, cnt2 = 0;
12. **for** (**int** k = 0; k < nums.size(); k++)
13. {
14. **if** ((k+1) & bit) cnt1++;
15. **if** (nums[k] & bit) cnt2++;
16. }
17. **if** (cnt2 > cnt1) res += bit;
18. }
19. **return** res;
20. }
22. **int** main()
23. {
24. vector<**int**> nums = { 1, 3, 4, 5, 5 };
25. cout << findDuplicate(nums);
26. system("pause");
27. **return** 1;
28. }

**318. Maximum product of word lengths (M) [bit, string]**

**Problem:** Given a string array words, find the maximum value of length(word[i]) \* length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

**Example:** Input: ["abcw","baz","foo","bar","xtfn","abcdef"]

Output: 16

Explanation: The two words can be "abcw", "xtfn".

**Analysis:** This problem defines only small letter from a to z. 32 bit int data type can be used to represent each word. The lower 26 of 32 bits of int are used. For example:

abcd -> 0000 0000 0000 0000 0000 0000 0000 1111

acxy -> 0000 0001 1000 0000 0000 0000 0000 0101

1. Claim a int vector to store each word;
2. Use AND operation to determine each two int pair are same (not same -> 0);
3. Compare and return the maximum value;

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. #include <algorithm>
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. **int** maxLength(vector<string>& input)
12. {
13. vector<**int**> temp(input.size(), 0);
14. **int** result = 0;
15. **for** (**int** i = 0; i < input.size(); i++)
16. {
17. **for** (**int** j = 0; j < input[i].size(); j++)
18. {
19. temp[i] |= (1 << (input[i][j] - 'a'));
20. //cout << input[i][j];
21. }
23. **for** (**int** k = 0; k < i; k++)
24. {
25. **if** ((temp[i] & temp[k]) == 0) // (temp[i] & temp[k])!!!
26. {
27. result = max(result, **static\_cast**<**int**>(input[i].size()\*input[k].size()));
28. }
29. }
30. }
31. **return** result;
32. }
33. };
35. **int** main()
36. {
37. Solution a;
38. vector<string> input{ "abcw", "baz", "foo", "bar", "xtfn", "abcdef" };
39. **int** result = a.maxLength(input);
40. cout << "The result is: " << result << endl;
41. cin.get();
42. **return** 0;
43. }

**Caution:**

* In line 25, since “==” has higher precedence than “&”, brackets are must!
* To use std::max(), include header <algorithm>

**338. Counting bits (M) [bit]**

**Problem:** Given a non negative integer number num. For every numbers i in the range 0 ≤ i ≤ num calculate the number of 1's in their binary representation and return them as an array.

**Example:** Input: 5

Output: [0,1,1,2,1,2]

**Follow up:**

* It is very easy to come up with a solution with run time O(n\*sizeof(integer)). But can you do it in linear time O(n) /possibly in a single pass?
* Space complexity should be O(n).
* Can you do it like a boss? Do it without using any builtin function like \_\_builtin\_popcount in c++ or in any other language.

**Solution:**

1. #include <vector>
2. #include <iostream>
3. **using** **namespace** std;
5. vector<**int**> countingBits(**int** num)
6. {
7. vector<**int**> res(num + 1, 0);
8. **for** (**int** i = 1; i <= num; i++)
9. {
10. res[i] = res[i&(i - 1)] + 1;
11. }
12. **return** res;
13. }
15. **int** main()
16. {
17. vector<**int**> res = countingBits(5);
18. **for**(**int** i = 0; i < res.size(); i++)
19. {
20. cout << res[i] << " ";
21. }
22. system("pause");
23. **return** 1;
24. }

**342. Power of four (E) [bit]**

**Problem:** Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

**Example:** Input: 16

Output: true

**Analysis:** similar to 231

**Solution:**

1. #include <iostream>
2. #include <math.h>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **bool** powerOfFour(**int** data)
9. {
10. **if** ((data > 0) && ((data&(data-1)) == 0) && ((data-1)%3 == 0))
11. {
12. **return** **true**;
13. }
14. **else** **return** **false**;
15. }
16. };
18. **int** main()
19. {
20. Solution a;
21. cout << a.powerOfFour(16) << endl;
22. system("pause");
23. **return** 1;
24. }

**371. Sum of two integers (E) [bit]**

**Problem:** Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -

**Example:** Input: a = 1, b = 2

Output: 3

**Analysis:** easy…

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **int** sumTwoInt(**int** a, **int** b)
9. {
10. **while**(b)
11. {
12. **int** carry = (a & b) << 1;
13. **int** add = a ^ b;
14. a = add;
15. b  = carry;
16. }
17. **return** a;
18. }
19. };
21. **int** main()
22. {
23. Solution a;
24. cout << a.sumTwoInt(11, -1) << endl;
25. system("pause");
26. **return** 1;
27. }

**389. Find the difference (E) [bit, string]**

**Problem:** Given two strings s and t which consist of only lowercase letters. String t is generated by random shuffling string s and then add one more letter at a random position. Find the letter that was added in t.

**Example:** Input: s = "abcd" t = "abcde"

Output: e

**Analysis:** similar to 268

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **char** findDifference(string s, string t)
9. {
10. **int** res = t[0] - 'a';
11. **for**(**int** i = 0; i < s.size(); i++)
12. {
13. res ^= (s[i] - 'a') ^ (t[i + 1] - 'a');
14. }
15. **return** res + 'a';
16. }
17. };
19. **int** main()
20. {
21. Solution a;
22. cout << a.findDifference("abcd", "bdeca") << endl;
23. system("pause");
24. **return** 1;
25. }

**401. Binary watch (E) [bit]**

**Problem:** A binary watch has 4 LEDs on the top which represent the hours (0-11), and the 6 LEDs on the bottom represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

Given a non-negative integer n which represents the number of LEDs that are currently on, return all possible times the watch could represent.

**Example:** Input: n = 1

Return: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16", "0:32"]

**Note:**

* The order of output does not matter.
* The hour must not contain a leading zero, for example "01:00" is not valid, it should be "1:00".
* The minute must be consist of two digits and may contain a leading zero, for example "10:2" is not valid, it should be "10:02".

****

For example, the above binary watch reads "3:25".

**Analysis:** easy…

**Solution:**

1. #include <iostream>
2. #include <bitset>
3. #include <string>
4. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. vector<string> binaryWatch(**int** n)
11. {
12. vector<string> res;
13. **for** (**int** h = 0; h < 12; h++)
14. {
15. **for** (**int** m = 0; m < 60; m++)
16. {
17. **if** (bitset<10>(h << 6 | m).count() == n)
18. {
19. res.push\_back(to\_string(h) + (m < 10 ? ":0" : ":") + to\_string(m));
20. }
21. }
22. }
23. **return** res;
24. }
25. };

28. **int** main()
29. {
30. Solution a;
31. vector<string> res = a.binaryWatch(1);
32. **for** (**int** i = 0; i < res.size(); i++)
33. cout << res[i] << ", ";
34. system("Pause");
35. **return** 1;
36. }

**421. Maximum XOR of two number in an array (M) [bit, array]**

**Problem:** Given a non-empty array of numbers, a0, a1, a2, … , an-1, where 0 ≤ ai < 231. Find the maximum result of ai XOR aj, where 0 ≤ i, j < n.

Could you do this in O(n) runtime?

**Example:** Input: [3, 10, 5, 25, 2, 8]

Output: 28

**Solution:**

1. #include <vector>
2. #include <iostream>
3. #include <unordered\_set>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** maxXor(vector<**int**>& num)
10. {
11. **int** res = 0, mask = 0;
12. **for** (**int** i = 31; i >= 0; i--)
13. {
14. mask |= (1 << i);
15. unordered\_set<**int**> s;
16. **for** (**int** val : num)
17. {
18. s.insert(mask&val);
19. }
21. **int** t = res | (1 << i);
22. **for** (**int** val : s)
23. {
24. **if** (s.count(t^val))
25. {
26. res = t;
27. **break**;
28. }
29. }
30. }
31. **return** res;
32. }
33. };
35. **int** main()
36. {
37. vector<**int**> num{ 3, 10, 5, 25, 2, 8 };
38. Solution a;
39. cout << a.maxXor(num) << endl;
40. system("pause");
41. **return** 1;
42. }

**461. Hamming distance (E) [bit]**

**Problem:** The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, calculate the Hamming distance.

Note: 0 ≤ x, y < 231

**Example:** Input: x = 1, y = 4

Output: 2

**Analysis:** similar to 191

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **int** hammingDistance(**int** a, **int** b)
9. {
10. **int** temp = a ^ b;
11. **int** result = 0;
12. **while** (temp)
13. {
14. temp = temp & (temp - 1);
15. result++;
16. }
17. **return** result;
18. }
19. };
21. **int** main()
22. {
23. Solution a;
24. cout << a.hammingDistance(1, 4) << endl;
25. system("pause");
26. **return** 1;
27. }

**477. Total hamming distance (M) [bit]**

**Problem:** TheHamming distance between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

**Example:** Input: 4, 14, 2

Output: 6

**Note:**

* Elements of the given array are in the range of 0 to 10^9
* Length of the array will not exceed 10^4.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** totalHammingDistance(vector<**int**>& nums)
6. {
7. **int** res = 0;
8. **for** (**int** i = 0; i < 32; i++)
9. {
10. **int** cnt = 0;
11. **for** (**int** j = 0; j < nums.size(); j++)
12. {
13. **if** (nums[j] & (1 << i)) cnt++;
14. }
15. res += (cnt\*(nums.size() - cnt));
16. }
17. **return** res;
18. }
20. **int** main()
21. {
22. vector<**int**> nums{ 4, 14, 2 };
23. cout << totalHammingDistance(nums);
24. system("pause");
25. **return** 1;
26. }

**645. Set mismatch (E) [array]**

**Problem:** The set S originally contains numbers from 1 to n. But unfortunately, due to the data error, one of the numbers in the set got duplicated to another number in the set, which results in repetition of one number and loss of another number.

Given an array nums representing the data status of this set after the error. Your task is to firstly find the number occurs twice and then find the number that is missing. Return them in the form of an array.

**Example:** Input: nums = [1,2,2,4]

Output: [2,3]

**Analysis:** similar to 268

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. vector<**int**> setMismatch(vector<**int**>& input)
9. {
10. vector<**int**> res;
11. **int** dup, mis;
12. **for** (**int** i = 0; i < input.size(); i++)
13. {
14. **for** (**int** j = i + 1; j < input.size(); j++)
15. {
16. **if** (input[j] == input[i])
17. {
18. dup = input[j];
19. }
20. }
21. }
22. res.push\_back(dup);
24. **int** sum = 0;
25. **for** (**int** i = 0; i < input.size(); i++)
26. {
27. sum += input[i];
28. }
29. mis = (1 + input.size())\*input.size() / 2 - (sum - dup);
30. res.push\_back(mis);
32. **return** res;
33. }
34. };
36. **int** main()
37. {
38. vector<**int**> input{ 1, 2, 2, 4 };
39. Solution a;
40. vector<**int**> res = a.setMismatch(input);
42. cout << res[0] << ", " << res[1] << endl;
43. system("Pause");
44. **return** 1;
45. }

**693. Binary number with alternating bits (E) [bit]**

**Problem:** Given a positive integer, check whether it has alternating bits: namely, if two adjacent bits will always have different values.

**Example:** Input: 5

Output: True

**Analysis:**

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **bool** alternatingBits(**int** num)
9. {
10. **int** bit = -1;
11. **while** (num)
12. {
13. **if** (num & 1)
14. {
15. **if** (bit == 1) **return** **false**;
16. **else** bit = 1;
17. }
18. **else**
19. {
20. **if** (bit == 0) **return** **false**;
21. **else** bit = 0;
22. }
23. num >>= 1;
24. }
25. **return** **true**;
26. }
27. };
29. **int** main()
30. {
31. Solution a;
32. cout << a.alternatingBits(5) << endl;
33. system("pause");
34. **return** 1;
35. }

**Section 02. Array**

**015. 3 sum (M) [array, two pointers]**

**Problem:** Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

**Example:** Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:

[ [-1, 0, 1],

[-1, -1, 2] ]

**Note:** The solution set must not contain duplicate triplets.

**Analysis:**

1. Sort nums in increasing order;
2. Start the loop from 0 – nums.size()-2;
3. If it is a positive number, break; because this is an order sequence;
4. Add condition, if repetition, continue;
5. Define a target = 0 – nums[i], then find another two elements whose sum equals to the target;
6. Define two “pointers”, one at the next position after target, and the last position after target;

* If this two numbers add up to the target, push back; also need to avoid repetition;
* If this two numbers’ sum smaller than target, left pointer +1
* If this two number’s sum larger than target, right pointer - 1

1. Return the result.

**Solution:**

1. #include <vector>
2. #include <iostream>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. vector<vector<**int**>> threeSum(vector<**int**>& input)
11. {
12. vector<vector<**int**>> res;
13. If (input.size() < 3) return {};
14. sort(input.begin(), input.end());
16. **for** (**int** i = 0; i < input.size() - 2; i++)
17. {
18. **if** (input[i] > 0) **break**;
20. **if** (i > 0 && (input[i] == input[i - 1])) **continue**;
22. **int** target = 0 - input[i];
23. **int** m = i + 1, n = input.size() - 1;
24. **while** (m < n)
25. {
26. **if** (input[m] + input[n] == target)
27. {
28. res.push\_back({ input[i], input[m], input[n] });
29. **while** ((m < n) && (input[m] == input[m + 1])) m++;
30. **while** ((m < n) && (input[n] == input[n - 1])) n--;
31. m++;
32. n--;
33. }
34. **else** **if** (input[m] + input[n] < target)
35. {
36. m++;
37. }
38. **else**
39. n--;
40. }
41. }
42. **return** res;
43. }
44. };
46. **int** main()
47. {
48. Solution a;
49. vector<**int**> input{ -1, 0, 1, 2, -1, -4 };
50. vector<vector<**int**>> res = a.threeSum(input);
51. cout << "The result is: " << endl;
52. **for** (**int** i = 0; i < res.size(); i++)
53. {
54. **for** (**int** j = 0; j < res[i].size(); j++)
55. {
56. cout << res[i][j] << ", ";
57. }
58. cout << endl;
59. }
60. cin.get();
61. **return** 0;
62. }

**016. 3 sum closest (M) [array, two pointers]**

**Problem:** Given an array nums of n integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

**Example:** Given array nums = [-1, 2, 1, -4], and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

**Analysis:** Similar to 015…

**Solution:**

1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** threeSumClosest(vector<**int**> input, **int** target)
11. {
12. sort(input.begin(), input.end());
13. if(input.size() < 3) return 0;
14. **int** min\_diff = numeric\_limits<**int**>::max();
15. **int** res = numeric\_limits<**int**>::max();
16. **for** (**int** i = 0; i < input.size() - 2; i++)
17. {
18. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
20. **int** c = target - input[i];
21. **int** a = i + 1, b = input.size() - 1;
22. **while** (a < b)
23. {
24. **int** sum = input[a] + input[b];
25. **if** (sum == c)
26. {
27. **return** target;
28. }
29. **else** **if** (sum < c)
30. {
31. ++a;
32. }
33. **else**
34. {
35. --b;
36. }
38. **if** (abs(sum - c) < min\_diff)
39. {
40. min\_diff = abs(sum - c);
41. res = sum + input[i];
42. }
43. }
44. }
46. **return** res;
47. }
48. };
50. **int** main()
51. {
52. vector<**int**> input{ -1, 2, 1, -4 };
53. **int** target = 1;
54. Solution a;
55. **int** res = a.threeSumClosest(input, target);
56. cout << res << endl;
57. cin.get();
58. **return** 0;
59. }

**Caution:**

* Line 24 must! Or m, n will change!!!
* If want to return the vector, determine the abs() and diff first.

**018. 4 sum (M) [array, two pointers]**

**Problem:** Given an array nums of n integers and an integer target, are there elements a, b, c, and d in nums such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

**Example:** Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

A solution set is: [ [-1, 0, 0, 1],

[-2, -1, 1, 2],

[-2, 0, 0, 2] ]

**Note:** The solution set must not contain duplicate quadruplets.

**Analysis:** similar to 015…

**Solution:**

1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. vector<vector<**int**>> fourSum(vector<**int**>& input, **int** target)
11. {
12. sort(input.begin(), input.end());
14. vector<vector<**int**>> res{};
15. **for** (**int** i = 0; i < input.size() - 3; i++)
16. {
17. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
18. **for** (**int** j = i + 1; j < input.size() - 2; j++)
19. {
20. **if** ((j > i + 1) && (input[j] == input[j - 1])) **continue**;
21. **int** a = j + 1, b = input.size() - 1;
22. **while** (a < b)
23. {
24. **int** sum = input[i] + input[j] + input[a] + input[b];
25. **if** (sum == target)
26. {
27. res.push\_back({ input[i], input[j], input[a], input[b] });
28. **while** ((a < b) && (input[a] == input[a + 1])) a++;
29. **while** ((a < b) && (input[b] == input[b - 1])) b--;
30. a++;
31. b--;
32. }
33. **else** **if** (sum < target)
34. a++;
35. **else**
36. b--;
37. }
38. }
39. }
40. **return** res;
41. }
42. };
44. **int** main()
45. {
46. Solution a;
47. vector<**int**> input{ 0, 0, 0, 0 }; // {0, 0, 0, 0}
48. **int** target = 0;
49. vector<vector<**int**>> res{};
50. res = a.fourSum(input, target);
51. **for** (**int** i = 0; i < res.size(); i++)
52. {
53. **for** (**int** j = 0; j < res[i].size(); j++)
54. {
55. cout << res[i][j] << " ";
56. }
57. cout << endl;
58. }
59. system("pause");
60. **return** 0;
61. }

**Caution:**

* i loops from 0 – input.size()-3; j loops from i+1 – input.size()-2.
* Three places to add to exclude duplicates.

**026. Remove duplicates from sorted array (E) [array, two pointers]**

**Problem:** Given a sorted array nums, remove the duplicates in-place such that each element appear only once and return the new length.

**Example:** Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.

**Note:** Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

**Analysis:** easy…

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** removeDuplicates(vector<**int**>& input)
10. {
11. **int** sum = 0;
12. **for** (**int** i = 0; i < input.size(); i++)
13. {
14. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
15. sum++;
16. }
17. **return** sum;
18. }
19. };
21. **int** main()
22. {
23. vector<**int**> input{ 0,0,1,1,1,2,2,3,3,4 };
24. **int** res = 0;
25. Solution a;
26. res = a.removeDuplicates(input);
27. cout << res << endl;
28. cin.get();
29. **return** 0;
30. }

**027. Remove element (E) [array, two pointers]**

**Problem:** Given an array nums and a value val, remove all instances of that value in-place and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

**Example:** Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

**Analysis:** Easy…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** removeElement(vector<**int**>& input, **int** target)
11. {
12. sort(input.begin(), input.end());
14. **int** res = 0;
15. **for** (**int** i = 0; i < input.size(); i++)
16. {
17. **if** (input[i] == target) **continue**;
18. res++;
19. }
20. **return** res;
21. }
22. };
24. **int** main()
25. {
26. Solution a;
27. vector<**int**> input{ 0,1,2,2,3,0,4,2 };
28. **int** target = 2;
29. **int** res = a.removeElement(input, target);
30. cout << res << endl;
31. cin.get();
32. **return** 0;
33. }

**031. Next permutation (M) [array, math]**

**Problem**: Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place and use only constant extra memory.

**Example:** 1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

**Analysis:**

* Solution: use c++ std::next\_permutation
* Solution1: given an array (325421); first, from the lowest bit, find the first element (2) that descending; then switch this element with the one that is smallest one that is larger than this element in the right hand side (345221); finally, order the right hand side element in ascending order (341225).

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **void** nextPermutation(vector<**int**>& input)
11. {
12. next\_permutation(input.begin(), input.end());
13. }
14. };
16. **class** Solution1
17. {
18. **public**:
19. **void** nextPermutation(vector<**int**>& input)
20. {
21. **int** keyIndex = input.size() - 1;
22. **while** ((keyIndex > 0) && (input[keyIndex] <= input[keyIndex - 1]))
23. {
24. keyIndex--;
25. }
27. **if** (keyIndex == 0)
28. {
29. sort(input.begin(), input.end());
30. }
31. **else**
32. {
33. **int** minNum = input[keyIndex - 1];
34. **for** (**int** i = input.size() - 1; i > keyIndex - 1; i--)
35. {
36. **if** (input[i] > minNum)
37. {
38. **int** temp;
39. temp = input[i];
40. input[i] = input[keyIndex - 1];
41. input[keyIndex - 1] = temp;
42. **break**;
43. }
44. }
45. }
46. sort(input.begin() + keyIndex, input.end());
47. }
48. };
50. **int** main()
51. {
52. vector<**int**> input{ 3,2,5,4,2,1 };
53. Solution1 a;
54. a.nextPermutation(input);
55. **for** (**int** i = 0; i < input.size(); i++)
56. cout << input[i] << ", ";
57. cin.get();
58. **return** 0;
59. }

**036. Valid Sudoku (M) [array]**

**Problem:** Determine if a 9x9 Sudoku board is valid. Only the filled cells need to be validated according to the following rules:

* Each row must contain the digits 1-9 without repetition.
* Each column must contain the digits 1-9 without repetition.
* Each of the 9 3x3 sub-boxes of the grid must contain the digits 1-9 without repetition.

