**53. Maximum Subarray**

class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        int sum = 0;

        int maximum = INT\_MIN;

         for(int i=0;i<nums.size();i++)

         {

             sum+=nums[i];

             if(sum > maximum){

                 maximum = sum;

             }

             if(sum < 0)

             {

                 sum = 0;

             }

         }

         return maximum;

    }

};

**75. Sort Colors**

class Solution {

public:

    void sortColors(vector<int>& nums) {

    //   int f =0;

    //   int s = 0;

    //   int t = 0;

    //   for(int i=0;i<nums.size();i++)

    //   {

    //       if(nums[i] == 0)

    //       {

    //           f++;

    //       }

    //       else if(nums[i] == 1)

    //       {

    //           s++;

    //       }

    //       else

    //       {

    //           t++;

    //       }

    //   }

    //   for(int i=0;i<f;i++)

    //   {

    //       nums[i] = 0;

    //   }

    //   for(int i=f;i<f+s;i++)

    //   {

    //       nums[i] = 1;

    //   }

    //   for(int i=f+s;i<f+s+t;i++)

    //   {

    //       nums[i] = 2;

    //   }

    int l =0;

    int m = 0;

    int h = nums.size()-1;

     while(m <= h)

     {

       int x = nums[m];

       if(x == 0)

       {

           swap(nums[l],nums[m]);

           l++;

           m++;

       }

       else if(x == 1)

       {

           m++;

       }

       else

       {

           swap(nums[m],nums[h]);

           h--;

       }

     }

    }

};

**121. Best Time to Buy and Sell Stock**

class Solution {

public:

    int maxProfit(vector<int>& prices) {

      int mini = prices[0];

      int maxProfit = 0;

      int n = prices.size();

      for(int i=0;i<n;i++)

      {

          int cost = prices[i] - mini;

          maxProfit = max(maxProfit,cost);

          mini = min(mini,prices[i]);

      }

      return maxProfit;

    }

};

**48. Rotate Image**

class Solution {

public:

    void rotate(vector<vector<int>>& matrix) {

       int n = matrix.size();

       for(int i=0;i<n-1;i++)

       {

           for(int j=i+1;j<n;j++)

           {

               swap(matrix[i][j],matrix[j][i]);

           }

       }

       for(int i=0;i<n;i++)

       {

           reverse(matrix[i].begin(),matrix[i].end());

       }

    }

};

**56. Merge Intervals**

class Solution {

public:

    vector<vector<int>> merge(vector<vector<int>>& intervals) {

      vector<vector<int>> mergedIntervals;

      if(intervals.size() == 0)

      {

          return mergedIntervals;

      }

      sort(intervals.begin(),intervals.end());

      vector<int> tempInterval = intervals[0];

      for(auto interval : intervals)

      {

          if(interval[0] <= tempInterval[1])//merging condition

          {

              tempInterval[1] = max(interval[1],tempInterval[1]);

          }

          else//not merging condition

          {

              mergedIntervals.push\_back(tempInterval);

              tempInterval = interval;

          }

      }

      mergedIntervals.push\_back(tempInterval);

      return mergedIntervals;

    }

};

**88. Merge Sorted Array**

class Solution {

public:

    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

       int i = m-1;

       int j = n-1;

       int k = m+n-1;

       while(j>=0)

       {

           if(i>=0 && nums1[i] > nums2[j])

           {

               nums1[k] = nums1[i];

               k--;

               i--;

           }

           else

           {

               nums1[k] = nums2[j];

               k--;

               j--;

           }

       }

    }

};

**287. Find the Duplicate Number**

class Solution {

public:

    int findDuplicate(vector<int>& nums) {

       int slow = 0;

       int fast = 0;

       do

       {

           slow = nums[slow];

           fast = nums[nums[fast]];

       }while(slow!=fast);

       slow = 0;

       while(slow != fast){

           slow = nums[slow];

           fast = nums[fast];

       }

       return slow;

    }

};

**Repeat and Missing Number Array**

vector<int> Solution::repeatedNumber(const vector<int> &A) {

    int n = A.size();

    int temp[n];

    for(int i=0;i<n;i++)

    {

        temp[i] = 0;

    }

    int repeat = 0;

    int miss = 0;

    for(int i=0;i<n;i++)

    {

        if(temp[A[i]-1] == 0)

        {

           temp[A[i]-1] = 1;

        }

        if(temp[A[i]-1] == 1){

            repeat = A[i];

        }

    }

    for(int i=0;i<n;i++)

    {

        if(temp[i] == 0)

        {

            miss = i+1;

        }

    }

    vector<int> ans;

    ans.push\_back(repeat);

    ans.push\_back(miss);

    return ans;

}

**Count Inversions**

#include <bits/stdc++.h>

long long getInversions(long long \*arr, int n){

    // Write your code here.

    long long temp[n];

    long long ans = merge\_sort(arr,temp,0,n-1);

    return ans;