**Example:** Input: [ ["5","3",".",".","7",".",".",".","."],

["6",".",".","1","9","5",".",".","."],

[".","9","8",".",".",".",".","6","."],

["8",".",".",".","6",".",".",".","3"],

["4",".",".","8",".","3",".",".","1"],

["7",".",".",".","2",".",".",".","6"],

[".","6",".",".",".",".","2","8","."],

[".",".",".","4","1","9",".",".","5"],

[".",".",".",".","8",".",".","7","9"] ]

Output: true

Note:

* A Sudoku board (partially filled) could be valid but is not necessarily solvable.
* Only the filled cells need to be validated according to the mentioned rules.
* The given board contain only digits 1-9 and the character '.'.
* The given board size is always 9x9.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **bool** isSudoku(vector<vector<**char**>>& s)
6. {
7. **int** m = s.size(), n = s[0].size();
8. vector<vector<**bool**>> rowFlag(m, vector<**bool**>(n, **false**));
9. vector<vector<**bool**>> colFlag(m, vector<**bool**>(n ,**false**));
10. vector<vector<**bool**>> subFlag(m, vector<**bool**>(n, **false**));
12. **for** (**int** i = 0; i < s.size(); i++)
13. {
14. **for** (**int** j = 0; j < s[i].size(); j++)
15. {
16. **if** ((s[i][j] >= '1') && (s[i][j] <= '9'))
17. {
18. **int** c = s[i][j] - '1';
19. **if** ((rowFlag[i][c]) || (colFlag[c][j]) || (subFlag[3 \* (i / 3) + j / 3][c])) **return** **false**;
20. rowFlag[i][c] = **true**;
21. colFlag[c][j] = **true**;
22. subFlag[3 \* (i / 3) + j / 3][c] = **true**;
23. }
24. }
25. }
26. **return** **true**;
27. }
29. **int** main()
30. {
31. vector<vector<**char**>> s{ {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
32. {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
33. {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
34. {'8', '.', '.', '.', '6', '.', '.', '.', '3'},
35. {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
36. {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
37. {'.', '6', '.', '.', '.', '.', '2', '8', '.'},
38. {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
39. {'.', '.', '.', '.', '8', '.', '.', '7', '9'} };
40. cout << isSudoku(s) << endl;
41. system("pause");
42. **return** 1;
43. }

**041. First missing positive (H)**

**Problem:** Given an unsorted integer array, find the smallest missing positive integer.

**Example:** Input: [1,2,0]

Output: 3

**Note:** Your algorithm should run in O(n) time and uses constant extra space.

**Analysis:**

1. sort the array in ascending order;
2. find the first positive element position i (if not, result is the last element potion);
3. start the loop from i to end of array;
4. if there is repeat element, continue;
5. if A[i] – res != 1, return res+1 (res initial value set as 0);
6. else res=A[i];
7. finish the loop and return res+1.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** firstMissing(vector<**int**> input)
11. {
12. sort(input.begin(), input.end());
14. **int** res = 0;
15. **int** i = 0;
16. **while** ((i < input.size()) && (input[i] <= 0)) i++;
18. **for** (; i < input.size(); i++)
19. {
20. **if** ((i > 0) && (input[i] == input[i - 1])) **continue**;
22. **if** (input[i] - res != 1) **return** res + 1;
23. **else** res = input[i];
24. }
25. **return** res + 1;
26. }
27. };
29. **int** main()
30. {
31. vector<**int**> input{ 7,8,9,11,12 };
32. Solution a;
33. **int** res = a.firstMissing(input);
34. cout << res << endl;
35. cin.get();
36. **return** 1;
37. }

**048. Rotate image (M)**

**Problem:** You are given an n x n 2D matrix representing an image. Rotate the image by 90 degrees (clockwise).

**Example:** Given input matrix =

[ [1,2,3],

[4,5,6],

[7,8,9]],

rotate the input matrix in-place such that it becomes:

[

[7,4,1],

[8,5,2],

[9,6,3]

]

**Note:** You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

**Analysis:**

* i loops from 0 – input.size()/2
* j loops from i – input[i].size-1-i

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> rotateImage(vector<vector<**int**>>& input)
10. {
11. **int** n = input.size() - 1;
13. **for** (**int** i = 0; i < input.size() / 2; i++)
14. {
15. **for** (**int** j = i; j < input[i].size() - 1 - i; j++)
16. {
17. **int** temp = input[i][j];
18. input[i][j] = input[n - j][i];
19. input[n - j][i] = input[n - i][n - j];
20. input[n - i][n - j] = input[j][n - i];
21. input[j][n - i] = temp;
22. }
23. }
24. }
25. };

**054. Spiral matrix (M) [array]**

**Problem:** Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

**Example:** Input: [ [1, 2, 3, 4],

[5, 6, 7, 8],

[9,10,11,12]]

Output: [1,2,3,4,8,12,11,10,9,5,6,7]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;

6. vector<**int**> spiralMatrix(vector<vector<**int**>>& mat)
7. {
8. **int** u = 0, d = mat.size() - 1, l = 0, r = mat[0].size() - 1;
9. vector<**int**> res;
10. **if** (mat.empty()) **return** res;
11. res.push\_back(mat[0][0]);
13. **int** i = 0, j = 0;
14. **char** direction = 'r';
15. **while** ((l <= r) && (u <= d))
16. {
17. **switch** (direction)
18. {
19. **case** 'r':
20. {
21. **while** (j < r)
22. {
23. res.push\_back(mat[i][j + 1]);
24. j++;
25. }
26. direction = 'd';
27. u++;
28. **break**;
29. }
31. **case** 'd':
32. {
33. **while** (i < d)
34. {
35. res.push\_back(mat[i + 1][j]);
36. i++;
37. }
38. direction = 'l';
39. r--;
40. **break**;
41. }
43. **case** 'l':
44. {
45. **while** (j > l)
46. {
47. res.push\_back(mat[i][j - 1]);
48. j--;
49. }
50. direction = 'u';
51. d--;
52. **break**;
53. }
55. **case** 'u':
56. {
57. **while** (i > u)
58. {
59. res.push\_back(mat[i - 1][j]);
60. i--;
61. }
62. direction = 'r';
63. l++;
64. **break**;
65. }
66. }
67. }
68. **return** res;
69. }
71. **int** main()
72. {
73. vector<vector<**int**>> mat{ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
74. vector<**int**> res = spiralMatrix(mat);
75. **for** (**int** i = 0; i < res.size(); i++)
76. {
77. cout << res[i] << " ";
78. }
79. system("pause");
80. **return** 1;
81. }

**059. Spiral matrix II (M) [array]**

**Problem:** Given a positive integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

**Example:** Input: 3

Output: [ [ 1, 2, 3 ],

[ 8, 9, 4 ],

[ 7, 6, 5 ]]

**Analysis:** similar to 54..

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. vector<vector<**int**>> spiralMat(**int** n)
6. {
7. **if** (n <= 0) **return** {};
8. vector<vector<**int**>> res(n, vector<**int**>(n, 0));
10. **int** l = 0, r = n - 1, u = 0, d = n - 1;
11. **int** data = 1;
12. res[0][0] = data;
13. **char** direction = 'r';
14. **int** i = 0, j = 0;
15. **while** ((l <= r) && (u <= d))
16. {
17. **switch** (direction)
18. {
19. **case** 'r':
20. {
21. **while** (j < r) res[i][++j] = ++data;
22. direction = 'd';
23. u++;
24. **break**;
25. }
26. **case** 'd':
27. {
28. **while** (i < d) res[++i][j] = ++data;
29. direction = 'l';
30. r--;
31. **break**;
32. }
33. **case** 'l':
34. {
35. **while** (j > l) res[i][--j] = ++data;
36. direction = 'u';
37. d--;
38. **break**;
39. }
40. **case** 'u':
41. {
42. **while** (i > u) res[--i][j] = ++data;
43. direction = 'r';
44. l++;
45. **break**;
46. }
47. }
48. }
49. **return** res;
50. }
52. **int** main()
53. {
54. vector<vector<**int**>> res = spiralMat(5);
55. **for** (**int** i = 0; i < res.size(); i++)
56. {
57. **for** (**int** j = 0; j < res[i].size(); j++)
58. {
59. cout << res[i][j] << " ";
60. }
61. cout << endl;
62. }
63. system("pause");
64. **return** 1;
65. }

**066. Plus one (E) [array]**

**Problem:** Given a non-empty array of digits representing a non-negative integer, plus one to the integer. The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

**Example:** Input: [4,3,2,1]

Output: [4,3,2,2]

**Analysis:** similar to 067

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<**int**> plusOne(vector<**int**>& input)
10. {
11. reverse(input.begin(), input.end());
12. **int** carry = 0;
13. input[0]++;
14. **for** (**int** i = 0; i < input.size(); i++)
15. {
16. input[i] += carry;
17. carry = input[i] / 10;
18. input[i] %= 10;
19. }
20. **if** (carry == 1)
21. {
22. input.push\_back(1);
23. }
24. reverse(input.begin(), input.end());
25. **return** input;
26. }
27. };
29. **int** main()
30. {
31. vector<**int**> input{ 9,9,9 };
32. Solution a;
33. vector<**int**> res = a.plusOne(input);
34. **for** (**int** i = 0; i < res.size(); i++)
35. cout << res[i] << ", ";
36. system("pause");
37. **return** 1;
38. }

**073. Set matrix zeros (M) [array]**

**Problem:** Given a m x n matrix, if an element is 0, set its entire row and column to 0. Do it in-place.

**Example:** Input: [ [1,1,1],

[1,0,1],

[1,1,1] ]

Output: [ [1,0,1],

[0,0,0],

[1,0,1] ]

**Follow up:**

* A straight forward solution using O(mn) space is probably a bad idea.
* A simple improvement uses O(m + n) space, but still not the best solution.
* Could you devise a constant space solution?

**Analysis:**

1. Scan first col and row, if 0 exists, corresponding flag true;
2. Scan entire matrix except first col and row, if find 0 element, corresponding first element in col and row set to 0;
3. Scan entire matrix except first col and row, if first element in col and row is 0, set entire col and row to 0;
4. Determine flag, if true, set corresponding first col or row to 0.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **void** setZeros(vector<vector<**int**>>& mat)
6. {
7. **if** ((mat.empty()) || (mat[0].empty())) **return**;
8. **int** m = mat.size() - 1, n = mat[0].size() - 1;
9. **bool** rowFlag = **false**, colFlag = **false**;
11. **for** (**int** i = 0; i <= m; i++)
12. {
13. **if** (mat[i][0] == 0) rowFlag = **true**;
14. }
15. **for** (**int** j = 0; j <= n; j++)
16. {
17. **if** (mat[0][j] == 0) colFlag = **true**;
18. }
20. **for** (**int** i = 1; i <= m; i++)
21. {
22. **for** (**int** j = 1; j <= n; j++)
23. {
24. **if** (mat[i][j] == 0)
25. {
26. mat[0][j] = 0;
27. mat[i][0] = 0;
28. }
29. }
30. }
32. **for** (**int** i = 1; i <= m; i++)
33. {
34. **for** (**int** j = 1; j <= n; j++)
35. {
36. **if** ((mat[0][j] == 0) || (mat[i][0] == 0))
37. {
38. mat[i][j] = 0;
39. }
40. }
41. }
43. **if** (rowFlag)
44. {
45. **for** (**int** i = 0; i <= m; i++)
46. {
47. mat[i][0] = 0;
48. }
49. }
50. **if** (colFlag)
51. {
52. **for** (**int** j = 0; j <= n; j++)
53. {
54. mat[0][j] = 0;
55. }
56. }
57. }
59. **int** main()
60. {
61. vector<vector<**int**>> mat{ {1, 1, 1}, {1, 0, 1}, {1, 1, 1} };
62. setZeros(mat);
63. **for** (**int** i = 0; i < mat.size(); i++)
64. {
65. **for** (**int** j = 0; j < mat[i].size(); j++)
66. {
67. cout << mat[i][j] << " ";
68. }
69. cout << endl;
70. }
71. system("pause");
72. **return** 1;
73. }

**080. Remove Duplicates from Sorted Array II (M) [array, two pointers]**

**Problem:** Given a sorted array nums, remove the duplicates in-place such that duplicates appeared at most twice and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

**Example:** Given nums = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of nums being modified to 0, 0, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.

**Analysis:** similar to 026…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** removeDuplicates(vector<**int**>& nums)
6. {
7. **if** (nums.size() <= 2) **return** nums.size();
8. **int** flag = 1, index = 0;
9. **for** (**int** i = 1; i < nums.size(); i++)
10. {
11. **if** ((nums[index] == nums[i]) && (flag == 1))
12. {
13. nums[++index] = nums[i];
14. flag = 2;
15. }
16. **else** **if** (nums[index] != nums[i])
17. {
18. nums[++index] = nums[i];
19. flag = 1;
20. }
21. }
22. **return** index + 1;
23. }
25. **int** main()
26. {
27. vector<**int**> nums{ 0,0,1,1,1,1,2,3,3 };
28. **int** length = removeDuplicates(nums);
29. cout << length << endl;
30. **for** (**int** i = 0; i < length; i++)
31. {
32. cout << nums[i] << " ";
33. }
34. system("pause");
35. **return** 1;
36. }

**118. Pascal’s triangle (E) [array]**

**Problem:** Given a non-negative integer numRows, generate the first numRows of Pascal's triangle.

**Example:** Input: 5

Output: [ [1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1] ]

**Analysis:** easy

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> pascalTriangle(**int** n)
10. {
11. vector<vector<**int**>> res;
12. vector<**int**> prev;
14. **for** (**int** i = 1; i <= n; i++)
15. {
16. vector<**int**> curr;
17. curr.push\_back(1);
18. **for** (**int** j = 1; j < i - 1; j++)
19. {
20. **int** prev2 = prev.at(j - 1);
21. **int** prev1 = prev.at(j);
22. curr.push\_back(prev2 + prev1);
23. }
24. **if** (i > 1)
25. {
26. curr.push\_back(1);
27. }
28. prev = curr;
29. res.push\_back(curr);
30. }
31. **return** res;
32. }
33. };
35. **int** main()
36. {
37. Solution a;
38. vector<vector<**int**>> res = a.pascalTriangle(30);
39. **for** (**int** i = 0; i < res.size(); i++)
40. {
41. **for** (**int** j = 0; j < res[i].size(); j++)
42. {
43. cout << res[i][j] << " ";
44. }
45. cout << endl;
46. }
47. system("pause");
48. **return** 1;
49. }

**119. Pascal’s triangle II (E) [array]**

**Problem:** Given a non-negative index k where k ≤ 33, return the kth index row of the Pascal's triangle.

Note that the row index starts from 0.

**Example:** Input: 3

Output: [1,3,3,1]

**Follow up:** Could you optimize your algorithm to use only O(k) extra space?

**Analysis:** easy..

**Solution:** similar to 118

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<**int**> addRow(**int** n)
10. {
11. vector<**int**> prev;
12. **for** (**int** i = 0; i <= n; i++)
13. {
14. vector<**int**> curr;
15. curr.push\_back(1);
16. **for** (**int** j = 1; j <= i - 1; j++)
17. {
18. **int** prev2 = prev.at(j - 1);
19. **int** prev1 = prev.at(j);
20. curr.push\_back(prev2 + prev1);
21. }
22. **if** (i >= 1)
23. {
24. curr.push\_back(1);
25. }
26. prev = curr;
27. }
28. **return** prev;
29. }
30. };
32. **int** main()
33. {
34. Solution a;
35. vector<**int**> res = a.addRow(3);
36. **for** (**int** i = 0; i < res.size(); i++)
37. cout << res[i] << " ";
38. system("pause");
39. **return** 1;
40. }

**121. Best time to buy and sell stock (E) [array]**

**Problem:** Say you have an array for which the ith element is the price of a given stock on day i. If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

**Example:** Input: [7,1,5,3,6,4]

Output: 5

**Analysis:** easy…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** buySellStock(vector<**int**>stock)
11. {
12. **int** profit = 0;
13. **for** (**int** i = 0; i < stock.size() - 1; i++)
14. {
15. **for** (**int** j = i + 1; j < stock.size(); j++)
16. {
17. **if** (stock[j] > stock[i])
18. {
19. profit = max(stock[j] - stock[i], profit);
20. }
21. }
22. }
23. **return** profit;
24. }
25. };
27. **class** Solution1
28. {
29. **public**:
30. **int** buySellStock(vector<**int**>stock)
31. {
32. **int** profit = 0;
33. **int** minPrice = stock[0];
34. **for** (**int** i = 1; i < stock.size(); i++)
35. {
36. profit = max(stock[i] - minPrice, profit);
37. minPrice = min(minPrice, stock[i]);
38. }
39. **return** profit;
40. }
41. };
43. **int** main()
44. {
45. Solution1 a;
46. cout << a.buySellStock({ 7, 1, 5, 3, 6, 4 }) << endl; // 7,6,4,3,1
47. system("pause");
48. **return** 1;
49. }

**128. Longest consecutive sequence (H) [array, hash map]**

**Problem:** Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Your algorithm should run in O(n) complexity.

**Example:** Input: [100, 4, 200, 1, 3, 2]

Output: 4

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <unordered\_map>
4. #include <algorithm>
5. **using** **namespace** std;
7. **int** longestConsecutiveSeuqnce(vector<**int**>& nums)
8. {
9. unordered\_map<**int**, **bool**> count;
10. **for** (auto num : nums) count[num] = **true**;
11. **int** res = 0;
12. **for** (auto num : nums)
13. {
14. **int** len = 1;
15. **int** tmp = num + 1;
16. **while** (count.count(tmp))
17. {
18. count.erase(tmp);
19. tmp++;
20. len++;
21. }
22. tmp = num - 1;
23. **while** (count.count(tmp))
24. {
25. count.erase(tmp);
26. tmp--;
27. len++;
28. }
29. res = max(res, len);
30. }
31. **return** res;
32. }
34. **int** main()
35. {
36. vector<**int**> nums{ 100, 4, 200, 1, 3, 2 };
37. cout << longestConsecutiveSeuqnce(nums) << endl;
38. system("pause");
39. **return** 1;
40. }

**169. Majority element (E) [array, hash map]**

**Problem:** Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

**Example:** Input: [2,2,1,1,1,2,2]

Output: 2

**Analysis:**

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. #include <unordered\_map>
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. **int** majorityElement(vector<**int**>& input)
12. {
13. unordered\_map<**int**, **int**> count;
14. **for** (**int** i = 0; i < input.size(); i++)
15. {
16. count[input[i]]++;
17. }
19. **for** (auto it : count)
20. {
21. **if** (it.second > input.size() / 2)
22. **return** it.first;
23. }
24. }
25. };
27. **class** Solution1
28. {
29. **public**:
30. **int** majorityElement(vector<**int**>& input)
31. {
32. sort(input.begin(), input.end());
33. **return** input.at(input.size() / 2);
35. }
36. };
38. **int** main()
39. {
40. Solution1 a;
41. vector<**int**> input{ 2,2,1,1,1,2,2 };
42. cout << a.majorityElement(input) << endl;
43. system("pause");
44. **return** 1;
45. }

**189. Rotate array (E) [array]**

**Problem:** Given an array, rotate the array to the right by k steps, where k is non-negative.

**Example:** Input: [1,2,3,4,5,6,7] and k = 3

Output: [5,6,7,1,2,3,4]

**Note:**

* Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
* Could you do it in-place with O(1) extra space?

**Analysis:** easy…

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<**int**> rotateArray(vector<**int**>& input, **int** k)
10. {
11. **int** temp;
12. **for** (**int** i = 1; i <= k; i++)
13. {
14. temp = input[input.size() - 1];
15. **for** (**int** j = input.size()-1; j >0; j--)
16. {
17. input[j] = input[j - 1];
18. }
19. input[0] = temp;
20. }
21. **return** input;
22. }
23. };
25. **class** Solution1
26. {
27. **public**:
28. vector<**int**> rotateArray(vector<**int**>& input, **int** k)
29. {
30. **for** (**int** i = 0; i < input.size() - k; i++)
31. {
32. input.push\_back(input[0]);
33. input.erase(input.begin());
34. }
35. **return** input;
36. }
37. };
39. **int** main()
40. {
41. Solution1 a;
42. vector<**int**> input = { 1,2,3,4,5,6,7 };
43. vector<**int**> res = a.rotateArray(input, 3);
44. **for** (**int** i = 0; i < res.size(); i++)
45. cout << res[i] << " ";
46. system("Pause");
47. **return** 1;
48. }

**442. Find all duplicates in an array (M) [array]**

**Problem:** Given an array of integers, 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.

Find all the elements that appear twice in this array.

Could you do it without extra space and in O(n) runtime?

**Example:** Input: [4,3,2,7,8,2,3,1]

Output: [2,3]

**Soluion:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. vector<**int**> findDuplicates(vector<**int**>& nums)
7. {
8. vector<**int**> res{};
9. **if** (nums.size() < 2) **return** res;
10. **if** ((nums.size() == 2) && (nums[0] == nums[1])) **return** { nums[0] };
11. sort(nums.begin(), nums.end());
12. **for** (**int** i = 0; i < nums.size() - 1; i++)
13. {
14. **if** (nums[i + 1] == nums[i])
15. {
16. res.push\_back(nums[i]);
17. }
18. }
19. **return** res;
20. }
22. **int** main()
23. {
24. vector<**int**> nums{ 4, 3, 2, 7, 8, 2, 3, 1 };
25. vector<**int**> res = findDuplicates(nums);
26. **for** (**int** i = 0; i < res.size(); i++)
27. {
28. cout << res[i] << " ";
29. }
30. system("pause");
31. **return** 1;
32. }

**448. Find all numbers disappeared in an array (E) [array]**

**Problem:** Given an array of integers where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.

Find all the elements of [1, n] inclusive that do not appear in this array.

Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.

**Example:** Input: [4,3,2,7,8,2,3,1]

Output: [5,6]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. vector<**int**> findDisappeared(vector<**int**>& nums)
7. {
8. vector<**int**> res;
9. **for** (**int** i = 0; i < nums.size(); i++)
10. {
11. **int** tmp = abs(nums[i]) - 1;
12. nums[tmp] = nums[tmp] > 0 ? -nums[tmp] : nums[tmp];
13. }
14. **for** (**int** i = 0; i < nums.size(); i++)
15. {
16. **if** (nums[i] > 0) res.push\_back(i + 1);
17. }
18. **return** res;
19. }
21. **int** main()
22. {
23. vector<**int**> nums{ 4, 3, 2, 7, 8, 2, 3, 1 };
24. vector<**int**> res = findDisappeared(nums);
25. **for** (**int** i = 0; i < res.size(); i++)
26. {
27. cout << res[i] << " ";
28. }
29. system("pause");
30. **return** 1;
31. }

**989. Add to arrary form of integer (E) [array]**

**Problem:** For a non-negative integer X, the array-form of X is an array of its digits in left to right order. For example, if X = 1231, then the array form is [1,2,3,1].

Given the array-form A of a non-negative integer X, return the array-form of the integer X+K.

**Example:** Input: A = [2,1,5], K = 806

Output: [1,0,2,1]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. vector<**int**> addToArrayForm(vector<**int**>& A, **int** K)
7. {
8. **int** carry = 0;
9. **if** (A.empty())
10. {
11. A.push\_back(K);
12. reverse(A.begin(), A.end());
13. }
14. **else**
15. {
16. reverse(A.begin(), A.end());
17. A[0] += K;
18. }
19. **for** (**int** i = 0; i < A.size(); i++)
20. {
21. A[i] += carry;
22. carry = A[i] / 10;
23. A[i] %= 10;
24. }
25. **while** (carry)
26. {
27. A.push\_back(carry % 10);
28. carry /= 10;
29. }
30. reverse(A.begin(), A.end());
31. **return** A;
32. }
34. **int** main()
35. {
36. vector<**int**> A = {9};
37. **int** K = 10000;
38. addToArrayForm(A, K);
39. **for** (**int** i = 0; i < A.size(); i++)
40. {
41. cout << A[i] << " ";
42. }
43. system("pause");
44. **return** 1;
45. }

**Section 03. String**

**005. Longest palindromic substring (M)**

**Problem:** Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

**Example:** Input: "babad"

Output: "bab"

Note: "aba" is also a valid answer.

Analysis: Manacher’s anglrithms

1. inset “#” to each side of each element (bob -> #b#o#b#, noon -> #n#o#o#n#) so that it will surely have odd number of elements
2. insert “$” to beginning of the string (#b#o#b# -> $#b#o#b#)
3. the longest palindromic substring’s length is radius -1 ($#b#o#b# -> length = 4-1 =3 -> bob)
4. the longest palindromic substring’s position is (middle position – radius)/2 ($#b#o#b# -> position = (4-4)/2 =0 -> bob)
5. assume there is a palindromic substring x that can reach the most right side; define mx as the right side position of x; define id as the middle point of x; define p[i] is the radius of the palindromic substring which t[i] character as center
6. p[i] = mx > i ? min(p[2 \* id - i], mx - i) : 1;
7. if mx > i, then p[i] = min(p[2 \* id - i], mx - i);

* mx - i > P[j], palindromic substring centered at S[j] must be included by S[id], since i and j are symmetric, palindromic substring centered at S[i] also must be include by S[id] and p[i]=p[j];



* mx - i <= P[j], palindromic substring centered at S[j] is not necessarily included by S[id], since i and j are symmetric, the green portion in the following would be same, this means palindromic substring centered at S[i] will expand at least to mx position, then p[i]-mx-i. However, for element after mx, need to find out if it is palindromic…



1. mx <= i, have to assume P[i]=1 and then find out palindromic situation…

**Solution:**

1. #include <vector>
2. #include <string>
3. #include <iostream>
4. #include <algorithm>
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. string longestPalindromicSubstring(string s)
12. {
13. string t = "$#";
14. **for** (**int** i = 0; i < s.size(); i++)
15. {
16. t += s[i];
17. t += "#";
18. }
20. vector<**int**> p(t.size(), 0);
21. **int** mx = 0, id = 0, resLen = 0, resCenter = 0;
22. **for** (**int** i = 1; i < t.size(); i++)
23. {
24. p[i] = mx > i ? min(p[2 \* id - i], mx - i) : 1;
26. **while** (t[i + p[i]] == t[i - p[i]]) ++p[i];
27. **if** (mx < i + p[i])
28. {
29. mx = i + p[i];
30. id = i;
31. }
32. **if** (resLen < p[i])
33. {
34. resLen = p[i];
35. resCenter = i;
36. }
37. }
39. **return** s.substr((resCenter - resLen) / 2, resLen - 1);
40. }
41. };
43. **int** main()
44. {
45. string s = "12212";
46. Solution a;
47. cout << a.longestPalindromicSubstring(s) << endl;
48. cin.get();
49. **return** 0;
50. }

**006. ZigZag conversion (E) [string, math]**

**Problem:** The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

P A H N

A P L S I I G

Y I R

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

string convert(string s, int numRows);

**Example:** s = "PAYPALISHIRING", numRows = 4

Output: "PINALSIGYAHRPI"

Explanation:

P I N

A L S I G

Y A H R

P I

**Analysis:**

1. every row repeat in a pattern of 2\*nRow-2;
2. other than the first and last row, the rows in middle will repeat just one more time, this distant of this element to the first position of row is 2\*nRow -2 – 2\*i;
3. first loop i is from 0 – nRow, with step 1; second loop j is from i – input.size();
4. push back i+2\*nRow-2
5. if it is one of the middle rows, push back another j + 2\*nRow-2-2\*i

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string zigzagConvert(string s, **int** numRows)
10. {
11. string res;
12. **if** (numRows < 2) **return** s;
13. **for** (**int** i = 0; i < numRows; i++)
14. {
15. **for** (**int** j = i; j < s.size(); j += 2 \* (numRows - 1))
16. {
17. res.push\_back(s[j]);
19. **if** ((i > 0) && (i < numRows - 1))
20. {
21. **if** (j + 2 \* (numRows - i - 1) < s.length())
22. {
23. res.push\_back(s[j + 2 \* (numRows - i - 1)]);
24. }
25. }
26. }
27. }
28. **return** res;
29. }
30. };
32. **int** main()
33. {
34. Solution a;
35. cout << a.zigzagConvert("PAYPALISHIRING", 3) << endl;
36. cin.get();
37. **return** 0;
38. }

**008. String to integer (E) [string]**

**Problem:** Implement atoi (ASCII to integer) which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

**Example:** Input: "42" Output: 42

Input: " -42" Output: -42

Input: "words and 987" Output: 0

Input: "4193 with words" Output: 4193

Input: "-91283472332" Output: -2147483648

**Note:**

* Only the space character ' ' is considered as whitespace character.
* Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. If the numerical value is out of the range of representable values, INT\_MAX (231 − 1) or INT\_MIN (−231) is returned.

**Analysis:**

1. if string s is empty, return 0;
2. define ans as 0, sign as 1, i as 0;
3. remove whilespace ‘ ’ or ‘\t’;
4. if i==s.length(), return 0;
5. determine sign
6. loop from i to s.size()
7. if ans > (numeric\_limits<int>::max()-(s[i]-‘0’))/10 return max or min
8. ans \*= 10, ans+=s[i] – ‘0’
9. determine sign and return ans

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** string2integer(string s)
10. {
11. **if** (s.empty()) **return** 0;
13. **int** ans = 0;
14. **int** sign = 1;
15. **int** i = 0;
17. **while** ((s[i] == ' ') || (s[i] == '\t')) i++;
19. **if** (i == s.length()) **return** 0;
21. **if** (s[i] == '+') i++;
22. **else** **if** (s[i] == '-')
23. {
24. sign = -1;
25. i++;
26. }
28. **for** (; (i < s.length()) && (isdigit(s[i])); i++)
29. {
30. **if** ((ans) > (numeric\_limits<**int**>::max()- (s[i] - '0'))/10)
31. {
32. **return** sign > 0 ? numeric\_limits<**int**>::max() : numeric\_limits<**int**>::min();
33. }
34. ans \*= 10;
35. ans += s[i] - '0';
36. }
37. ans \*= sign;
38. **return** ans;
39. }
40. };
42. **int** main()
43. {
44. Solution a;
45. cout << numeric\_limits<**int**>::max() << " " << numeric\_limits<**int**>::min() << endl;
46. cout << a.string2integer("-91283472332") << endl;
47. cin.get();
48. **return** 0;
49. }

**014. Longest common prefix (E) [string]**

**Problem:** Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

**Example:** Input: ["flower","flow","flight"]

Output: "fl"

**Note:** All given inputs are in lowercase letters a-z.

**Analysis:** seems easy…but…

1. If input is empty, return “”;
2. first loop i is from 0 to input[0].size();
3. second loop j is from 0 to input.size();
4. if i>=input[j].size || input[j][i]!=input[0][i], return input[0].substr(0, i)
5. finish loops; and if nothing is return, return input[0]

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <vector>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. string longestCommonPrefix(vector<string> input)
11. {
12. **if** (input.empty()) **return** "";
14. **for** (**int** i = 0; i < input[0].size(); i++)
15. {
16. **for** (**int** j = 0; j < input.size(); j++)
17. {
18. **if** ((i >= input[j].size()) || (input[j][i] != input[0][i]))
19. {
20. **return** input[0].substr(0, i);
21. }
22. }
23. }
24. //return input[0];
25. }
26. };
28. **int** main()
29. {
30. Solution a;
31. vector<string> input{ "flower", "flowerere", "flowererereight" };
32. cout << a.longestCommonPrefix(input) << endl;
33. cin.get();
34. **return** 0;
35. }

**028. Implement strStr() (E) [string]**

**Problem:** Return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

**Example:** Input: haystack = "hello", needle = "ll"

Output: 2

**Note:** What should we return when needle is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when needle is an empty string. This is consistent to C's strstr() and Java's indexOf().

Analysis: easy…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** strStr(string haystack, string needle)
10. {
11. **int** m = haystack.size(), n = needle.size();
13. **if** (n == 0) **return** 0;
15. **for** (**int** i = 0; i < m; i++)
16. {
17. **if** (i + n > m) **break**;
18. **for** (**int** j = 0; j < n; j++)
19. {
20. **if** (haystack[i + j] != needle[j]) **break**;
22. **else** **if** ((j = n - 1) && (haystack[i + j] == needle[j])) **return** i;
23. }
24. }
25. **return** -1;
26. }
27. };
29. **int** main()
30. {
31. Solution a;
32. cout << a.strStr("aaaaa", "bba") << endl;
33. cin.get();
34. **return** 0;
35. }

**038. Count and say (E) [string]**

**Problem:** The count-and-say sequence is the sequence of integers with the first five terms as following:

1. 1

2. 11

3. 21

4. 1211

**Example:** Input: 4

Output: "1211"

**Analysis:** easy…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string countAndSay(**int** n)
10. {
11. **if** (n < 1) **return** "";
13. string str = "1";
14. **for** (**int** i = 1; i < n; i++)
15. {
16. string tmp = "";
17. **int** count = 0;
18. **char** current = str[0];
19. **int** index = 0;
20. **while** (index < str.size())
21. {
22. **if** (str[index] == current)
23. {
24. count++;
25. }
26. **else**
27. {
28. tmp += to\_string(count) + current;
29. current = str[index];
30. count = 1;
31. }
32. index++;
33. }
34. tmp += to\_string(count) + current;
35. str = tmp;
36. }
37. **return** str;
38. }
39. };
41. **int** main()
42. {
43. Solution a;
44. **for** (**int** i = 1; i < 13; i++)
45. {
46. cout << "i = " << i << ",\t" << "seq = " << a.countAndSay(i) << endl;
47. }
48. cin.get();
49. **return** 0;
51. }

**043. Multiply strings (M)**

**Problem:** Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.

**Example:** Input: num1 = "2", num2 = "3"

Output: "6"

**Note:**

* The length of both num1 and num2 is < 110.
* Both num1 and num2 contain only digits 0-9.
* **Both num1 and num2 do not contain any leading zero, except the number 0 itself**.
* You must not use any built-in BigInteger library or convert the inputs to integer directly

**Analysis:**

1. determine if num1 or num2 are null, if so, return “”;
2. reverse both num1 and num2 so that lower bit starting from the right;
3. fisrt loop j starts from 0 to num2.size()
4. initialize int carry = 0 and int val = num2[j]-‘0’
5. second loop I starts from 0 to num1.size();
6. calculate carry, res[i+j] and update carry
7. complete second loop and deal with the situation thet if highest carry !=0
8. complete first loop and reverse res
9. remove left hand redundant ‘0’

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string multiplyStrings(string num1, string num2)
10. {
11. **if** (num1.empty() || num2.empty()) **return** "";
13. reverse(num1.begin(), num1.end());
14. reverse(num2.begin(), num2.end());
16. string res(num1.size() + num2.size(), '0');
18. **for** (**int** j = 0; j < num2.size(); j++)
19. {
20. **int** val = num2[j] - '0';
21. **int** carry = 0;
22. **for** (**int** i = 0; i < num1.size(); i++)
23. {
24. carry += (num1[i] - '0') \* val + (res[i + j] - '0');
25. res[i + j] = carry % 10 + '0';
26. carry /= 10;
27. }
28. **if** (carry != 0) res[num1.size() + j] = carry + '0';
29. }
31. reverse(res.begin(), res.end());
33. **int** count = 0;
34. **while** ((count < res.size() - 1) && (res[count] == '0')) count++;
35. res.erase(0, count);
36. **return** res;
37. }
38. };
40. **int** main()
41. {
42. string num1 = "123", num2 = "456";
43. Solution a;
44. cout << num1 << " x " << num2 << " = " << a.multiplyStrings(num2, num1) << endl;
45. cin.get();
46. **return** 0;
47. }

**058. Length of last word (E) [string]**

**Problem:** Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string. If the last word does not exist, return 0.

**Example:** Input: "Hello World"

Output: 5

**Note:** A word is defined as a character sequence consists of non-space characters only.

**Analysis:** east…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** lengthOfLastWord(string s)
10. {
11. **int** i = s.size() - 1;
12. **while** ((i >= 0) && (s[i] == ' ')) --i;
14. **int** length = 0;
15. **while** ((i >= 0) && (s[i] != ' '))
16. {
17. length++;
18. i--;
19. }
20. **return** length;
21. }
22. };
24. **int** main()
25. {
26. Solution a;
27. cout << a.lengthOfLastWord("Hello world    ") << endl;
28. cin.get();
29. **return** 0;
30. }

**067. Add binary (E) [string]**

**Problem:** Given two binary strings, return their sum (also a binary string).

The input strings are both non-empty and contains only characters 1 or 0.

**Example:** Input: a = "11", b = "1"

Output: "100"

**Analysis:** easy…

**Solution:**

1. #include <string>
2. #include <algorithm>
3. #include <iostream>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. string addBinary(string bin1, string bin2)
11. {
12. reverse(bin1.begin(), bin1.end());
13. reverse(bin2.begin(), bin2.end());
15. string sum;
16. **int** n = max(bin1.size(), bin2.size());
17. **int** carry = 0;
18. **for** (**int** i = 0; i < n; i++)
19. {
20. **if** (i < bin1.size()) carry += bin1[i] - '0';
21. **if** (i < bin2.size()) carry += bin2[i] - '0';
22. sum.push\_back(carry % 2 + '0');
23. carry /= 2;
24. }
25. **if** (carry) sum.push\_back('1');
26. reverse(sum.begin(), sum.end());
27. **return** sum;
28. }
29. };
31. **int** main()
32. {
33. Solution a;
34. string bin1 = "1010", bin2 = "1011";
35. cout << a.addBinary(bin1, bin2) << endl;
36. cin.get();
37. **return** 0;
38. }

**068. Text justification (H)**

**Problem:** Given an array of words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

**Example:** Input: words = ["This", "is", "an", "example", "of", "text", "justification."] maxWidth = 16

Output: [ "This is an",

"example of text",

"justification. "]

**Analysis:** I gave up… from now on, I will change strategy which starting from easy to medium to hard ones…

**125. Valid palindrome (E) [string, two pointers]**

**Problem:** Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

**Example:** Input: "A man, a plan, a canal: Panama"

Output: true

**Analysis:** isalnum()

tolower()

**Solution:**

1. #include <iostream>
2. #include <string>
3. //#include <cctype>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **bool** isPalindrome(string str)
11. {
12. **if** (str.empty()) **return** **true**;
14. **int** left = 0, right = str.size() - 1;
15. **while** (left < right)
16. {
17. **if** (!isalnum(str[left])) left++;
18. **else** **if** (!isalnum(str[right])) right--;
19. **else** **if** (tolower(str[left]) != tolower(str[right])) **return** **false**;
20. **else** {
21. left++;
22. right--;
23. }
24. }
25. **return** **true**;
26. }
27. };
29. **int** main()
30. {
31. Solution a;
32. cout << a.isPalindrome("A man, a plan, a canal: Panama") << endl;
33. system("pause");
34. **return** 1;
35. }

**165. Compare version numbers (E) [string]**

**Problem:** Compare two version numbers version1 and version2. If version1 > version2 return 1; if version1 < version2 return -1; otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . characters.

The . character does not represent a decimal point and is used to separate number sequences.

For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

You may assume the default revision number for each level of a version number to be 0. For example, version number 3.4 has a revision number of 3 and 4 for its first and second level revision number. Its third and fourth level revision number are both 0.

**Example:** Input: version1 = "7.5.2.4", version2 = "7.5.3"

Output: -1

Input: version1 = "1.01", version2 = "1.001"

Output: 0

**Analysis:** easy…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** compareVersion(string str1, string str2)
10. {
11. **int** m = str1.size(), n = str2.size();
13. **for** (**int** i = 0, j = 0; i < m || j < n; i++, j++)
14. {
15. **int** v1 = 0, v2 = 0;
16. **while** ((i < m) && (str1[i] != '.'))
17. {
18. v1 = v1 \* 10 + (str1[i] - '0');
19. i++;
20. }
21. **while** ((j < n) && (str2[j] != '.'))
22. {
23. v2 = v2 \* 10 + (str2[j] - '0');
24. j++;
25. }
26. **if** (v1 != v2)
27. **return** v1 > v2 ? 1 : -1;
28. }
29. **return** 0;
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. cout << a.compareVersion("1.01", "1.001") << endl;
37. system("pause");
38. **return** 1;
39. }

**242. Valid anagram (E) [string, hash map]**

**Problem:** Given two strings s and t, write a function to determine if t is an anagram of s.

**Note:** You may assume the string contains only lowercase alphabets.

**Follow up:** What if the inputs contain unicode characters? How would you adapt your solution to such case?

L“” ⬄ wstring

**Example:** Input: s = "anagram", t = "nagaram"

Output: true

**Solution:**

1. #include <string>
2. #include <unordered\_map>
3. #include <iostream>
4. #include <algorithm> // sort()
6. **using** **namespace** std;
8. **class** Solution
9. {
10. **public**:
11. **bool** validAnagram(wstring str1, wstring str2)
12. {
13. sort(str1.begin(), str1.end());
14. sort(str2.begin(), str2.end());
15. **return** str1 == str2;
16. }
17. };
19. **class** Solution2
20. {
21. **public**:
22. **bool** validAnagram(string str1, string str2)
23. {
24. **if** (str1.size() != str2.size()) **return** **false**;
26. unordered\_map<**char**, **int**> count;
27. **for** (**int** i = 0; i < str1.size(); i++)
28. {
29. count[str1[i]]++;
30. }
32. **for** (**int** j = 0; j < str2.size(); j++)
33. {
34. count[str2[j]]--;
35. **if** (count[str2[j]] < 0) **return** **false**;
36. }
37. **return** **true**;
38. }
39. };
41. **int** main()
42. {
43. Solution a;
44. //cout << a.validAnagram("anagrama", "nagaram") << endl;
45. cout << a.validAnagram(L"你好", L"好你") << endl;
46. system("pause");
47. **return** 1;
48. }

**383. Ransom note (E) [string, hash map]**

**Problem:** Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

**Example:** canConstruct("aa", "aab") -> true

canConstruct("aa", "ab") -> false

**Analysis:** similar to 242, using hash map

**Solution:**

1. #include <string>
2. #include <unordered\_map>
3. #include <iostream>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **bool** canConstruct(string str1, string str2)
11. {
12. **int** m = str1.size(), n = str2.size();
13. unordered\_map<**char**, **int**> count;
14. **for** (**int** j = 0; j < n; j++)
15. {
16. ++count[str2[j]];
17. }
18. **for** (**int** i = 0; i < m; i++)
19. {
20. --count[str1[i]];
21. **if** (count[str1[i]] < 0) **return** **false**;
22. }
23. **return** **true**;
24. }
25. };
27. **int** main()
28. {
29. Solution a;
30. cout << a.canConstruct("aa", "cbtdyayca") << endl;
31. system("pause");
32. **return** 1;
34. }

**405. Convert a number to hexadecimal (E) [string, bit]**

**Problem:** Given an integer, write an algorithm to convert it to hexadecimal. For negative integer, two’s complement method is used.

**Example:** Input:26

Output: "1a"

Input: -1

Output: "ffffffff"

**Note:**

* All letters in hexadecimal (a-f) must be in lowercase.
* The hexadecimal string must not contain extra leading 0s. If the number is zero, it is represented by a single zero character '0'; otherwise, the first character in the hexadecimal string will not be the zero character.
* The given number is guaranteed to fit within the range of a **32-bit signed integer**.
* You must not use any method provided by the library which converts/formats the number to hex directly.

**Analysis:**

* No need to distinguish between + and –
* 32 bit range

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string int2hex(**int** num)
10. {
11. string res = "";
12. **if** (!num) **return** "0";
14. **while** (num && res.size() != **sizeof**(**int**) \* 2)
15. {
16. **int** hex = num & 15;
17. **if** (hex > 10)
18. {
19. res.push\_back(hex - 10 + 'a');
20. }
21. **else**
22. {
23. res.push\_back(hex + '0');
24. }
25. num >>= 4;
26. cout << num << endl;
27. }
28. reverse(res.begin(), res.end());
29. **return** res;
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. cout << a.int2hex(111) << endl;
37. system("pause");
38. **return** 1;
39. }

**408. Valid word abbreviation (E) [string]**

**Problem:** Given a non-empty string s and an abbreviation abbr, return whether the string matches with the given abbreviation.

A string such as "word" contains only the following valid abbreviations:

["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

Notice that only the above abbreviations are valid abbreviations of the string "word". Any other string is not a valid abbreviation of "word".

**Example:** Given s = "internationalization", abbr = "i12iz4n":

Return true.

**Note:** Assume s contains only lowercase letters and abbr contains only lowercase letters and digits.

**Analysis:** don’t forget two situations!

* abbr ended with numbers
* ‘0’ in abbr

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **bool** validAbbr(string word, string abbr)
10. {
11. **int** j = 0;
12. **int** temp = 0;
13. **for** (**int** i = 0; i < abbr.size(); i++)
14. {
15. **if** ((abbr[i] >= '0') && (abbr[i] <= '9'))
16. {
17. **if** ((abbr[i] == '0') && (temp == 0)) **return** **false**;
18. temp = temp \* 10 + (abbr[i] - '0');
19. }
20. **else** **if** (abbr[i] >= 'a'&&abbr[i] <= 'z')
21. {
22. j += temp;
23. **if** (abbr[i] != word[j]) **return** **false**;
24. **else**
25. {
26. j++;
27. temp = 0;
28. }
29. }
31. }
33. **if** (j + temp == word.size()) **return** **true**;
34. **else** **return** **false**;
35. }
36. };
38. **int** main()
39. {
40. Solution a;
41. cout << a.validAbbr("internationalizationaaa", "i12iz4n3") << endl;
42. system("pause");
43. **return** 1;
44. }

**415. Add strings (E) [string]**

**Problem:** Given two non-negative integers num1 and num2 represented as string, return the sum of num1 and num2.

**Note:**

* The length of both num1 and num2 is < 5100.
* Both num1 and num2 contains only digits 0-9.
* Both num1 and num2 does not contain any leading zero.
* You must not use any built-in BigInteger library or convert the inputs to integer directly.

**Analysis:** similar to 067…

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string addStrings(string str1, string str2)
10. {
11. reverse(str1.begin(), str1.end());
12. reverse(str2.begin(), str2.end());
14. **int** m = str1.size() >= str2.size() ? str1.size() : str2.size();
15. **int** carry = 0;
16. string res;
17. **for** (**int** i = 0; i < m; i++)
18. {
19. **if** (i < str1.size()) carry += (str1[i] - '0');
20. **if** (i < str2.size()) carry += (str2[i] - '0');
21. res.push\_back(carry % 10 + '0');
22. carry /= 10;
23. }
24. **if** (carry) res.push\_back(carry + '0');
25. reverse(res.begin(), res.end());
26. **return** res;
27. }
28. };
30. **int** main()
31. {
32. Solution a;
33. cout << a.addStrings("99", "99") << endl;
34. system("pause");
35. **return** 1;
36. }

**434. Number of segments in a string (E) [string]**

**Problem:** Count the number of segments in a string, where a segment is defined to be a contiguous sequence of non-space characters.

Please note that the string does not contain any non-printable characters.

**Example:** Input: "Hello, my name is John"

Output: 5

**Analysis:** consider several situations:

* white space ‘\n’ ‘\t’ ‘ ’
* null string
* white space at the end

**Solution:**

1. #include <string>
2. #include <iostream>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** countSegments(string str)
10. {
11. **int** count = 0;
12. **if** (str.empty()) **return** 0;
14. **for** (**int** i = 1; i < str.size(); i++)
15. {
16. **if** (((str[i] == ' ') || (str[i] == '\t')) &&
17. (str[i - 1] != ' ') && str[i - 1] != '\t') count++;
18. }
19. **if** ((str[str.size() - 1] == ' ')||(str[str.size() - 1] == '\t')) **return** count;
20. **else** **return** count - 1;
21. }
22. };
24. **int** main()
25. {
26. Solution a;
27. cout << a.countSegments("Hello, my name is John ") << endl;
28. system("pause");
29. **return** 1;
30. }

**443. String compression (E) [string, double pointers]**

**Problem:** Given an array of characters, compress it in-place.

The length after compression must always be smaller than or equal to the original array.

Every element of the array should be a character (not int) of length 1.

After you are done modifying the input array in-place, return the new length of the array.

**Example:** Input: ["a","a","b","b","c","c","c"]

Output: Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

Explanation: "aa" is replaced by "a2". "bb" is replaced by "b2". "ccc" is replaced by "c3".

**Note:**

* All characters have an ASCII value in [35, 126].
* 1 <= len(chars) <= 1000.

**Follow up:**

Could you solve it using only O(1) extra space?

**Analysis:**

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** strCompress(vector<**char**>& input)
11. {
12. **int** cur = 0;
13. **for** (**int** i = 0, j = 0; i < input.size(); i = j)
14. {
15. **while** ((j < input.size()) && (input[j] == input[i])) j++;
16. input[cur++] = input[i];
17. **if** (j - i == 1) **continue**;
18. **else**
19. {
20. **for** (**char** c : to\_string(j - i)) input[cur++] = c;
21. }
22. }
23. **return** cur;
24. }
25. };
27. **int** main()
28. {
29. Solution a;
30. vector<**char**> input{ 'a','b','b','b','b','b','b','b','b','b','b','a','a' };
31. cout << a.strCompress(input) << endl;
32. system("pause");
33. **return** 1;
34. }

**Section 04. Linked List**

**002. Add two numbers (M) [linked list]**

**Problem:** You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

**Example:** Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

**Analysis:** learn about linked list, check if a linked list is circular, then perform adding two numbers

**Solution:**

1. // c++ program to check if a linked list is circular
2. #include<iostream>
4. **using** **namespace** std;
6. **struct** Node
7. {
8. **int** data;
9. Node\* next;
10. };
12. Node\* newNode(**int** data)
13. {
14. Node\* temp = **new** Node;
15. temp->data = data;
16. temp->next = NULL;
17. **return** temp;
18. }
20. **class** Solution
21. {
22. **public**:
23. Node\* addTwoNumbers(Node\* node1, Node\* node2)
24. {
25. **int** carry = 0;
26. Node\* head = newNode(0);
27. Node\* res = head;
28. **while** ((node1 != NULL) || (node2 != NULL))
29. {
30. **if** (node1 != NULL)
31. {
32. carry += node1->data;
33. node1 = node1->next;
34. }
35. **if** (node2 != NULL)
36. {
37. carry += node2->data;
38. node2 = node2->next;
39. }
40. res->next = newNode(carry % 10);
41. res = res->next;
42. carry /= 10;
43. }
44. **if** (carry) res->next = newNode(carry);
45. **return** head->next;
46. }
48. **bool** isCircular(Node\* head)
49. {
50. **if** (head == NULL) **return** **true**;
52. Node\* node = head->next;
53. **while** ((node != NULL) && (node != head))
54. {
55. node = node->next;
56. }
57. **return** (node == head);
58. }
59. };
61. **int** main()
62. {
63. Node\* head = newNode(1);
64. head->next = newNode(2);
65. head->next = newNode(3);
66. //head->next = head;
67. cout << "This linked list ";
68. Solution a;
69. a.isCircular(head) ? cout << " is circular!" : cout << "is not curcular!" << endl;
71. Node \*node1 = newNode(2);
72. node1->next = newNode(4);
73. node1->next->next = newNode(3);
75. Node \*node2 = newNode(5);
76. node2->next = newNode(6);
77. node2->next->next = newNode(9);
79. Node \*res = a.addTwoNumbers(node1, node2);
80. **while** (res != NULL)
81. {
82. cout << res->data << " ";
83. res = res->next;
84. }
85. system("pause");
86. **return** 1;
87. }

**021. Merge two sorted lists (E) [linked list]**

**Problem:** Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

**Example:** Input: 1->2->4, 1->3->4

Output: 1->1->2->3->4->4

**Analysis:** key:

1. temp->next = node1 ? node1 : node2;

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **struct** Node
6. {
7. **int** data;
8. Node\* next;
9. };
11. Node\* newNode(**int** data)
12. {
13. Node\* temp = **new** Node;
14. temp->data = data;
15. temp->next = NULL;
16. **return** temp;
17. }
19. **class** Solution
20. {
21. **public**:
22. Node\* mergeLists(Node\* node1, Node\* node2)
23. {
24. Node\* head = newNode(0);
25. Node\* temp = head;
26. **while** ((node1 != NULL) && (node2 != NULL))
27. {
28. **if** (node1->data <= node2->data)
29. {
30. temp->next = node1;
31. node1 = node1->next;
32. }
33. **else**
34. {
35. temp->next = node2;
36. node2 = node2->next;
37. }
38. temp = temp->next;
39. }
40. temp->next = node1 ? node1 : node2;
41. **return** head->next;
42. }
43. };
45. **int** main()
46. {
47. Node\* node1 = newNode(1);
48. node1->next = newNode(2);
49. node1->next->next = newNode(4);
51. Node\* node2 = newNode(1);
52. node2->next = newNode(3);
53. node2->next->next = newNode(4);
55. Solution a;
56. Node\* res = a.mergeLists(node1, node2);
57. **while** (res != NULL)
58. {
59. cout << res->data << " ";
60. res = res->next;
61. }
62. system("pause");
63. **return** 1;
64. }

**024. Swap nodes in pairs (E) [linked list]**

**Problem:** Given a linked list, swap every two adjacent nodes and return its head. You may not modify the values in the list's nodes, only nodes itself may be changed.

**Example:** Given 1->2->3->4, you should return the list as 2->1->4->3

**Analysis:**

0

dummy (pre)

1

1

3

4

cur



**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **struct** Node
6. {
7. **int** data;
8. Node\* next;
9. };
11. Node\* newNode(**int** data)
12. {
13. Node\* temp = **new** Node;
14. temp->data = data;
15. temp->next = NULL;
16. **return** temp;
17. }
19. **class** Solution
20. {
21. **public**:
22. Node\* swapNodes(Node\* node)
23. {
24. Node\* head = newNode(0);
25. head->next = node;
26. Node\* pre = head;
27. Node\* cur = head->next;
29. **while** ((cur != NULL) && (cur->next != NULL))
30. {
31. pre->next = cur->next;
32. cur->next = cur->next->next;
33. pre->next->next = cur;
35. pre = cur;
36. cur = cur->next;
37. }
38. **return** head->next;
39. }
40. };
42. **int** main()
43. {
44. Node\* node = newNode(1);
45. node->next = newNode(2);
46. node->next->next = newNode(3);
47. node->next->next->next = newNode(4);
48. node->next->next->next->next = newNode(5);
50. Solution a;
51. Node\* res = a.swapNodes(node);
52. **while** (res != NULL)
53. {
54. cout << res->data << " ";
55. res = res->next;
56. }
57. system("pause");
58. **return** 1;
59. }

**083. Remove duplicate from sorted list (E) [linked list]**

**Problem:** Given a sorted linked list, delete all duplicates such that each element appear only once.

**Example:** Input: 1->1->2->3->3

Output: 1->2->3

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **struct** Node
6. {
7. **int** data;
8. Node\* next;
9. };
11. Node\* newNode(**int** data)
12. {
13. Node\* temp = **new** Node;
14. temp->data = data;
15. temp->next = NULL;
16. **return** temp;
17. }
19. **class** Solution
20. {
21. **public**:
22. Node\* removeDuplicated(Node\* node)
23. {
24. Node\* head = newNode(0);
25. head->next = node;
26. **while** (node != NULL && node->next != NULL)
27. {
28. //cout << node->next->data;
30. **if** (node->data == node->next->data)
31. {
32. Node\* temp = node->next;
33. node->next = node->next->next;
34. **delete** temp;
35. }
36. **else**
37. {
38. node = node->next;
39. }
40. }
42. **return** head->next;
43. }
44. };
46. **int** main()
47. {
48. Node\* node = newNode(1);
49. node->next = newNode(1);
50. node->next->next = newNode(1);
51. node->next->next->next = newNode(3);
52. node->next->next->next->next = newNode(3);
53. Solution a;
54. Node\* res = a.removeDuplicated(node);
55. **while** (res != NULL)
56. {
57. cout << res->data << " ";
58. res = res->next;
59. }
60. system("pause");
61. **return** 1;
62. }

**Section 05. Stack**

**020. Valid parentheses (E) [stack]**

**Problem:** Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

* Open brackets must be closed by the same type of brackets.
* Open brackets must be closed in the correct order.

**Note**: an empty string is also considered valid.

**Example:** Input: "{[]}"

Output: true

**Solution:**

1. #include <iostream>
2. #include <stack>
3. **using** **namespace** std;
5. **bool** validParentheses(string s)
6. {
7. stack<**char**> parentheses;
8. **for** (**int** i = 0; i < s.size(); i++)
9. {
10. **if** ((s[i]=='(')||(s[i]=='{')||(s[i]=='[')) parentheses.push(s[i]);
11. **else**
12. {
13. **if** (parentheses.empty()) **return** **false**;
14. **if** ((s[i] == ')') && (parentheses.top() != '(')) **return** **false**;
15. **if** ((s[i] == '}') && (parentheses.top() != '{')) **return** **false**;
16. **if** ((s[i] == ']') && (parentheses.top() != '[')) **return** **false**;
17. parentheses.pop();
18. }
19. }
20. **return** parentheses.empty();
21. }
23. **int** main()
24. {
25. string s = "([})";
26. cout << validParentheses(s) << endl;
27. system("pause");
28. **return** 1;
29. }

**032. Longest valid parentheses (H) [stack]**

**Problem:** Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring**.**

**Example:** Input: ")()())"

Output: 4

**Solution:**

1. #include <iostream>
2. #include <stack>
3. #include <algorithm>
4. **using** **namespace** std;
6. **int** longestValidParentheses(string s)
7. {
8. **int** start = 0, res = 0;
9. stack<**int**> st;
10. **for** (**int** i = 0; i < s.size(); i++)
11. {
12. **if** (s[i] == '(') st.push(i);
13. **else** **if** (s[i] == ')')
14. {
15. **if** (st.empty()) start = i + 1;
16. **else**
17. {
18. st.pop();
19. res = st.empty() ? max(res, i - start + 1) : max(res, i - st.top());
20. }
21. }
22. }
23. **return** res;
24. }
26. **int** main()
27. {
28. string s = ")(()))";    // ")()())"
29. cout << longestValidParentheses(s) << endl;
30. system("pause");
31. **return** 1;
32. }

**101. Symmetric tree (E) [tree, stack]**

**Problem:** Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

**Example:** this binary tree [1,2,2,3,4,4,3] is symmetric

1

/ \

2 2

/ \ / \

3 4 4 3

**Note:** Bonus points if you could solve it both recursively and iteratively.

**Solution:**

1. #include <iostream>
2. #include <stack>
3. **using** **namespace** std;
5. **struct** treeNode
6. {
7. **int** val;
8. treeNode\* left;
9. treeNode\* right;
10. treeNode(**int** val) : val(val), left(NULL), right(NULL) {}
11. };
13. // iterative
14. **class** Solution1
15. {
16. **public**:
17. **bool** isSymmetric(treeNode\* root)
18. {
19. **if** (!root) **return** **true**;
21. stack<treeNode\*> s;
22. s.push(root->left);
23. s.push(root->right);
24. **while** (!s.empty())
25. {
26. auto right = s.top();
27. s.pop();
28. auto left = s.top();
29. s.pop();
30. **if** ((!right) && (!left)) **continue**;
31. **if** ((!right) || (!left) || (left->val != right->val)) **return** **false**;
32. s.push(left->left);
33. s.push(right->right);
34. s.push(left->right);
35. s.push(right->left);
36. }
37. **return** **true**;
38. }
39. };
41. // recursive
42. **class** Solution2
43. {
44. **public**:
45. **bool** isSymmetric(treeNode\* root)
46. {
47. **if** (!root) **return** **true**;
48. **return** isSymmetric(root->left, root->right);
49. }
50. **private**:
51. **bool** isSymmetric(treeNode\* left, treeNode\* right)
52. {
53. **if** ((!left) && (!right)) **return** **true**;
54. **if** ((!left) || (!right) || (left->val != right->val)) **return** **false**;
55. **return** isSymmetric(left->left, right->right) && isSymmetric(left->right, right->left);
56. }
57. };
59. **int** main()
60. {
61. Solution2 a;
62. treeNode\* node = **new** treeNode(1);
63. node->left = **new** treeNode(2);
64. node->right = **new** treeNode(2);
65. node->left->left = **new** treeNode(3);
66. node->left->right = **new** treeNode(4);
67. node->right->left = **new** treeNode(4);
68. node->right->right = **new** treeNode(3);
69. cout << a.isSymmetric(node) << endl;
70. **delete** node;
71. system("pause");
72. **return** 1;
73. }

**150. Evaluate reverse polish notation (M) [stack]**

**Problem:** Evaluate the value of an arithmetic expression in Reverse Polish Notation. Valid operators are +, -, \*, /. Each operand may be an integer or another expression.

**Example:** Input: ["4", "13", "5", "/", "+"]

Output: 6

**Note:**

* Division between two integers should truncate toward zero.
* The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. #include <stack>
5. **using** **namespace** std;
7. **int** evaluateRPN(vector<string>& s)
8. {
9. **if** (s.size() == 1) **return** stoi(s[0]);
10. stack<**int**> st;
11. **for** (**int** i = 0; i < s.size(); i++)
12. {
13. **if** ((s[i] != "+") && (s[i] != "-") && (s[i] != "\*") && (s[i] != "/"))
14. {
15. st.push(stoi(s[i]));
16. }
17. **else**
18. {
19. **int** num1 = st.top();
20. st.pop();
21. **int** num2 = st.top();
22. st.pop();
23. **if** (s[i] == "+") st.push(num1 + num2);
24. **if** (s[i] == "-") st.push(num2 - num1);
25. **if** (s[i] == "\*") st.push(num2 \* num1);
26. **if** (s[i] == "/") st.push(num2 / num1);
27. }
28. }
29. **return** st.top();
30. }
32. **int** main()
33. {
34. vector<string> s = { "10", "6", "9", "3", "+", "-11", "\*", "/", "\*", "17", "+", "5", "+" };
35. cout << evaluateRPN(s) << endl;
36. system("pause");
37. **return** 1;
38. }

**155. Min stack (E) [stack]**

**Problem:** Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

* push(x) -- Push element x onto stack.
* pop() -- Removes the element on top of the stack.
* top() -- Get the top element.
* getMin() -- Retrieve the minimum element in the stack.

**Example**: MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); --> Returns -3.

minStack.pop();

minStack.top(); --> Returns 0.

minStack.getMin(); --> Returns -2.

**Solution:**

1. #include <iostream>
2. #include <stack>
3. **using** **namespace** std;
5. **class** MinStack
6. {
7. **public**:
8. **void** push(**int** val)
9. {
10. **if** (st.empty()) min.push(val);
11. **else**
12. {
13. **if** (val <= min.top()) min.push(val);
14. }
15. st.push(val);
16. }
18. **void** pop()
19. {
20. **if** (st.empty()) **return**;
21. **if** (st.top() == min.top()) min.pop();
22. st.pop();
23. }
25. **int** getMin()
26. {
27. **return** min.top();
28. }
30. **int** top()
31. {
32. **return** st.top();
33. }
35. **private**:
36. stack<**int**> min;
37. stack<**int**> st;
38. };
40. **int** main()
41. {
42. MinStack a;
43. a.push(-2);
44. a.push(0);
45. a.push(-3);
46. cout << a.getMin() << endl; // return -3
47. a.pop();
48. cout << a.top() << endl;    // return 0
49. cout << a.getMin() << endl; // return -2
50. system("pause");
51. **return** 1;
52. }

**224. Basic calculator (H) [stack]**

**Problem:** Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ), the plus + or minus sign -, non-negative integers and empty spaces.

**Example:** Input: "(1+(4+5+2)-3)+(6+8)"

Output: 23

**Note:**

* You may assume that the given expression is always valid.
* Do not use the eval built-in library function.

**Solution:**

1. #include <iostream>
2. #include <stack>
3. **using** **namespace** std;
5. **int** calculator(string s)
6. {
7. stack<**int**> st;
8. **int** res = 0, sign = 1;
9. **for** (**int** i = 0; i < s.size(); i++)
10. {
11. **if** ((s[i] >= '0') && (s[i] <= '9'))
12. {
13. **int** num = 0;
14. **while** ((s[i] >= '0') && (s[i] <= '9') && (i < s.size()))
15. {
16. num = num \* 10 + (s[i] - '0');
17. i++;
18. }
19. res += num \* sign;
20. i--;
21. }
22. **else** **if** (s[i] == '+')
23. {
24. sign = 1;
25. }
26. **else** **if** (s[i] == '-')
27. {
28. sign = -1;
29. }
30. **else** **if** (s[i] == '(')
31. {
32. st.push(res);
33. st.push(sign);
34. res = 0;
35. sign = 1;
36. }
37. **else** **if** (s[i] == ')')
38. {
39. res \*= st.top();
40. st.pop();
41. res += st.top();
42. st.pop();
43. }
44. }
45. **return** res;
46. }
48. **int** main()
49. {
50. cout << calculator("   (1+(4+5+2)-3)+(6+8)");
51. system("pause");
52. **return** 1;
53. }

**227. Basic calculator II (M) [stack]**

**Problem:** Implement a basic calculator to evaluate a simple expression string.

The expression string contains only non-negative integers, +, -, \*, / operators and empty spaces . The integer division should truncate toward zero.

**Example:** Input: " 3+5 / 2 "

Output: 5

**Note:**

* You may assume that the given expression is always valid.
* Do not use the eval built-in library function.

**Solution:**

1. #include <iostream>
2. #include <stack>
3. **using** **namespace** std;
5. **int** calculator(string s)
6. {
7. stack<**int**> st;
8. **long** res = 0, num = 0;
9. **char** op = '+';
10. **for** (**int** i = 0; i < s.size(); i++)
11. {
12. **if** ((s[i] >= '0') && (s[i] <= '9'))
13. {
14. **while** ((s[i] >= '0') && (s[i] <= '9') && (i < s.size()))
15. {
16. num = num \* 10 + (s[i] - '0');
17. i++;
18. }
19. i--;
20. }
21. **if** ((s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/') || (i == s.size() - 1))
22. {
23. **if** (op == '+')
24. {
25. st.push(num);
26. num = 0;
27. op = s[i];
28. }
29. **else** **if** (op == '-')
30. {
31. st.push((-1 \* num));
32. num = 0;
33. op = s[i];
34. }
35. **else** **if** (op == '\*')
36. {
37. num = st.top()\*num;
38. st.pop();
39. st.push(num);
40. num = 0;
41. op = s[i];
42. }
43. **else** **if** (op == '/')
44. {
45. num = st.top() / num;
46. st.pop();
47. st.push(num);
48. num = 0;
49. op = s[i];
50. }
51. }
52. }
53. **while** (!st.empty())
54. {
55. res += st.top();
56. st.pop();
57. }
58. **return** res;
59. }
61. **int** main()
62. {
63. cout << calculator(" 3 + 5 / 2 ") << endl;
64. system("pause");
65. **return** 1;
66. }

**232. Implement queue using stack (E) [stack]**

**Problem:** Implement the following operations of a queue using stacks.

* push(x) -- Push element x to the back of queue.
* pop() -- Removes the element from in front of queue.
* peek() -- Get the front element.
* empty() -- Return whether the queue is empty.

**Example:**

MyQueue queue = new MyQueue();

queue.push(1);

queue.push(2);

queue.peek(); // returns 1

queue.pop(); // returns 1

queue.empty(); // returns false

**Notes:**

* You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.
* Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
* You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

**Solution:**

1. #include <iostream>
2. #include <stack>
3. #include <queue>
4. **using** **namespace** std;
6. **class** QueueUsingStack
7. {
8. **public**:
9. QueueUsingStack() {}
11. **void** push(**int** x)
12. {
13. stack<**int**> t;
14. **while** (!s.empty())
15. {
16. t.push(s.top());
17. s.pop();
18. }
19. s.push(x);
20. **while** (!t.empty())
21. {
22. s.push(t.top());
23. t.pop();
24. }
25. }
27. **void** pop()
28. {
29. s.pop();
30. }
32. **int** peek()
33. {
34. **return** s.top();
35. }
37. **bool** empty()
38. {
39. **return** s.empty();
40. }
42. **private**:
43. stack<**int**> s;
44. };
46. **int** main()
47. {
48. QueueUsingStack q;
49. q.push(1);
50. q.push(2);
51. cout << q.peek() << endl;   // return 1
52. q.pop();
53. cout << q.empty() << endl;  // return false
54. system("pause");
55. **return** 1;
56. }

**716. Max stack (E) [stack]**

**Problem:** Design a max stack that supports push, pop, top, peekMax and popMax.

* push(x) -- Push element x onto stack.
* pop() -- Remove the element on top of the stack and return it.
* top() -- Get the element on the top.
* peekMax() -- Retrieve the maximum element in the stack.
* popMax() -- Retrieve the maximum element in the stack, and remove it. If you find more than one maximum elements, only remove the top-most one.

**Example:**

MaxStack stack = new MaxStack();

stack.push(5);

stack.push(1);

stack.push(5);

stack.top(); -> 5

stack.popMax(); -> 5

stack.top(); -> 1

stack.peekMax(); -> 5

stack.pop(); -> 1

stack.top(); -> 5

**Note:**

* -1e7 <= x <= 1e7
* Number of operations won't exceed 10000.
* The last four operations won't be called when stack is empty.

**Solution:**

1. #include <iostream>
2. #include <stack>
3. **using** **namespace** std;
5. **class** MaxStack
6. {
7. **public**:
8. MaxStack() {}
10. **void** push(**int** x)
11. {
12. **if** (st.empty())
13. {
14. maxSt.push(x);
15. }
16. **if** (maxSt.top() <= x)
17. {
18. maxSt.push(x);
19. }
20. st.push(x);
21. }
23. **int** pop()
24. {
25. **if** (maxSt.top() == st.top())
26. {
27. maxSt.pop();
28. }
29. **int** tmp = st.top();
30. st.pop();
31. **return** tmp;
32. }
34. **int** top()
35. {
36. **return** st.top();
37. }
39. **int** peekMax()
40. {
41. **return** maxSt.top();
42. }
44. **int** popMax()
45. {
46. **int** tmp = maxSt.top();
47. stack<**int**> t;
48. **while** (st.top() != maxSt.top())
49. {
50. t.push(st.top());
51. st.pop();
52. }
53. st.pop();
54. maxSt.pop();
55. **while** (!t.empty())
56. {
57. st.push(t.top());
58. t.pop();
59. }
60. **return** tmp;
61. }
63. **private**:
64. stack<**int**> st;
65. stack<**int**> maxSt;
66. };
68. **int** main()
69. {
70. MaxStack stack;
71. stack.push(5);
72. stack.push(1);
73. stack.push(5);
74. cout << stack.top() << endl;    // return 5
75. cout << stack.popMax() << endl; // return 5
76. cout << stack.top() << endl;    // return 1
77. cout << stack.peekMax() << endl;// return 5
78. stack.pop();
79. cout << stack.top() << endl;            // return 5
80. system("pause");
81. **return** 1;
82. }

**739. Daily temperatures (M) [stack]**

**Problem:** Given a list of daily temperatures T, return a list such that, for each day in the input, tells you how many days you would have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead.

**Example:** For example, given the list of temperatures T = [73, 74, 75, 71, 69, 72, 76, 73], your output should be [1, 1, 4, 2, 1, 1, 0, 0].

**Note:** The length of temperatures will be in the range [1, 30000]. Each temperature will be an integer in the range [30, 100].

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <stack>
4. **using** **namespace** std;
6. vector<**int**> dailyTemperatures(vector<**int**>& t)
7. {
8. vector<**int**> res(t.size(), 0);
9. **for** (**int** i = 0; i < t.size(); i++)
10. {
11. **for** (**int** j = i+1; j < t.size(); j++)
12. {
13. **if** (t[j] > t[i])
14. {
15. res[i] = j - i;
16. **break**;
17. }
18. }
19. }
20. **return** res;
21. }
23. vector<**int**> dailyTemperatures1(vector<**int**>& t)
24. {
25. vector<**int**> res(t.size(), 0);
26. stack<**int**> st;
27. **for** (**int** i = 0; i < t.size(); i++)
28. {
29. **while** ((!st.empty()) && (t[i] > t[st.top()]))
30. {
31. auto tmp = st.top();
32. st.pop();
33. res[tmp] = i - tmp;
34. }
35. st.push(i);
36. }
37. **return** res;
38. }
40. **int** main()
41. {
42. vector<**int**> t{ 73,74,75,71,69,72,76,73 };
43. vector<**int**> res = dailyTemperatures1(t);
44. **for** (**int** i = 0; i < res.size(); i++)
45. {
46. cout << res[i] << " ";
47. }
48. system("pause");
49. **return** 1;
50. }

**770. Basic calculator IV (E) [stack]**

Problem:

Example:

Solution:

**772. Basic calculator III (H) [recursion, string]**

**Problem:** Implement a basic calculator to evaluate a simple expression string.

The expression string contains only non-negative integers, +, -, \*, / operators , open ( and closing parentheses ) and empty spaces . The integer division should truncate toward zero.

You may assume that the given expression is always valid. All intermediate results will be in the range of [-2147483648, 2147483647].

**Example:** "(2+6\* 3+5- (3\*14/7+2)\*5)+3"=-12

**Note:** Do not use the eval built-in library function.

**Solution:**

1. #include <iostream>
2. #include <string>
3. **using** **namespace** std;
5. **int** calculator(string s)
6. {
7. **int** res = 0, cur = 0, num = 0;
8. **char** op = '+';
9. **for** (**int** i = 0; i < s.size(); i++)
10. {
11. **if** ((s[i] >= '0') && (s[i] <= '9'))
12. {
13. num = num \* 10 + (s[i] - '0');
14. }
16. **if** (s[i] == '(')
17. {
18. **int** j = i, cnt = 0;
19. **for** (; i < s.size(); i++)
20. {
21. **if** (s[i] == '(') cnt++;
22. **if** (s[i] == ')') cnt--;
23. **if** (cnt == 0) **break**;
24. }
25. num = calculator(s.substr(j + 1, i - j - 1));
26. }
28. **if** ((s[i] == '+') || (s[i] == '-') || (s[i] == '\*') || (s[i] == '/') || (i == s.size() - 1))
29. {
30. **switch** (op)
31. {
32. **case** '+': cur += num; **break**;
33. **case** '-': cur -= num; **break**;
34. **case** '\*': cur \*= num; **break**;
35. **case** '/': cur /= num; **break**;
36. }
37. **if** ((s[i] == '+') || (s[i] == '-') || (i == s.size() - 1))
38. {
39. res += cur;
40. cur = 0;
41. }
42. op = s[i];
43. num = 0;
44. }
45. }
46. **return** res;
47. }
49. **int** main()
50. {
51. cout << calculator("(2+6\* 3+5- (3\*14/7+2)\*5)+3");
52. system("pause");
53. **return** 1;
54. }

**Section 06. Queue**

**225. Implement stack using queue (H) [queue]**

**Problem:** Implement the following operations of a stack using queues.

* push(x) -- Push element x onto stack.
* pop() -- Removes the element on top of the stack.
* top() -- Get the top element.
* empty() -- Return whether the stack is empty.

**Example:**

MyStack stack = new MyStack();

stack.push(1);

stack.push(2);

stack.top(); // returns 2

stack.pop(); // returns 2

stack.empty(); // returns false

**Note:**

* You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
* Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
* You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

**Solution:**

1. #include <iostream>
2. #include <queue>
3. **using** **namespace** std;
5. **class** MyStack
6. {
7. **public**:
8. **void** push(**int** x)
9. {
10. queue<**int**> tmp;
11. **while** (!q.empty())
12. {
13. tmp.push(q.front());
14. q.pop();
15. }
16. q.push(x);
17. **while** (!tmp.empty())
18. {
19. q.push(tmp.front());
20. tmp.pop();
21. }
22. }
24. **void** pop()
25. {
26. q.pop();
27. }
29. **int** top()
30. {
31. **return** q.front();
32. }
34. **bool** empty()
35. {
36. **return** q.empty();
37. }

40. **private**:
41. queue<**int**> q;
42. };
44. **int** main()
45. {
46. MyStack s;
47. s.push(1);
48. s.push(2);
49. cout << s.top() << endl;    // return 2
50. s.pop();
51. cout << s.empty() << endl;  // return 0
52. system("pause");
53. **return** 1;
54. }

**239. Sliding window maximum (H) [queue]**

**Problem:** Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window.

**Example:**

Input: nums = [1,3,-1,-3,5,3,6,7], and k = 3

Output: [3,3,5,5,6,7]

**Solution:**

1. #include <vector>
2. #include <deque>
3. #include <iostream>
4. #include <algorithm>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. vector<**double**> slidingMax(vector<**double**>& num, **int** k)
11. {
12. vector<**double**> res;
13. deque <**double**> q;
15. **for** (**int** i = 0; i < num.size(); i++)
16. {
17. **if** (!q.empty() && (q.front() == i - k)) q.pop\_front();
18. **while** (!q.empty() && (num[q.back()] < num[i])) q.pop\_back();
19. q.push\_back(i);
21. **if** (i >= k - 1) res.push\_back(num[q.front()]);
22. }
23. **return** res;
24. }
25. };
27. **int** main()
28. {
29. vector<**double**> num{ 1, 3, -1, -3, 5, 3, 6, 7 };
30. Solution a;
31. vector<**double**> res = a.slidingMax(num, 3);
32. **for** (**int** i = 0; i < res.size(); i++)
33. {
34. cout << res[i] << " ";
35. }
36. system("pause");
37. **return** 0;
38. }

**281. Zigzag iterator (M) [queue]**

**Problem:** Given two 1d vectors, implement an iterator to return their elements alternately.

**Example:**

v1 = [1, 2]

v2 = [3, 4, 5, 6]

By calling next repeatedly until has Next returns false, the order of elements returned by next should be: [1, 3, 2, 4, 5, 6].

**Follow up:** What if you are given k 1d vectors? How well can your code be extended to such cases?

**Solution:**

1. #include <queue>
2. #include <iostream>
3. **using** **namespace** std;
5. **class** ZigzagIterator
6. {
7. **public**:
8. ZigzagIterator(vector<**int**>& v1, vector<**int**>& v2)
9. {
10. **if** (!v1.empty())
11. {
12. q.push(make\_pair(v1.begin(), v1.end()));
13. }
14. **if** (!v2.empty())
15. {
16. q.push(make\_pair(v2.begin(), v2.end()));
17. }
18. }
20. **int** next()
21. {
22. auto it = q.front().first, end = q.front().second;
23. q.pop();
24. **if** (it + 1 != end)
25. {
26. q.push(make\_pair(it + 1, end));
27. }
28. **return** \*it;
29. }
31. **bool** hasNext()
32. {
33. **return** !q.empty();
34. }
36. **private**:
37. queue<pair<vector<**int**>::iterator, vector<**int**>::iterator>> q;
38. };
40. **int** main()
41. {
42. vector<**int**> v1{ 1, 2 }, v2{ 3, 4, 5, 6 };
43. ZigzagIterator a(v1, v2);
44. **while** (a.hasNext())
45. {
46. cout << a.next() << " ";
47. }
48. system("pause");
49. **return** 1;
50. }

**346. Moving average from data stream (E) [queue]**

**Problem:** Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

**Example:**

MovingAverage m = new MovingAverage(3);

m.next(1) = 1 // return 1.00000

m.next(10) = (1 + 10) / 2 // return 5.50000

m.next(3) = (1 + 10 + 3) / 3 // return 4.66667

m.next(5) = (10 + 3 + 5) / 3 // return 6.00000

**Solution:**

1. #include <queue>
2. #include <iostream>
3. **using** **namespace** std;
5. **class** Movingaverage
6. {
7. **public**:
8. Movingaverage(**int** size)
9. {
10. window = size;
11. sum = 0;
12. }
14. **double** get(**double** val)
15. {
16. sum += val;
17. stream.push(val);
18. **if** (stream.size() > window)
19. {
20. sum -= stream.front();
21. stream.pop();
22. }
23. **return** sum / window;
24. }
25. **private**:
26. **double** sum;
27. **int** window;
28. queue<**double**> stream;
29. };
31. **int** main()
32. {
33. Movingaverage a(3);
34. cout << a.get(1) << endl;
35. cout << a.get(2) << endl;
36. cout << a.get(3) << endl;
37. cout << a.get(4) << endl;
38. system("pause");
39. **return** 1;
40. }

**Section 09. Hash table**

**001. Two sum (E) [hash table, array]**

**Problem:** Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same **element** twice.

**Example:** nums = [2, 7, 11, 15], target = 9,

return [0, 1].

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <unordered\_map>
4. **using** **namespace** std;
6. vector<**int**> twoSum(vector<**int**>& nums, **int** target)
7. {
8. vector<**int**> res;
9. unordered\_map<**int**, **int**> m;
10. **for** (**int** i = 0; i < nums.size(); i++)
11. {
12. m[nums[i]] = i;
13. }
14. **for** (**int** i = 0; i < nums.size(); i++)
15. {
16. **if** ((m.count(target - nums[i])) && (m[target - nums[i]] != i))
17. {
18. res.push\_back(i);
19. res.push\_back(m[target - nums[i]]);
20. **break**;
21. }
22. }
23. **return** res;
24. }
26. **int** main()
27. {
28. vector<**int**> nums{ 2, 7, 11, 15 };
29. **int** target = 9;
30. vector<**int**> res = twoSum(nums, target);
31. **for** (**int** i = 0; i < res.size(); i++)
32. {
33. cout << res[i] << " ";
34. }
35. system("pause");
36. **return** 1;
37. }

**003. Longest substring without repeating characters (M) [hash table, string]**

**Problem:** Given a string, find the length of the longest substring without repeating characters.

**Example:** Input: "pwwkew"

Output: 3

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
4. #include <algorithm>
5. **using** **namespace** std;
7. **int** longestSubstring(string s)
8. {
9. unordered\_map<**char**, **int**> m;
10. **int** len = 0, left = -1;
11. **for** (**int** i = 0; i < s.size(); i++)
12. {
13. **if** ((m.count(s[i])) && (m[s[i]] > left))
14. {
15. left = m[s[i]];
16. }
17. m[s[i]] = i;
18. len = max(len, i - left);
19. }
20. **return** len;
21. }
23. **int** main()
24. {
25. cout << longestSubstring("pwwkew") << endl; // "bbbbbb"
26. system("pause");
27. **return** 1;
28. }

**049. Group anagrams (M) [hash table, string]**

**Problem:** Given an array of strings, group anagrams together.

**Example:** Input: ["eat", "tea", "tan", "ate", "nat", "bat"],

Output: [ ["ate","eat","tea"],

["nat","tan"],

["bat"] ]

**Note:**

* All inputs will be in lowercase.
* The order of your output does not matter.

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
4. #include <vector>
5. #include <algorithm>
6. **using** **namespace** std;
8. vector<vector<string>> groupAnagrams(vector<string>& s)
9. {
10. vector<vector<string>> res;
11. unordered\_map<string, vector<string>> m;
12. **for** (auto str:s)
13. {
14. string tmp = str;
15. sort(tmp.begin(), tmp.end());
16. m[tmp].push\_back(str);
17. }
18. **for** (auto a : m)
19. {
20. res.push\_back(a.second);
21. }
22. **return** res;
23. }
25. **int** main()
26. {
27. vector<string> str{ "eat", "tea", "tan", "ate", "nat", "bat" };
28. vector<vector<string>> res = groupAnagrams(str);
29. **for** (**int** i = 0; i < res.size(); i++)
30. {
31. **for** (**int** j = 0; j < res[i].size(); j++)
32. {
33. cout << res[i][j] << " ";
34. }
35. cout << endl;
36. }
37. system("pause");
38. **return** 1;
39. }

**159. Longest substring with at most two distinct characters (H) [hash table, string]**

**Problem:** Given a string S, find the length of the longest substring T that contains at most two distinct characters.

**Example:** Given S = “eceba”,

T is “ece” which its length is 3.

**Solution:**

1. #include <iostream>
2. #include <unordered\_map>
3. #include <string>
4. #include <algorithm>
5. **using** **namespace** std;
7. **int** longestSubstring(string s)
8. {
9. **int** len = 0, left = 0;
10. unordered\_map<**char**, **int**> m;
11. **for** (**int** i = 0; i < s.size(); i++)
12. {
13. m[s[i]]++;
14. **while** (m.size() > 2)
15. {
16. m[s[left]]--;
17. **if** (m[s[left]] == 0) m.erase(s[left]);
18. left++;
19. }
20. len = max(len, i - left + 1);
21. }
22. **return** len;
23. }
25. **int** main()
26. {
27. cout << longestSubstring("eceba") << endl;
28. system("pause");
29. **return** 1;
30. }

**170. Two sum III (E) [hash table]**

**Problem:** Design and implement a TwoSum class. It should support the following operations:add and find.

add - Add the number to an internal data structure.

find - Find if there exists any pair of numbers which sum is equal to the value.

**Example:**

add(1); add(3); add(5);

find(4) -> true

find(7) -> false

**Solution:**

1. #include <iostream>
2. #include <unordered\_map>
3. **using** **namespace** std;
5. **class** TwoSum
6. {
7. **public**:
8. TwoSum() {}
10. **void** add(**int** val)
11. {
12. ++m[val];
13. }
15. **bool** find(**int** val)
16. {
17. **for** (auto x : m)
18. {
19. **int** target = val - x.first;
20. **if** (((target!=x.first)&&(m.count(target)))||((target==x.first)&&(x.second>1)))
21. {
22. **return** **true**;
23. }
24. }
25. **return** **false**;
26. }
27. **private**:
28. unordered\_map<**int**, **int**> m;
29. };
31. **int** main()
32. {
33. TwoSum a;
34. a.add(1);
35. a.add(1);
36. a.add(1);
37. cout << a.find(1) << endl;
38. cout << a.find(2) << endl;
39. cout << a.find(3) << endl;
40. system("pause");
41. **return** 1;
42. }

**202. Happy number (E) [hash table, math]**

**Problem:** Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

**Example:** Input: 19

Output: true

**Solution:**

1. #include <unordered\_set>
2. #include <iostream>
3. **using** **namespace** std;
5. **bool** isHappyNumber(**int** n)
6. {
7. unordered\_set<**int**> s;
8. **while** (n != 1)
9. {
10. **int** sum = 0;
11. **while** (n)
12. {
13. sum += (n % 10)\*(n % 10);
14. n /= 10;
15. }
16. n = sum;
17. **if** (s.count(n)) **break**;
18. s.insert(n);
19. }
20. **return** n == 1;
21. }
23. **int** main()
24. {
25. cout << isHappyNumber(11) << endl;
26. system("pause");
27. **return** 1;
28. }

**205. Isomorphic string (E) [hash table]**

**Problem:** Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

**Example:** Input: s = "paper", t = "title"

Output: true

**Note:** You may assume both s and t have the same length.

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
4. **using** **namespace** std;

7. **bool** isIsomorphic(string s, string t) {
8. unordered\_map<**char**, **char**> st;
9. unordered\_map<**char**, **char**> ts;
10. **for** (**int** i = 0; i < s.size(); i++)
11. {
12. **if** ((st.count(s[i])) && (t[i] != st[s[i]])) **return** **false**;
13. **if** ((ts.count(t[i])) && (s[i] != ts[t[i]])) **return** **false**;
14. st[s[i]] = t[i];
15. ts[t[i]] = s[i];
16. }
17. **return** **true**;
18. }


22. **int** main()
23. {
24. cout << isIsomorphic("egg", "add");
25. system("pause");
26. **return** 1;
27. }

**217. Contains duplicate (E) [hash table, array]**

**Problem:** Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

**Example:** Input: [1,2,3,1]

Output: true

**Solution:**

1. #include <iostream>
2. #include <unordered\_set>
3. #include <vector>
4. **using** **namespace** std;
6. **bool** containsDuplicate(vector<**int**>& nums)
7. {
8. unordered\_set<**int**> s;
9. **for** (**int** i = 0; i < nums.size(); i++)
10. {
11. **if** (s.count(nums[i])) **return** **true**;
12. s.insert(nums[i]);
13. }
14. **return** **false**;
15. }
17. **int** main()
18. {
19. vector<**int**> nums{ 1, 3, 2, 1 };
20. cout << containsDuplicate(nums) << endl;
21. system("pause");
22. **return** 1;
23. }

**219. Contains duplicate (E) [hash table, array]**

**Problem:** Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the absolute difference between i and j is at most k.

**Example:** Input: nums = [1,2,3,1,2,3], k = 2

Output: false

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <unordered\_map>
4. **using** **namespace** std;
6. **bool** containsNearbyDuplicate(vector<**int**>& nums, **int** k) {
7. unordered\_map<**int**, **int**> m;
8. **for** (**int** i = 0; i < nums.size(); i++)
9. {
10. **if** (m.count(nums[i]))
11. {
12. **if** (i - m[nums[i]] <= k) **return** **true**;
13. }
14. m[nums[i]] = i;
15. }
16. **return** **false**;
17. }
19. **int** main()
20. {
21. vector<**int**> nums{ 1, 2, 3, 1, 2, 3 };
22. **int** k = 2;
23. cout << containsNearbyDuplicate(nums, 2) << endl;
24. system("pause");
25. **return** 1;
26. }

**249. Group shifted strings (H) [hash table, string]**

**Problem:** Given a string, we can "shift" each of its letter to its successive letter, for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence: "abc" -> "bcd" -> ... -> "xyz"

Given a list of strings which contains only lowercase alphabets, group all strings that belong to the same shifting sequence.

**Example:** Input: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"],

Output: [ ["abc","bcd","xyz"],

["az","ba"],

["acef"],

["a","z"] ]

**Note:** For the return value, each inner list's elements must follow the lexicographic order.

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
4. #include <set>
5. #include <vector>
6. **using** **namespace** std;
8. vector<vector<string>> groupShifted(vector<string>& strs)
9. {
10. vector<vector<string>> res;
11. unordered\_map<string, multiset<string>> m;
12. **for** (auto str : strs)
13. {
14. string t = "";
15. **for** (auto c : str)
16. {
17. t += to\_string((c - str[0] + 26) % 26) + ","; // key...
18. }
19. m[t].insert(str);
20. }
21. **for** (auto it = m.begin(); it != m.end(); it++)
22. {
23. res.push\_back(vector<string>(it->second.begin(), it->second.end()));
24. }
25. **return** res;
26. }
28. **int** main()
29. {
30. vector<string> strs{ "abc", "bcd", "acef", "xyz", "az", "ba", "a", "z" };
31. vector<vector<string>> res = groupShifted(strs);
32. **for** (**int** i = 0; i < res.size(); i++)
33. {
34. **for** (**int** j = 0; j < res[i].size(); j++)
35. {
36. cout << res[i][j] << " ";
37. }
38. cout << endl;
39. }
40. system("pause");
41. **return** 1;
42. }

**290. Word pattern (E) [hash table]**

**Problem:** Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

**Example:** Input: pattern = "abba", str = "dog cat cat dog"

Output: true

**Note:** You may assume pattern contains only lowercase letters, and str contains lowercase letters that may be separated by a single space.

**Solution:**

1. #include <iostream>
2. #include <sstream>
3. #include <string>
4. #include <unordered\_map>
5. **using** **namespace** std;

8. **bool** wordPattern(string pattern, string strs) {
9. unordered\_map<**char**, string> ps;
10. unordered\_map<string, **char**> sp;
12. istringstream in(strs);
13. **int** i = 0;
14. **for** (string str; in >> str; i++)
15. {
16. **if** ((ps.count(pattern[i])) && (ps[pattern[i]] != str)) **return** **false**;
17. **if** ((sp.count(str)) && (sp[str] != pattern[i])) **return** **false**;
18. ps[pattern[i]] = str;
19. sp[str] = pattern[i];
20. }
21. **return** i == pattern.size();
22. }
24. **int** main()
25. {
26. cout << wordPattern("abba", "dog cat cat dog") << endl;
27. system("pause");
28. **return** 1;
29. }

**340. Longest substring with at most K distinct characters (H) [hash table, string]**

**Peoblem:** Given a string, find the length of the longest substring T that contains at most k distinct characters.

**Example:** Given s = “eceba” and k = 2,

T is "ece" which its length is 3

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
4. #include <algorithm>
5. **using** **namespace** std;
7. **int** longestSubstring(string s, **int** k)
8. {
9. **int** left = 0, len = 0;
10. unordered\_map<**char**, **int**> m;
11. **for** (**int** i = 0; i < s.size(); i++)
12. {
13. m[s[i]]++;
14. **while** (m.size() > k)
15. {
16. m[s[left]]--;
17. **if** (m[s[left]] == 0) m.erase(s[left]);
18. left++;
19. }
20. len = max(len, i - left + 1);
21. }
22. **return** len;
23. }
25. **int** main()
26. {
27. cout << longestSubstring("eceba", 2) << endl;
28. system("pause");
29. **return** 1;
30. }

**Section 10. Math**

**007. Reverse integer (E) [math]**

**Problem:** Given a 32-bit signed integer, reverse digits of an integer.

**Example:** Input: -123

Output: -321

**Note:** Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231, 231 − 1]. For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **int** reverseInteger(**int** x)
9. {
10. **int** res = 0;
11. **while** (x)
12. {
13. **if** (abs(res) > INT\_MAX / 10) **return** 0;
14. res = res \* 10 + x % 10;
15. x /= 10;
16. }
17. **return** res;
18. }
19. };
21. **int** main()
22. {
23. Solution a;
24. cout << a.reverseInteger(123) << endl; // 2^31-1=2147483647
25. system("pause");
26. **return** 1;
27. }

**009. Palindrome number (E) [math]**

**Problem:** Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

**Example:** Input: 121 Output: true

Input: -121 Output: false

**Follow up:** could you solve it without converting the integer to a string?

**Solution:**

1. #include <iostream>
2. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **bool** palindromeNumber(**int** x)
10. {
11. **if** (x < 0) **return** **false**;
13. string num = to\_string(x);
14. **int** i = 0, j = num.size() - 1;
16. **while** (i < j)
17. {
18. **if** (num[i] != num[j]) **return** **false**;
19. i++;
20. j--;
21. }
22. **return** **true**;
23. }
24. };
26. **class** Solution2
27. {
28. **public**:
29. **bool** palindromeNumber(**int** x)
30. {
31. **if** (x < 0) **return** **false**;
33. **int** temp = x, reverse = 0;
34. **while** (temp)
35. {
36. **if** (isOverflow(reverse, temp % 10)) **return** **false**;
37. **else**
38. {
39. reverse = reverse \* 10 + temp % 10;
40. temp /= 10;
41. }
42. }
43. **return** reverse == x;
44. }
46. **private**:
47. **bool** isOverflow(**int** q, **int** r)
48. {
49. **static** **const** **int** q\_max = numeric\_limits<**int**>::max() / 10;
50. **static** **const** **int** r\_max = numeric\_limits<**int**>::max() % 10;
51. **return**(q > q\_max) || ((q == q\_max) && (r > r\_max));
52. }
53. };
55. **int** main()
56. {
57. Solution2 a;
58. cout << a.palindromeNumber(121) << endl; // -121
59. system("pause");
60. **return** 1;
61. }

**012. Integer to roman (M) [math]**

**Problem:** Given an integer, convert it to a roman numeral. Input is guaranteed to be within the range from 1 to 3999.

**Example:** Input: 1994

Output: "MCMXCIV"

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

**Analysis:** similar to 013…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. string integerToRoman(**int** x)
11. {
12. vector<string> rom = { "M", "CM", "D", "CD", "C",
13. "XC", "L", "XL", "X", "IX", "V", "IV", "I" };
15. vector<**int**> num = { 1000, 900, 500, 400, 100,
16. 90, 50, 40, 10, 9, 5, 4, 1 };
18. string res;
19. **int** i = 0;
20. **while** (x > 0)
21. {
22. **int** times = x / num[i];
23. **while** (times)
24. {
25. x -= num[i];
26. res.append(rom[i]);
27. times--;
28. }
29. i++;
30. }
31. **return** res;
32. }
33. };
35. **int** main()
36. {
37. Solution a;
38. cout << a.integerToRoman(1994) << endl;
39. system("pause");
40. **return** 1;
41. }

**013. Roman to integer (E)**

**Problem:** Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol Value

I 1

V 5

X 10

L 50

C 100

D 500

M 1000

For example, two is written as II in Roman numeral, just two one's added together. Twelve is written as, XII, which is simply X + II. The number twenty seven is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

I can be placed before V (5) and X (10) to make 4 and 9.

X can be placed before L (50) and C (100) to make 40 and 90.

C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

**Example:** Input: "MCMXCIV"

Output: 1994

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

**Analysis:**

* hash map
* subtract one more lower digit…

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <unordered\_map>
5. **using** **namespace** std;
7. **class** Solution
8. {
9. **public**:
10. **int** romanToInt(string str)
11. {
12. unordered\_map<**char**, **int**> rom\_num = {
13. {'I', 1},
14. {'V', 5},
15. {'X', 10},
16. {'L', 50},
17. {'C', 100},
18. {'D', 500},
19. {'M', 1000}
20. };
21. **int** res = 0;
22. **for** (**int** i = 0; i < str.size(); i++)
23. {
24. **if** ((i > 0) && (rom\_num[str[i]] > rom\_num[str[i - 1]]))
25. {
26. res += rom\_num[str[i]] - 2 \* rom\_num[str[i - 1]];
27. }
28. **else**
29. {
30. res += rom\_num[str[i]];
31. }
32. }
33. **return** res;
34. }
35. };
37. **int** main()
38. {
39. Solution a;
40. cout << a.romanToInt("MCMXCIV") << endl;
41. system("pause");
42. **return** 0;
43. }

**168. Excel sheet column title (E)**

**Problem:** Given a positive integer, return its corresponding column title as appear in an Excel sheet.

**Example:** Input: 701

Output: "ZY"

Notice: first subtract 1 then…

**Solution:**

1. #include <iostream>
2. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. string excelTitle(**int** x)
10. {
11. string str;
12. **while** (x)
13. {
14. str.push\_back((x - 1) % 26 + 'A');
15. x = (x - 1) / 26;
16. }
17. reverse(str.begin(), str.end());
18. **return** str;
19. }
20. };
22. **int** main()
23. {
24. Solution a;
25. cout << a.excelTitle(701) << endl;
26. system("pause");
27. **return** 1;
28. }

**171. Excel sheet column number (E)**

**Problem:** Given a column title as appear in an Excel sheet, return its corresponding column number.

**Example:** Input: "AB" Output: 28

Input: "ZY" Output: 701

**Solution:**

1. #include <iostream>
2. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** excelToNumber(string str)
10. {
11. **int** result = 0;
12. **for** (**int** i = str.size()-1; i >= 0; i--)
13. {
14. result += (str[i] - 'A' + 1) \* pow(26, str.size() -1 - i);
15. }
16. **return** result;
17. }
18. };
20. **int** main()
21. {
22. Solution a;
23. cout << a.excelToNumber("AB") << endl;
24. system("pause");
25. **return** 1;
26. }

**172. Factorial trailing zeros (E)**

**Problem:** Given an integer n, return the number of trailing zeroes in n!.

**Example:** Input: 5

Output: 1

Explanation: 5! = 120, one trailing zero

**Note:** Your solution should be in logarithmic time complexity.

**Solution:**

1. #include <iostream>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **int** fractorialTrailingZeros(**int** x)
9. {
10. **int** num = 1;
11. **for** (**int** i = x; i >= 1; i--)
12. {
13. num \*= i;
14. }
15. **int** res = 0;
16. **while** (num % 10 == 0)
17. {
18. num /= 10;
19. res++;
20. }
21. **return** res;
22. }
23. };
25. **class** Solution2
26. {
27. **public**:
28. **int** fractorialTrailingZeros(**int** x)
29. {
30. **int** res = 0;
31. **while** (x)
32. {
33. res += x / 5;
34. x /= 5;
35. }
36. **return** res;
37. }
38. };
40. **int** main()
41. {
42. Solution2 a;
43. cout << a.fractorialTrailingZeros(10) << endl;
44. system("pause");
45. **return** 1;
46. }

**204. Count prime (E) [math]**

**Problem:** Count the number of prime numbers less than a non-negative number, n.

**Example:** Input: 10

Output: 4

**Solution:**

1. #include <vector>
2. #include <iostream>
3. **using** **namespace** std;
5. **int** countPrime(**int** n)
6. {
7. **int** res = 0;
8. vector<**bool**> prime(n, **true**);
9. **for** (**int** i = 2; i < n; i++)
10. {
11. **if** (prime[i])
12. {
13. res++;
14. **for** (**int** j = 2; i \* j < n; j++)
15. {
16. prime[i\*j] = **false**;
17. }
18. }
19. }
20. **return** res;
21. }
23. **int** main()
24. {
25. cout << countPrime(10) << endl;
26. system("pause");
27. **return** 1;
28. }

**223. Rectangle area (M)**

**Problem:** Find the total area covered by two rectilinear rectangles in a 2D plane. Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.

**Example:** Input: A = -3, B = 0, C = 3, D = 4, E = 0, F = -1, G = 9, H = 2

Output: 45

**Note:** Assume that the total area is never beyond the maximum possible value of int.

**Solution:**

1. #include <iostream>
2. #include <algorithm>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **int** computeArea(**int** A, **int** B, **int** C, **int** D,
10. **int** E, **int** F, **int** G, **int** H)
11. {
12. **return** (D - B) \* (C - A) +
13. (G - E) \* (H - F) -
14. max(0, (min(C, G) - max(A, E))) \*
15. max(0, (min(D, H) - max(B, F)));
17. }
18. };
20. **int** main()
21. {
22. Solution a;
23. cout << a.computeArea(-3, 0, 3, 4, 0, -1, 9, 2) << endl;
24. system("pause");
25. **return** 1;
26. }

**263. Ulgy number (E) [math]**

**Problem:** Write a program to check whether a given number is an ugly number. Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.

**Example:** Input: 6

Output: true

**Solution:**

1. #include <iostream>
2. **using** **namespace** std;
4. **bool** isUlgy(**int** num)
5. {
6. **while** (num >= 2)
7. {
8. **if** (num % 2 == 0) num /= 2;
9. **else** **if** (num % 3 == 0) num /= 3;
10. **else** **if** (num % 5 == 0) num /= 5;
11. **else** **return** **false**;
12. }
13. **return** num == 1;
14. }
16. **int** main()
17. {
18. cout << isUlgy(6) << endl;
19. system("pause");
20. **return** 1;
21. }

**264. Ulgy number II (M) [math]**

**Problem:** Write a program to find the n-th ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.

**Example:** Input: n = 10

Output: 12

**Note:**

* 1 is typically treated as an ugly number.
* n does not exceed 1690.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **int** ulgyNumber(**int** n)
7. {
8. vector<**int**> res(1, 1);
9. **int** i2 = 0, i3 = 0, i5 = 0;
10. **while** (res.size() < n)
11. {
12. **int** m2 = res[i2] \* 2, m3 = res[i3] \* 3, m5 = res[i5] \* 5;
13. **int** mn = min(m2, min(m3, m5));
14. **if** (mn == m2) i2++;
15. **if** (mn == m3) i3++;
16. **if** (mn == m5) i5++;
17. res.push\_back(mn);
18. }
19. **return** res[n - 1];
20. }
22. **int** main()
23. {
24. cout << ulgyNumber(20) << endl;
25. system("pause");
26. **return** 1;
27. }

**313. Super ugly number (M) [math]**

**Problem: Write a program to find the nth super ugly number.**

Super ugly numbers are positive numbers whose all prime factors are in the given prime list primes of size k.

**Example:** Input: n = 12, primes = [2,7,13,19]

Output: 32

**Note:**

* 1 is a super ugly number for any given primes.
* The given numbers in primes are in ascending order.
* 0 < k ≤ 100, 0 < n ≤ 106, 0 < primes[i] < 1000.
* The nth super ugly number is guaranteed to fit in a 32-bit signed integer.

**Solution:**

1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
4. **using** **namespace** std;
6. **int** superUglyNumber(**int** n, vector<**int**>& primes)
7. {
8. vector<**int**> res(1, 1);
9. vector<**int**> idx(primes.size(), 0);
11. **while** (res.size() < n)
12. {
13. vector<**int**> tmp;
14. **int** x = INT\_MAX;
15. **for** (**int** i = 0; i < primes.size(); i++)
16. {
17. tmp.push\_back(primes[i] \* res[idx[i]]);
18. }
19. **for** (**int** i = 0; i < primes.size(); i++)
20. {
21. x = min(x, tmp[i]);
22. }
23. **for** (**int** i = 0; i < primes.size(); i++)
24. {
25. **if** (x == tmp[i]) ++idx[i];
26. }
27. res.push\_back(x);
28. }
29. **return** res[n - 1];
30. }
32. **int** main()
33. {
34. vector<**int**> primes{ 2, 7, 13, 19 };
35. **int** n = 12;
36. cout << superUglyNumber(n, primes) << endl;
37. system("pause");
38. **return** 1;
39. }

**463. Island perimeter (E) [math]**

**Problem:** You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water.

Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

**Example:** Input: [ [0,1,0,0],

[1,1,1,0],

[0,1,0,0],

[1,1,0,0] ]

Output: 16



**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** islandPerimeter(vector<vector<**int**>>& grid)
6. {
7. **if** ((grid.empty()) || (grid[0].empty())) **return** 0;
8. **int** res = 0;
9. **for** (**int** i = 0; i < grid.size(); i++)
10. {
11. **for** (**int** j = 0; j < grid[i].size(); j++)
12. {
13. **if** (grid[i][j] == 0) **continue**;
14. **if** ((j == 0) || (grid[i][j - 1] == 0)) res++;
15. **if** ((j == grid[i].size() - 1) || (grid[i][j + 1] == 0)) res++;
16. **if** ((i == 0) || (grid[i - 1][j] == 0)) res++;
17. **if** ((i == grid.size() - 1) || (grid[i + 1][j] == 0)) res++;
18. }
19. }
20. **return** res;
21. }
23. **int** main()
24. {
25. vector<vector<**int**>> grid{ {0,1,0,0}, {1,1,1,0}, {0,1,0,0},{1,1,0,0} };
26. cout << islandPerimeter(grid) << endl;
27. system("pause");
28. **return** 1;
29. }

**Section 11. Two Pointers**

**141. Linked list cycle (E) [two pointers, linked list]**

**Problem:** Given a linked list, determine if it has a cycle in it.

To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

**Example:** Input: head = [3,2,0,-4], pos = 1

Output: true

**Analysis:** easy…

**Solution:**

1. #include <iostream>
2. **using** **namespace** std;
4. **struct** Node
5. {
6. **int** data;
7. Node\* next;
8. };
10. Node\* newNode(**int** x)
11. {
12. Node\* temp = **new** Node;
13. temp->data = x;
14. temp->next = NULL;
15. **return** temp;
16. }
18. **class** Solution
19. {
20. **public**:
21. **bool** hasCycle(Node\* n)
22. {
23. **if** (n == NULL ) **return** **false**;
25. Node\* slow = n;
26. Node\* fast = n;
27. **while** ((fast->next != NULL) && (fast->next->next != NULL))
28. {
29. slow = slow->next;
30. fast = fast->next->next;
31. **if** (slow == fast) **return** **true**;
32. }
33. **return** **false**;
34. }
35. };

**142. Linked list cycle II (M) [two pointers, linked list]**

**Problem:** Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

**Note:** Do not modify the linked list.

**Example:** Input: head = [3,2,0,-4], pos = 1

Output: tail connects to node index 1

**Analysis:**

when ’s meet, assume fast travel 2 step, then slow travel step…

circle length: r

linked list length: l

entry point (distance from head to cycle begin): a

distance from entry point to meeting point:

distance from head to cycle begin = distance from meeting point to entry point

**Solution:**

1. #include <iostream>
2. **using** **namespace** std;
4. **struct** Node
5. {
6. **int** data;
7. Node\* next;
8. };
10. Node\* newNode(**int** x)
11. {
12. Node\* temp = **new** Node;
13. temp->data = x;
14. temp->next = NULL;
15. **return** temp;
16. }
18. **class** Solution
19. {
20. **public**:
21. Node\* detectCycle(Node\* n)
22. {
23. **if** (n == NULL) **return** n;
25. Node\* slow = n;
26. Node\* fast = n;
27. **while** ((fast->next != NULL) && (fast->next->next != NULL))
28. {
29. slow = slow->next;
30. fast = fast->next->next;
31. **if** (fast == slow) **break**;
32. }
33. **if** ((fast->next == NULL) || (fast->next->next == NULL)) **return** NULL;
34. Node\* tmp = n;
35. **while** (slow != tmp)
36. {
37. slow = slow->next;
38. tmp = tmp->next;
39. }
40. **return** tmp;
41. }
42. };

**167. Two sum II (E) [two pointers]**

**Problem:** Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

**Note:**

* Your returned answers (both index1 and index2) are not zero-based.
* You may assume that each input would have exactly one solution and you may not use the same element twice.

**Example:** Input: numbers = [2,7,11,15], target = 9

Output: [1,2]

**Solution:**

1. #include <iostream>
2. #include <unordered\_map>
3. #include <vector>
4. **using** **namespace** std;
6. vector<**int**> twoSum(vector<**int**>& nums, **int** target)
7. {
8. vector<**int**> res;
9. unordered\_map<**int**, **int**> m;
10. **for** (**int** i = 0; i < nums.size(); i++)
11. {
12. m[nums[i]] = i + 1;
13. }
15. **for** (**int** i = 0; i < nums.size(); i++)
16. {
17. **if** ((m.count(target - nums[i])) && (m[target - nums[i]] != i + 1))
18. {
19. res.push\_back(i + 1);
20. res.push\_back(m[target - nums[i]]);
21. **break**;
22. }
23. }
24. **return** res;
25. }
27. vector<**int**> twoSum1(vector<**int**>& nums, **int** target)
28. {
29. **int** l = 0, r = nums.size() - 1;
30. **while** (l < r)
31. {
32. **if** (nums[l] + nums[r] == target) **return** { l + 1, r + 1 };
33. **else** **if** (nums[l] + nums[r] < target) l++;
34. **else** r--;
35. }
36. **return** {};
37. }
39. **int** main()
40. {
41. vector<**int**> nums{ 2, 7, 11, 15 };
42. **int** target = 9;
43. vector<**int**> res = twoSum1(nums, target);
44. **for** (**int** i = 0; i < res.size(); i++)
45. {
46. cout << res[i] << " ";
47. }
48. system("pause");
49. **return** 1;
50. }

**Section 12. Sort**

**088. Merge sorted array (E)**

**Problem:** Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

* The number of elements initialized in nums1 and nums2 are m and n respectively.
* You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2.

**Example:** Input: nums1 = [1,2,3,0,0,0], m = 3; nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

**Analysis:**

**Solution:**

1. #include <iostream>
2. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<**int**> mergeSortedArray(vector<**int**>& A, **int** m,
10. vector<**int**>& B, **int** n)
11. {
12. **int** pa = m - 1, pb = n - 1, pc = m + n - 1;
13. **while** ((pa >= 0) && (pb >= 0))
14. {
15. **if** (A[pa] > B[pb])
16. {
17. A[pc--] = A[pa--];
18. }
19. **else**
20. {
21. A[pc--] = B[pb--];
22. }
23. }
25. **while** (pa >= 0) A[pc--] = A[pa--];
26. **while** (pb >= 0) A[pc--] = B[pb--];
27. **return** A;
28. }
29. };
31. **int** main()
32. {
33. Solution a;
34. vector<**int**> A{ 1,2,3,0,0,0 };
35. vector<**int**> B{ 2,5,6 };
36. vector<**int**> res = a.mergeSortedArray(A, 3, B, 3);
37. **for** (**int** i = 0; i < res.size(); i++)
38. cout << res[i] << ", ";
39. system("pause");
40. **return** 1;
41. }

**252. Meeting room (E)**

**Problem:** Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), determine if a person could attend all meetings.

**Example:** Given [[0, 30],[5, 10],[15, 20]], return false.

**Analysis:** define a Interval struc…

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **struct** Interval
8. {
9. **int** start;
10. **int** end;
11. Interval() : start(0), end(0) {}
12. Interval(**int** s, **int** e) : start(s), end(e) {}
13. };
15. **class** Solution
16. {
17. **public**:
18. **bool** meetingRooms(vector<Interval>& intervals)
19. {
20. sort(intervals.begin(), intervals.end(),
21. [](**const** Interval& x, **const** Interval& y) {**return** x.start < y.start; });
22. **for** (**int** i = 1; i < intervals.size(); i++)
23. {
24. **if** (intervals[i].start < intervals[i - 1].end)
25. {
26. **return** **false**;
27. }
28. }
29. **return** **true**;
30. }
32. };
34. **int** main()
35. {
36. Solution a;
37. vector<Interval> intervals{ {0, 30},{5, 10},{15, 20} };
38. cout << a.meetingRooms(intervals);
39. system("pause");
40. **return** 1;
41. }

**973. K closest points to origin (E)**

**Problem:** We have a list of points on the plane. Find the K closest points to the origin (0, 0). (Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

**Example:** Input: points = [[3,3],[5,-1],[-2,4]], K = 2

Output: [[3,3],[-2,4]]

**Analysis:**

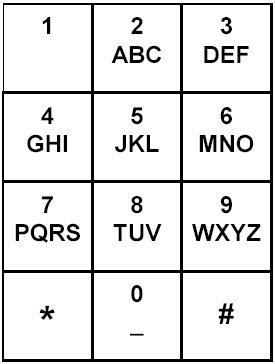
**Solution:**

**Section 18. Back Tracking**

**017. Letter combination of a phone number (M) [Back Tracking, string]**

**Problem:** Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

****

**Example:** Input: "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<string> findCombination(string number)
10. {
11. vector<string> res;
12. string dict[] = { " ", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz" };
13. string comb(number.size(), '\0');
14. dfs(number, 0, dict, comb, res);
15. **return** res;
16. }
18. **private**:
19. **void** dfs(string number, **int** index, string dict[], string& comb, vector<string>& res)
20. {
21. **if** (index == comb.size())
22. {
23. res.push\_back(comb);
24. **return**;
25. }
27. string letter = dict[number[index] - '0'];
28. **for** (**int** i = 0; i < letter.size(); i++)
29. {
30. comb[index] = letter[i];
31. dfs(number, index+1, dict, comb, res); // cannot be index++ or ++index!
32. }
33. }
34. };
36. **int** main()
37. {
38. Solution a;
39. vector<string> res = a.findCombination("23");
40. **for** (**int** i = 0; i < res.size(); i++)
41. {
42. cout << res[i] << " ";
43. }
44. cout << endl;
45. system("pause");
46. **return** 1;
47. }

**038. Sudoku solver (H) [Back Tracking, DFS]**

**Problem:** Write a program to solve a Sudoku puzzle by filling the empty cells. A sudoku solution must satisfy all of the following rules:

* Each of the digits 1-9 must occur exactly once in each row.
* Each of the digits 1-9 must occur exactly once in each column.
* Each of the the digits 1-9 must occur exactly once in each of the 9 3x3 sub-boxes of the grid.

**Note:**

* The given board contain only digits 1-9 and the character '.'.
* You may assume that the given Sudoku puzzle will have a single unique solution.
* The given board size is always 9x9.

**Example:**

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. **void** sudokuSolver(vector<vector<**char**>>& board)
9. {
10. **if** ((board.size() != 9) || (board[0].size() != 9)) **return**;
11. dfs(board, 0, 0);
12. }
14. **bool** dfs(vector<vector<**char**>>& board, **int** i, **int** j)
15. {
16. **if** (i == 9) **return** **true**;
17. **if** (j >= 9) **return** dfs(board, i + 1, 0);
18. **if** (board[i][j] == '.')
19. {
20. **for** (**int** k = 1; k <= 9; k++)
21. {
22. board[i][j] = (**char**)(k + '0');
23. **if** (isValid(board, i, j))
24. {
25. **if** (dfs(board, i, j + 1)) **return** **true**;
26. }
27. board[i][j] = '.';
28. }
29. }
30. **else**
31. {
32. **return** dfs(board, i, j + 1);
33. }
34. **return** **false**;
35. }
37. **bool** isValid(vector<vector<**char**>>& board, **int** i, **int** j)
38. {
39. **for** (**int** row = 0; row <= 8; row++)
40. {
41. **if** ((row != i) && (board[row][j] == board[i][j])) **return** **false**;
42. }
43. **for** (**int** col = 0; col <= 8; col++)
44. {
45. **if** ((col != j) && (board[i][col] == board[i][j])) **return** **false**;
46. }
47. **for** (**int** r = i / 3 \* 3; r < i / 3 \* 3 + 3; r++)
48. {
49. **for** (**int** c = j / 3 \* 3; c < j / 3 \* 3 + 3; c++)
50. {
51. **if** ((r != i || c != j) && (board[r][c] == board[i][j])) **return** **false**;
52. }
53. }
54. **return** **true**;
55. }
56. };
58. **int** main()
59. {
60. Solution a;
61. vector<vector<**char**>> board{ {'5', '3', '.', '.', '7', '.', '.', '.', '.'},
62. {'6', '.', '.', '1', '9', '5', '.', '.', '.'},
63. {'.', '9', '8', '.', '.', '.', '.', '6', '.'},
64. {'8', '.', '.', '.', '6', '.', '.', '.', '3'},
65. {'4', '.', '.', '8', '.', '3', '.', '.', '1'},
66. {'7', '.', '.', '.', '2', '.', '.', '.', '6'},
67. {'.', '6', '.', '.', '.', '.', '2', '8', '.'},
68. {'.', '.', '.', '4', '1', '9', '.', '.', '5'},
69. {'.', '.', '.', '.', '8', '.', '.', '7', '9'} };
70. a.sudokuSolver(board);
71. **for** (**int** i = 0; i < board.size(); i++)
72. {
73. **for** (**int** j = 0; j < board[i].size(); j++)
74. {
75. cout << board[i][j] << " ";
76. }
77. cout << endl;
78. }
79. system("pause");
80. **return** 1;
81. }

**039. Combination sum (M) [Back Tracking, DFS]**

**Problem:** Given a set of candidate numbers (candidates) (without duplicates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

The same repeated number may be chosen from candidates unlimited number of times.

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

**Example:** Input: candidates = [2,3,5], target = 8,

A solution set is: [ [2,2,2,2],

[2,3,3],

[3,5] ]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. vector<vector<**int**>> combinationSum(vector<**int**>& nums, **int** target)
9. {
10. vector<vector<**int**>> res;
11. vector<**int**> cur;
12. dfs(nums, target, 0, cur, res);
13. **return** res;
14. }
15. **private**:
16. **void** dfs(vector<**int**>& nums, **int** target, **int** start, vector<**int**>& cur, vector<vector<**int**>>& res)
17. {
18. **if** (target < 0) **return**;
19. **if** (target == 0)
20. {
21. res.push\_back(cur); **return**;
22. }
24. **for** (**int** i = start; i < nums.size(); i++)
25. {
26. cur.push\_back(nums[i]);
27. dfs(nums, target - nums[i], i, cur, res);
28. cur.pop\_back();
29. }
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. vector<**int**> nums{ 2, 3, 5 };
37. vector<vector<**int**>> res = a.combinationSum(nums, 8);
38. **for** (**int** i = 0; i < res.size(); i++)
39. {
40. **for** (**int** j = 0; j < res[i].size(); j++)
41. {
42. cout << res[i][j] << " ";
43. }
44. cout << endl;
45. }
46. system("pause");
47. **return** 1;
48. }

**040. Combination sum II (M) [Back Tracking, DFS]**

**Problem:** Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target.

Each number in candidates may only be used once in the combination.

**Note:**

* All numbers (including target) will be positive integers.
* The solution set must not contain duplicate combinations.

**Example:** Input: candidates = [10,1,2,7,6,1,5], target = 8,

A solution set is: [ [1, 7],

[1, 2, 5],

[2, 6],

[1, 1, 6] ]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> combinationSum(vector<**int**>& nums, **int** target)
10. {
11. vector<**int**> cur;
12. vector<vector<**int**>> res;
13. sort(nums.begin(), nums.end());
14. dfs(nums, target, 0, cur, res);
15. **return** res;
16. }
17. **private**:
18. **void** dfs(vector<**int**>&nums, **int** target, **int** start, vector<**int**>& cur, vector<vector<**int**>>& res)
19. {
20. **if** (target < 0) **return**;
21. **if** (target == 0)
22. {
23. res.push\_back(cur);
24. **return**;
25. }
27. **for** (**int** i = start; i < nums.size(); i++)
28. {
29. **if** ((i > start) && (nums[i] == nums[i - 1])) **continue**;
30. cur.push\_back(nums[i]);
31. dfs(nums, target - nums[i],  i + 1, cur, res);
32. cur.pop\_back();
33. }
34. }
35. };
37. **int** main()
38. {
39. vector<**int**> nums{ 10,1,2,7,6,1,5 };
40. **int** target = 8;
41. Solution a;
42. vector<vector<**int**>> res = a.combinationSum(nums, target);
43. **for** (**int** i = 0; i < res.size(); i++)
44. {
45. **for** (**int** j = 0; j < res[i].size(); j++)
46. {
47. cout << res[i][j] << " ";
48. }
49. cout << endl;
50. }
51. system("pause");
52. **return** 1;
53. }

**046. Permutations (M) [Back Tracking, DFS]**

**Problem:** Given a collection of distinct integers, return all possible permutations

**Example:** Input: [1,2,3]

Output: [ [1,2,3],

[1,3,2],

[2,1,3],

[2,3,1],

[3,1,2],

[3,2,1] ]

**Solution:**

1. #include <iostream>j
2. #include <vector>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. vector<vector<**int**>> permutations(vector<**int**>& nums)
9. {
10. vector<vector<**int**>> res;
11. vector<**int**> curr, visited(nums.size(), 0);
12. dfs(nums, 0, visited, curr, res);
13. **return** res;
14. }
15. **private**:
16. **void** dfs(vector<**int**>& nums, **int** level, vector<**int**>& visited, vector<**int**>& curr, vector<vector<**int**>>& res)
17. {
18. **if** (level == nums.size())
19. {
20. res.push\_back(curr);
21. **return**;
22. }
24. **for** (**int** i = 0; i < nums.size(); i++)
25. {
26. **if** (visited[i] == 1) **continue**;
27. curr.push\_back(nums[i]);
28. visited[i] = 1;
29. dfs(nums, level + 1, visited, curr, res);
30. curr.pop\_back();
31. visited[i] = 0;
32. }
33. }
34. };
36. **int** main()
37. {
38. vector<**int**> nums{ 1, 2, 3 };
39. Solution a;
40. vector<vector<**int**>> res = a.permutations(nums);
41. **for** (**int** i = 0; i < res.size(); i++)
42. {
43. **for** (**int** j = 0; j < res[i].size(); j++)
44. {
45. cout << res[i][j] << " ";
46. }
47. cout << endl;
48. }
49. system("pause");
50. **return** 1;
51. }

**047. Permutations II (M) [Back Tracking, DFS]**

**Problem:** Given a collection of numbers that might contain duplicates, return all possible unique permutations.

**Example:** Input: [1,1,2]

Output: [ [1,1,2],

[1,2,1],

[2,1,1] ]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> permutations(vector<**int**>& nums)
10. {
11. vector<vector<**int**>> res;
12. vector<**int**> curr, visited(nums.size(), 0);
13. sort(nums.begin(), nums.end());
14. dfs(nums, 0, visited, curr, res);
15. **return** res;
16. }
18. **private**:
19. **void** dfs(vector<**int**>& nums, **int** level, vector<**int**>& visited, vector<**int**>& curr, vector<vector<**int**>>& res)
20. {
21. **if** (level == nums.size())
22. {
23. res.push\_back(curr);
24. **return**;
25. }
26. **for** (**int** i = 0; i < nums.size(); i++)
27. {
28. **if** ((i > 0) && (nums[i] == nums[i - 1]) && (visited[i - i] == 0)) **continue**;
29. **if** (visited[i] == 1) **continue**;
30. curr.push\_back(nums[i]);
31. visited[i] = 1;
32. dfs(nums, level + 1, visited, curr, res);
33. curr.pop\_back();
34. visited[i] = 0;
35. }
36. }
37. };
39. **int** main()
40. {
41. vector<**int**> nums{ 1, 1, 2 };
42. Solution a;
43. vector<vector<**int**>> res = a.permutations(nums);
44. **for** (**int** i = 0; i < res.size(); i++)
45. {
46. **for** (**int** j = 0; j < res[i].size(); j++)
47. {
48. cout << res[i][j] << " ";
49. }
50. cout << endl;
51. }
52. system("pause");
53. **return** 1;
54. }

**077. Combination (M) [Back Tracking, DFS]**

**Problem:** Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

**Example:** Input: n = 4, k = 2

Output: [ [2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4] ]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. vector<vector<**int**>> combination(**int** n, **int** k)
9. {
10. vector<**int**> curr;
11. vector<vector<**int**>> res;
12. dfs(n, k, 1, curr, res);
13. **return** res;
14. }
15. **private**:
16. **void** dfs(**int** n, **int** k, **int** index, vector<**int**>& curr, vector<vector<**int**>>& res)
17. {
18. **if** (k == 0)
19. {
20. res.push\_back(curr);
21. **return**;
22. }
23. **for** (**int** i = index; i <= n; i++)
24. {
25. curr.push\_back(i);
26. dfs(n, k - 1, i + 1, curr, res);
27. curr.pop\_back();
28. }
29. }
30. };
32. **int** main()
33. {
34. Solution a;
35. vector<vector<**int**>> res = a.combination(4, 2);
36. **for** (**int** i = 0; i < res.size(); i++)
37. {
38. **for** (**int** j = 0; j < res[i].size(); j++)
39. {
40. cout << res[i][j] << " ";
41. }
42. cout << endl;
43. }
44. system("pause");
45. **return** 1;
46. }

**078. Subsets (M) [Back Tracking, DFS]**

**Problem:** Given a set of distinct integers, nums, return all possible subsets (the power set).

**Note:** The solution set must not contain duplicate subsets.

**Example:** Input: nums = [1,2,3]

Output: [ [3],

[1],

[2],

[1,2,3],

[1,3],

[2,3],

[1,2],

[] ]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> subsets(vector<**int**>& nums)
10. {
11. vector<vector<**int**>> res;
12. vector<**int**> curr;
13. sort(nums.begin(), nums.end());
14. **for** (**int** i = 0; i <= nums.size(); i++)
15. {
16. dfs(nums, 0, i, curr, res);
17. }
18. **return** res;
19. }
20. **private**:
21. **void** dfs(vector<**int**>&nums, **int** index, **int** target, vector<**int**>& curr, vector<vector<**int**>>& res)
22. {
23. **if** (target == 0)
24. {
25. res.push\_back(curr);
26. **return**;
27. }
28. **for** (**int** i = index; i < nums.size(); i++)
29. {
30. curr.push\_back(nums[i]);
31. dfs(nums, i + 1, target - 1, curr, res);
32. curr.pop\_back();
33. }
34. }
35. };
37. **int** main()
38. {
39. vector<**int**> nums{ 1, 2, 3 };
40. Solution a;
41. vector<vector<**int**>> res = a.subsets(nums);
42. **for** (**int** i = 0; i < res.size(); i++)
43. {
44. **for** (**int** j = 0; j < res[i].size(); j++)
45. {
46. cout << res[i][j] << " ";
47. }
48. cout << endl;
49. }
50. system("pause");
51. **return** 1;
52. }

**079. Word search (M) [Back Tracking, DFS]**

**Problem:** Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

**Example:** board = [ ['A','B','C','E'],

['S','F','C','S'],

['A','D','E','E'] ]

Given word = "ABCCED", return true.

Given word = "SEE", return true.

Given word = "ABCB", return false.

**Solution:**

1. #include <iostream>
2. #include <string>
3. #include <vector>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. **bool** wordExist(vector<vector<**char**>>& board, string word)
10. {
11. **if** ((board.empty()) || (board[0].empty())) **return** **false**;
12. **int** m = board.size(), n = board[0].size();
13. vector<vector<**bool**>> visited(m, vector<**bool**>(n, **false**));
14. **for** (**int** i = 0; i < m; i++)
15. {
16. **for** (**int** j = 0; j < n; j++)
17. {
18. **if** (wordSearch(board, word, 0, i, j, visited)) **return** **true**;
19. }
20. }
21. **return** **false**;
22. }
23. **private**:
24. **bool** wordSearch(vector<vector<**char**>>& board, string word, **int** index, **int** i, **int** j, vector<vector<**bool**>>& visited)
25. {
26. **if** (index == word.size()) **return** **true**;
27. **int** m = board.size(), n = board[0].size();
28. **if** ((i < 0) || (i >= m) || (j < 0) || (j >= n) || (visited[i][j]) || (board[i][j] != word[index])) **return** **false**;
30. visited[i][j] = **true**;
31. **bool** res = wordSearch(board, word, index + 1, i - 1, j, visited) ||
32. wordSearch(board, word, index + 1, i, j - 1, visited) ||
33. wordSearch(board, word, index + 1, i + 1, j, visited) ||
34. wordSearch(board, word, index + 1, i, j + 1, visited);
35. visited[i][j] = **false**;
36. **return** res;
37. }
38. };
40. **int** main()
41. {
42. vector<vector<**char**>> board{ {'A','B','C','E'}, {'S','F','C','S'}, {'A','D','E','E'} };
43. string word = "ABCCED";
44. Solution a;
45. cout << a.wordExist(board, word) << endl;
46. system("pause");
47. **return** 1;
48. }

**090. Subsets II (M) [Back Tracking, DFS]**

**Problem:** Given a collection of integers that might contain duplicates, nums, return all possible subsets (the power set).

**Note:** The solution set must not contain duplicate subsets.

**Example:** Input: [1,2,2]

Output: [ [2],

[1],

[1,2,2],

[2,2],

[1,2],

[] ]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<vector<**int**>> subsets(vector<**int**>& nums)
10. {
11. vector<**int**> curr;
12. vector<vector<**int**>> res;
13. sort(nums.begin(), nums.end());
14. dfs(nums, 0, curr, res);
15. **return** res;
16. }
17. **private**:
18. **void** dfs(vector<**int**>& nums, **int** index, vector<**int**>& curr, vector<vector<**int**>>& res)
19. {
20. res.push\_back(curr);
21. **for** (**int** i = index; i < nums.size(); i++)
22. {
23. **if** ((i > index) && (nums[i] == nums[i - 1])) **continue**;
24. curr.push\_back(nums[i]);
25. dfs(nums, i + 1, curr, res);
26. curr.pop\_back();
27. }
28. }
29. };
31. **int** main()
32. {
33. vector<**int**> nums{ 1, 2, 2 };
34. Solution a;
35. vector<vector<**int**>> res = a.subsets(nums);
36. **for** (**int** i = 0; i < res.size(); i++)
37. {
38. **for** (**int** j = 0; j < res[i].size(); j++)
39. {
40. cout << res[i][j] << " ";
41. }
42. cout << endl;
43. }
44. system("pause");
45. **return** 1;
47. }

**212. Word Search II (H) [Back Tracking, DFS]**

**Problem:** Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

**Example:** Input: board = [ ['o','a','a','n'],

['e','t','a','e'],

['i','h','k','r'],

['i','f','l','v'] ]

words = ["oath","pea","eat","rain"]

Output: ["eat","oath"]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<string> wordSearch(vector<vector<**char**>>& board, vector<string>& words)
10. {
11. vector<string> res;
12. **if** ((board.empty()) || (board[0].empty())) **return** res;
13. **int** m = board.size(), n = board[0].size();
15. **for** (**int** x = 0; x < words.size(); x++)
16. {
17. **bool** tmp = **false**;
18. vector<vector<**bool**>> visited(m, vector<**bool**>(n, **false**));
19. **for** (**int** i = 0; i < m; i++)
20. {
21. **for** (**int** j = 0; j < n; j++)
22. {
23. **if** (wordExist(board, words[x], 0, i, j, visited)) tmp = **true**;
24. }
25. }
26. **if** (tmp) res.push\_back(words[x]);
27. }
28. **return** res;
29. }
30. **private**:
31. **bool** wordExist(vector<vector<**char**>>& board, string word, **int** index, **int** i, **int** j, vector<vector<**bool**>>& visited)
32. {
33. **if** (index == word.size()) **return** **true**;
34. **int** m = board.size(), n = board[0].size();
35. **if** ((i < 0) || (i >= m) || (j < 0) || (j >= n) || (visited[i][j]) || (board[i][j] != word[index])) **return** **false**;
36. visited[i][j] = **true**;
37. **bool** res = wordExist(board, word, index + 1, i - 1, j, visited) ||
38. wordExist(board, word, index + 1, i, j - 1, visited) ||
39. wordExist(board, word, index + 1, i + 1, j, visited) ||
40. wordExist(board, word, index + 1, i, j + 1, visited);
41. visited[i][j] = **false**;
42. **return** res;
43. }
44. };
46. **int** main()
47. {
48. vector<vector<**char**>> board = { {'o','a','a','n'}, {'e','t','a','e'}, {'i','h','k','r'}, {'i','f','l','v'} };
49. vector<string> words{ "oath","pea","eat","rain" };
50. Solution a;
51. vector<string> res = a.wordSearch(board, words);
52. **for** (**int** i = 0; i < res.size(); i++)
53. {
54. cout << res[i] << endl;
55. }
56. system("pause");
57. **return** 1;
59. }

**216. Combination sum III (M) [Back Tracking, DFS]**

**Problem:** Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

**Note:**

* All numbers will be positive integers.
* The solution set must not contain duplicate combinations.

**Example:** Input: k = 3, n = 7

Output: [[1,2,4]]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **class** Solution
6. {
7. **public**:
8. vector<vector<**int**>>combinationSum(**int** k, **int** target)
9. {
10. vector<**int**> curr;
11. vector<vector<**int**>> res;
12. dfs(k, target, 1, curr, res);
13. **return** res;
14. }
15. **private**:
16. **void** dfs(**int** k, **int** target, **int** start, vector<**int**>& curr, vector<vector<**int**>>& res)
17. {
18. **if** (target < 0) **return**;
19. **if** ((curr.size() == k) && (target == 0))
20. {
21. res.push\_back(curr);
22. **return**;
23. }
24. **for** (**int** i = start; i <= 9; i++)
25. {
26. curr.push\_back(i);
27. dfs(k, target - i, i + 1, curr, res);
28. curr.pop\_back();
29. }
30. }
31. };
33. **int** main()
34. {
35. Solution a;
36. vector<vector<**int**>> res = a.combinationSum(3, 9);
37. **for** (**int** i = 0; i < res.size(); i++)
38. {
39. **for** (**int** j = 0; j < res[i].size(); j++)
40. {
41. cout << res[i][j] << " ";
42. }
43. cout << endl;
44. }
45. system("pause");
46. **return** 1;
47. }

**784. Letter case permutation (E) [Back Tracking, DFS]**

**Problem:** Given a string S, we can transform every letter individually to be lowercase or uppercase to create another string. Return a list of all possible strings we could create.

**Note:**

* S will be a string with length between 1 and 12.
* S will consist only of letters or digits.

**Examples:** Input: S = "a1b2"

Output: ["a1b2", "a1B2", "A1b2", "A1B2"]

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <string>
4. **using** **namespace** std;
6. **class** Solution
7. {
8. **public**:
9. vector<string> letterPermutation(string input)
10. {
11. vector<string> res;
12. dfs(input, 0, res);
13. **return** res;
14. }
15. **private**:
16. **void** dfs(string& input, **int** pos, vector<string>& res)
17. {
18. **if** (pos == input.size())
19. {
20. res.push\_back(input);
21. **return**;
22. }
24. dfs(input, pos + 1, res);
25. **if** (input[pos] > '9')
26. {
27. input[pos] ^= 32;
28. dfs(input, pos + 1, res);
29. }
30. }
31. };
33. **int** main()
34. {
35. string input = "a1b2";
36. Solution a;
37. vector<string> res = a.letterPermutation(input);
38. **for** (**int** i = 0; i < res.size(); i++)
39. {
40. cout << res[i] << endl;
41. }
42. system("pause");
43. **return** 1;
44. }

**Section 19. Dynamic Programming**

**062. Unique path (M) [dynamic programming, array]**

**Problem:** A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?



**Note:** m and n will be at most 100.

**Example:** Input: m = 7, n = 3

Output: 28

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** uniquePaths(**int** m, **int** n)
6. {
7. vector<vector<**int**>> path(m+1, vector<**int**>(n+1, 0));
8. **if** ((m < 1) || (n < 1)) **return** 0;
9. **for** (**int** i = 1; i <= m; i++) path[i][1] = 1;
10. **for** (**int** j = 1; j <= n; j++) path[1][j] = 1;
12. path[2][1] = 1;
13. path[1][2] = 1;
15. **for** (**int** i = 2; i <= m; i++)
16. {
17. **for** (**int** j = 2; j <= n; j++)
18. {
19. path[i][j] = path[i - 1][j] + path[i][j - 1];
20. }
21. }
22. **return** path[m][n];
23. }
25. **int** main()
26. {
27. cout << uniquePaths(7, 3) << endl;
28. system("pause");
29. **return** 1;
30. }

**063. Unique path II (M) [dynamic programming, array]**

**Problem:** A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

**Example:** Input: [ [0,0,0],

[0,1,0],

[0,0,0] ]

Output: 2

**Note:** m and n will be at most 100

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** uniquePaths(vector<vector<**int**>>& obstacleGrid)
6. {
7. **if** (obstacleGrid.empty() || obstacleGrid[0].empty() || obstacleGrid[0][0] == 1) **return** 0;
8. **int** m = obstacleGrid.size(), n = obstacleGrid[0].size();
9. vector<vector<**long**>> path(m + 1, vector<**long**>(n + 1, 0));
10. path[0][1] = 1;
11. **for** (**int** i = 1; i <= m; i++)
12. {
13. **for** (**int** j = 1; j <= n; j++)
14. {
15. **if** (obstacleGrid[i-1][j-1] == 1) **continue**;
16. path[i][j] = path[i][j - 1] + path[i - 1][j];
17. }
18. }
19. **return** path[m][n];
20. }
22. **int** main()
23. {
24. vector<vector<**int**>> grid{ {0, 0, 0}, {0, 1, 0}, {0, 0, 0} };
25. cout << uniquePaths(grid) << endl;
26. system("pause");
27. **return** 1;
28. }

**064. Minimum path sum (M) [dynamic programming, array]**

**Problem:** Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

**Example:** Input: [ [1,3,1],

[1,5,1],

[4,2,1] ]

Output: 7

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **int** minPathSum(vector<vector<**int**>>& x)
7. {
8. **if** ((x.empty()) || (x[0].empty())) **return** 0;
9. **int** m = x.size(), n = x[0].size();
10. vector<vector<**int**>> res(m, vector<**int**>(n, 0));
11. res[0][0] = x[0][0];
13. **for** (**int** i = 0; i < m; i++)
14. {
15. **for** (**int** j = 0; j < n; j++)
16. {
17. **if** ((i == 0) && (j > 0)) res[i][j] = res[i][j - 1] + x[i][j];
18. **else** **if** ((j == 0) && (i > 0)) res[i][j] = res[i - 1][j] + x[i][j];
19. **else** **if** ((i > 0) && (j > 0)) res[i][j] = min(res[i - 1][j] + x[i][j], res[i][j - 1] + x[i][j]);
20. }
21. }
22. **return** res[m-1][n-1];
23. }
25. **int** main()
26. {
27. vector<vector<**int**>> x{ {1, 3, 1}, {1, 5, 1}, {4, 2, 1} };
28. cout << minPathSum(x) << endl;
29. system("pause");
30. **return** 1;
31. }

**070. Climbing stairs (E) [dynamic programming]**

**Problem:** You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

**Note:** Given n will be a positive integer.

**Example:** Input: 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. // dp
6. **class** Solution
7. {
8. **public**:
9. **int** climbingStairs(**int** n)
10. {
11. **if** (n <= 1) **return** 1;
12. vector<**int**> dp(n, 0);
13. dp[0] = 1;
14. dp[1] = 2;
15. **for** (**int** i = 2; i < n; i++)
16. {
17. dp[i] = dp[i - 1] + dp[i - 2];
18. }
19. **return** dp[n - 1];
20. }
21. };
23. // back tracking
24. **class** Solution1
25. {
26. **public**:
27. **int** climbingStairs(**int** n)
28. {
29. vector<**int**> stairs(n + 1, 0);
30. **return** helper(stairs, n);
31. }
32. **private**:
33. **int** helper(vector<**int**>& m, **int** n)
34. {
35. **if** (n <= 1) **return** 1;
36. **if** (m[n] > 0) **return** m[n];
37. **return** m[n] = helper(m, n - 1) + helper(m, n - 2);
38. }
39. };
41. **int** main()
42. {
43. Solution1 a;
44. cout << a.climbingStairs(400) << endl;
45. system("pause");
46. **return** 1;
47. }

**120. Triangle (M) [dynamic programming. array]**

**Problem:** Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

**Example:** input: [ [2],

[3,4],

[6,5,7],

[4,1,8,3] ]

output: 11

**Note:** Bonus point if you are able to do this using only O(n) extra space, where n is the total number of rows in the triangle.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **int** triangle(vector<vector<**int**>>& triangle)
7. {
8. **if** ((triangle.empty()) || (triangle[0].empty())) **return** 0;
9. vector<**int**> dp = triangle.back();

12. **for** (**int** i = triangle.size()-2; i >= 0; i--)
13. {
14. **for** (**int** j = 0; j <= i; j++)
15. {
16. dp[j] = min(dp[j], dp[j + 1]) + triangle[i][j];
17. }
18. }
19. **return** dp[0];
20. }
22. **int** main()
23. {
24. vector<vector<**int**>> m{ {-1}, {2, 3}, {1, -1, -3}, {5, 3, -1, 2} };
25. cout << triangle(m) << endl;
26. system("pause");
27. **return** 1;
28. }

**174. Dungeon game (H) [dynamic programming. array]**

**Problem:** The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers).

In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.

**Example:**

|  |  |  |
| --- | --- | --- |
| -2 (K) | -3 | 3 |
| -5 | -10 | 1 |
| 10 | 30 | -5 (P) |

For example, given the dungeon below, the initial health of the knight must be at least 7 if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN.

**Note:**

* The knight's health has no upper bound.
* Any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

**Solution:**

**509. Fibonacci number (E) [dynamic programming. array]**

**Problem:** The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1

F(N) = F(N - 1) + F(N - 2), for N > 1.

Given N, calculate F(N).

**Example:** Input: 4

Output: 3

**Note:** 0 ≤ N ≤ 30.

**Solution:**

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
5. **int** fibonacci(**int** n)
6. {
7. vector<**int**> f(n+1, 0);
8. **if** (n < 1) **return** 0;
9. f[0] = 0;
10. f[1] = 1;
11. **for** (**int** i = 2; i <= n; i++)
12. {
13. f[i] = f[i - 1] + f[i - 2];
14. }
15. **return** f[n];
16. }
18. **int** fibonacci1(**int** n)
19. {
20. **if** (n < 1) **return** 0;
21. **int** a = 0, b = 1;
22. **int** res = 1;
23. **for** (**int** i = 2; i <= n; i++)
24. {
25. res = a + b;
26. a = b;
27. b = res;
28. }
29. **return** res;

32. }
34. **int** main()
35. {
36. cout << fibonacci1(1) << endl;
37. system("pause");
38. **return** 1;
39. }

**746. Min cost climbing staris (E) [dynamic programming. array]**

**Problem:** On a staircase, the i-th step has some non-negative cost cost[i] assigned (0 indexed).

Once you pay the cost, you can either climb one or two steps. You need to find minimum cost to reach the top of the floor, and you can either start from the step with index 0, or the step with index 1.

**Example:** Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]

Output: 6

Explanation: Cheapest is start on cost[0], and only step on 1s, skipping cost[3].

**Note:**

* cost will have a length in the range [2, 1000].
* every cost[i] will be an integer in the range [0, 999].

**Solution:**

1. #include <iostream>
2. #include <vector>
3. #include <algorithm>
4. **using** **namespace** std;
6. **int** minCost(vector<**int**>& cost)
7. {
8. vector<**int**> dp(cost.size() + 1, 0);
9. **for** (**int** i = 2; i <= cost.size(); i++)
10. {
11. dp[i] = min((dp[i - 2] + cost[i - 2]), (dp[i - 1] + cost[i - 1]));
12. }
13. **return** dp[cost.size()];
14. }
16. **int** main()
17. {
18. vector<**int**> cost{ 1, 100, 1, 1, 1, 100, 1, 1, 100, 1 };
19. cout << minCost(cost) << endl;
20. system("pause");
21. **return** 1;
22. }

**Section 20. Design**

**146. LRU cache (H) [double linked list, hash map]**

**Problem:** Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

**Follow up:** Could you do both operations in O(1) time complexity?

**Example:**

LRUCache cache = new LRUCache( 2 /\* capacity \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // returns 1

cache.put(3, 3); // evicts key 2

cache.get(2); // returns -1 (not found)

cache.put(4, 4); // evicts key 1

cache.get(1); // returns -1 (not found)

cache.get(3); // returns 3

cache.get(4); // returns 4

**Solution:**

1. #include <iostream>
2. #include <unordered\_map>
3. **using** **namespace** std;
5. **struct** LRUCacheNode
6. {
7. **int** key;
8. **int** val;
9. LRUCacheNode\* prev;
10. LRUCacheNode\* next;
12. LRUCacheNode(): key(0), val(0), prev(NULL), next(NULL){}
13. };
15. **class** LRUCache
16. {
17. **private**:
18. unordered\_map<**int**, LRUCacheNode\*> m;
19. LRUCacheNode\* head; //
20. LRUCacheNode\* tail; // LRU node's next
21. **int** count;
22. **int** capacity;
23. **public**:
24. LRUCache(**int** size);
25. ~LRUCache();
26. **int** get(**int** key);
27. **void** set(**int** key, **int** val);
28. **private**:
29. **void** detachNode(LRUCacheNode\* node);
30. **void** insertToFront(LRUCacheNode\* node);
31. **void** deleteNode();                      // delete node at end (LRU)
32. };
34. LRUCache::LRUCache(**int** size)
35. {
36. capacity = size;
37. count = 0;
38. head = **new** LRUCacheNode;
39. tail = **new** LRUCacheNode;
40. head->prev = NULL;
41. head->next = tail;
42. tail->prev = head;
43. tail->next = NULL;
44. }
46. LRUCache::~LRUCache()
47. {
48. **delete** head;
49. **delete** tail;
50. }
52. **void** LRUCache::detachNode(LRUCacheNode\* node)
53. {
54. node->prev->next = node->next;
55. node->next->prev = node->prev;
56. }
58. **void** LRUCache::insertToFront(LRUCacheNode\* node)
59. {
60. node->next = head->next;
61. node->prev = head;
62. head->next = node;
63. node->next->prev = node; // needed!
64. }
66. **void** LRUCache::deleteNode()
67. {
68. LRUCacheNode\* node = tail->prev;
69. detachNode(node);
70. m.erase(node->key);
71. count--;
72. }
74. **int** LRUCache::get(**int** key)
75. {
76. **if** (m.find(key) == m.end()) **return** -1;
77. **else**
78. {
79. detachNode(m[key]);
80. insertToFront(m[key]);
81. **return** m[key]->val;
82. }
83. }
85. **void** LRUCache::set(**int** key, **int** val)
86. {
87. **if** (m.find(key) == m.end())
88. {
89. **if** (count == capacity)
90. {
91. deleteNode();
92. }
93. LRUCacheNode\* node = **new** LRUCacheNode;
94. node->key = key;
95. node->val = val;
96. m[key] = node;
97. insertToFront(node);
98. ++count;
99. }
100. **else**
101. {
102. m[key]->val = val;
103. detachNode(m[key]);
104. insertToFront(m[key]);
105. }
106. }
108. **int** main()
109. {
110. LRUCache a(2);
111. a.set(1, 1);
112. a.set(2, 2);
113. cout << a.get(1) << endl;
114. a.set(3, 3);
115. cout << a.get(2) << endl;
116. cout << a.get(3) << endl;
117. a.set(4, 4);
118. cout << a.get(1) << endl;
119. cout << a.get(3) << endl;
120. cout << a.get(4) << endl;
121. system("pause");
122. **return** 1;
124. }

**355. Design twitter (M) [hash map, unordered set, map]**

**Problem:** Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets in the user's news feed. Your design should support the following methods:

postTweet(userId, tweetId): Compose a new tweet.

getNewsFeed(userId): Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent.

follow(followerId, followeeId): Follower follows a followee.

unfollow(followerId, followeeId): Follower unfollows a followee.

**Example:**

Twitter twitter = new Twitter();

// User 1 posts a new tweet (id = 5).

twitter.postTweet(1, 5);

// User 1's news feed should return a list with 1 tweet id -> [5].

twitter.getNewsFeed(1);

// User 1 follows user 2.

twitter.follow(1, 2);

// User 2 posts a new tweet (id = 6).

twitter.postTweet(2, 6);

// User 1's news feed should return a list with 2 tweet ids -> [6, 5].

// Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5.

twitter.getNewsFeed(1);

// User 1 unfollows user 2.

twitter.unfollow(1, 2);

// User 1's news feed should return a list with 1 tweet id -> [5],

// since user 1 is no longer following user 2.

twitter.getNewsFeed(1);

**Solution:**

1. #include <iostream>
2. #include <unordered\_map>
3. #include <map>
4. #include <unordered\_set>
5. **using** **namespace** std;
7. **class** Twitter
8. {
9. **public**:
10. Twitter()
11. {
12. time = 0;
13. }
15. **void** postTweet(**int** userId, **int** tweetId)
16. {
17. follow(userId, userId);
18. tweets[userId].insert({ ++time, tweetId });
19. }
21. vector<**int**> getNewsFeed(**int** userId)
22. {
23. vector<**int**> res;
24. map<**int**, **int**> top10;
25. **for** (auto id : friends[userId])
26. {
27. **for** (auto t : tweets[id])
28. {
29. top10.insert({ t.first, t.second });
30. **if** (top10.size() > 10)
31. {
32. top10.erase(top10.begin());
33. }
34. }
35. }
36. **for** (auto t : top10)
37. {
38. res.insert(res.begin(), t.second);
39. }
40. **return** res;
41. }
43. **void** follow(**int** followerId, **int** followeeId)
44. {
45. friends[followerId].insert(followeeId);
46. }
48. **void** unfollow(**int** followerId, **int** followeeId)
49. {
50. **if** (followerId != followeeId)
51. {
52. friends[followerId].erase(followeeId);
53. }
54. }
56. **private**:
57. **int** time;
58. unordered\_map<**int**, unordered\_set<**int**>> friends; // userId -> frends unordered set
59. unordered\_map<**int**, map<**int**, **int**>> tweets;       // userId -> <time, tweets> map
60. };
62. **int** main()
63. {
64. Twitter t;
65. t.postTweet(1, 5); // user 1 posts a new tweet (id = 5)
67. // user 1's news feed should return a list with 1 twwed id -> [5]
68. vector<**int**> res1 = t.getNewsFeed(1);
69. **for** (auto r : res1) cout << r << " ";
71. t.follow(1, 2);    // user 1 follows user 2
72. t.postTweet(2, 6); // user 2 posts a new tweet (id = 6)
74. // should return [6, 5];
75. vector<**int**> res2 = t.getNewsFeed(1);
76. cout << endl;
77. **for** (auto r : res2) cout << r << " ";
79. t.unfollow(1, 2);   // user 1 unfollows user 2
81. // should return [5];
82. vector<**int**> res3 = t.getNewsFeed(1);
83. cout << endl;
84. **for** (auto r : res3) cout << r << " ";
86. system("pause");
87. **return** 1;
88. }

**460. LFU cache (H) [hash map, list iterator]**

**Problem:** Design and implement a data structure for Least Frequently Used (LFU) cache. It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least frequently used item before inserting a new item. For the purpose of this problem, when there is a tie (i.e., two or more keys that have the same frequency), the least recently used key would be evicted.

**Follow up:** Could you do both operations in O(1) time complexity?

**Example:**

LFUCache cache = new LFUCache( 2 /\* capacity \*/ );

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // returns 1

cache.put(3, 3); // evicts key 2

cache.get(2); // returns -1 (not found)

cache.get(3); // returns 3.

cache.put(4, 4); // evicts key 1.

cache.get(1); // returns -1 (not found)

cache.get(3); // returns 3

cache.get(4); // returns 4

**Solution:**

1. #include <iostream>
2. #include <list>
3. #include <unordered\_map>
4. **using** **namespace** std;
6. **class** LFUCache
7. {
8. **private**:
9. unordered\_map<**int**, pair<**int**, **int**>> m;           // key -> {val, freq}
10. unordered\_map<**int**, list<**int**>::iterator> mIter;  // key -> key list iterator
11. unordered\_map<**int**, list<**int**>> fm;               // freq -> key list
12. **int** capacity, minFreq;
13. **public**:
15. LFUCache(**int** size)
16. {
17. capacity = size;
18. }
20. **int** get(**int** key)
21. {
22. // 1. determine if the keyue is in m
23. //if (m.count(key) == 0) return -1;
24. **if** (m.find(key) == m.end()) **return** -1;
25. **else**
26. {
27. // 2. erase corresponding data in fm
28. fm[m[key].second].erase(mIter[key]);
30. // 3. update m
31. ++m[key].second;
33. // 4. add data to fm
34. fm[m[key].second].push\_back(key);
36. // 5. update data in mIter
37. mIter[key] = --fm[m[key].second].end();
39. // 6. determine minFreq
40. **if** (fm[minFreq].size() == 0) ++minFreq;
42. // 7. return value
43. **return** m[key].first;
44. }
45. }
47. **void** put(**int** key, **int** val)
48. {
49. **if** (capacity <= 0) **return**;
50. // 1. if key not in m, update val and return
51. **if** (get(key) != -1)
52. {
53. m[key].first = val;
54. **return**;
55. }
56. **else**
57. {
58. // 2. if m.size() >= cap
59. **if** (m.size() >= capacity)
60. {
61. // 2a). erase m's minFreq corresponding list first element
62. m.erase(fm[minFreq].front());
63. // 2b). erase mIter corresonding data
64. mIter.erase(fm[minFreq].front());
65. // 2c) erase fm list first element
66. fm[minFreq].pop\_front();
67. }
69. // 3. in m, add key -> {val, freq}
70. m[key] = { val, 1 };
72. // 4. in fm, add 1 correspoinding list val at end
73. fm[1].push\_back(key);
75. // 5. in mIter, save fm[1] list position
76. mIter[key] = --fm[1].end();
78. // 6. reset minFreq to 1
79. minFreq = 1;
80. }
81. }
82. };
84. **int** main()
85. {
86. LFUCache a(2);
87. a.put(1, 1);
88. a.put(2, 2);
89. cout<< a.get(1) << endl;    // return 1
90. a.put(3, 3); // evicts key 2
91. cout << a.get(2) << endl;   // return -1
92. cout << a.get(3) << endl;   // return 3
93. a.put(4, 4); // evicts key 1
94. cout << a.get(1) <<endl;    // return -1
95. cout << a.get(3) << endl;   // return 3
96. cout << a.get(4) << endl;   // return 4
97. system("pause");
98. **return** 1;
99. }