}

long long merge\_sort(long long \*arr,long long \*temp,int left,int right){

    int mid;

    int inv\_count = 0;

    if(right > left){

        mid = left + (right - left)/2;

        inv\_count += merge\_sort(arr,temp,left,mid);

        inv\_count += merge\_sort(arr,temp,mid+1,right);

        inv\_count += merge(arr,temp,left,mid+1,right);

    }

    return inv\_count;

}

long long merge(long long \*arr,long long \*temp,int left,int mid,int right){

   int inv\_count = 0;

   int i = left;

   int j = mid;

   int k = left;

   while((i <= mid -1) && (j <=right))

   {

       if(arr[i] <= arr[j])

       {

           temp[k++] = arr[i++];

       }

       else

       {

           temp[k++] = arr[j++];

           inv\_count += (mid-i)

       }

   }

   while(i <=mid-1){

       temp[k++] = arr[i++];

   }

   while(j <= right){

       temp[k++] = arr[j++];

   }

   for(int i=left;i<=right;i++)

   {

       arr[i] = temp[i];

   }

   return inv\_count;

}

**74. Search a 2D Matrix**

class Solution {

public:

    bool searchMatrix(vector<vector<int>>& matrix, int target) {

     int r = 0;

        int c = matrix[0].size() - 1;

        while (r < matrix.size() && c >= 0)

        {

            if (matrix[r][c] == target)

            {

                return true;

            }

            else

            {

                if (matrix[r][c] > target)

                {

                    c--;

                }

                else

                {

                    r++;

                }

            }

        }

        return false;

    }

};

**50. Pow(x, n)**

class Solution {

public:

    double myPow(double x, int n) {

    double ans = 1.0;

    long long N = n;

    if (N < 0) N = -1 \* N;

    while (N) {

      if (N % 2) {

       ans = ans \* x;

       N = N - 1;

      } else {

        x = x \* x;

        N = N / 2;

      }

    }

     if (n < 0) ans = (double)(1.0) / (double)(ans);

     return ans;

    }

};

**169. Majority Element**

**Approach 1:**

class Solution {

public:

    int majorityElement(vector<int>& nums) {

      map<int,int> mp;

      for(int i=0;i<nums.size();i++)

      {

          mp[nums[i]]++;

      }

      for(auto it : mp){

          if(it.second > (nums.size()/2))

          {

              return it.first;

          }

      }

      return -1;

    }

};

**Approach 2 :**

class Solution {

public:

    int majorityElement(vector<int>& nums) {

      //size of the given array:

    int n = nums.size();

    int cnt = 0; // count

    int el; // Element

    //applying the algorithm:

    for (int i = 0; i < n; i++) {

        if (cnt == 0) {

            cnt = 1;

            el = nums[i];

        }

        else if (el == nums[i]) cnt++;

        else cnt--;

    }

    //checking if the stored element

    // is the majority element:

    int cnt1 = 0;

    for (int i = 0; i < n; i++) {

        if (nums[i] == el) cnt1++;

    }

    if (cnt1 > (n / 2)) return el;

    return -1;

    }

};

**229. Majority Element II**

class Solution {

public:

    vector<int> majorityElement(vector<int>& nums) {

        int num1 = -1;

        int num2 = -1;

        int count1 = 0;

        int count2 = 0;

        for(int i=0;i<nums.size();i++)

        {

            if(nums[i] == num1){

                count1++;

            }

            else if(nums[i] == num2){

                count2++;

            }

            else if(count1 == 0){

                num1 = nums[i];

                count1 = 1;

            }

            else if( count2 == 0){

                num2 = nums[i];

                count2 = 1;

            }

            else

            {

                count1--;

                count2--;

            }

        }

        vector<int> ans;

        count1 = 0;

        count2 = 0;

        for(int i=0;i<nums.size();i++)

        {

            if(nums[i] == num1){

                count1++;

            }

            else if(nums[i] == num2)

            {

                count2++;

            }

        }

        if(count1 > (nums.size()/3))

        {

            ans.push\_back(num1);

        }

        if(count2 > (nums.size()/3)){

            ans.push\_back(num2);

        }

        return ans;

    }

};

**62. Unique Paths**

**Memoization:**

class Solution {

public:

    int f(int i,int j,vector<vector<int>> &dp){

        if( i == 0 && j == 0) return 1;

        if( i < 0 || j < 0) return 0;

        if(dp[i][j]!=-1){

            return dp[i][j];

        }

        int up = f(i-1,j,dp);

        int left = f(i,j-1,dp);

        return dp[i][j] = up + left;

    }

    int uniquePaths(int m, int n) {

        vector<vector<int>> dp(m,vector<int>(n,-1));

       return f(m-1,n-1,dp);

    }

};

**Tabulation:**

class Solution {

public:

    int uniquePaths(int m, int n) {

      int dp[m][n];

      for(int i=0;i<m;i++)

      {

          for(int j=0;j<n;j++)

          {

              if(i == 0 && j == 0) dp[i][j] = 1;

              else{

                  int up = 0;

                  int left = 0;

                  if(i > 0) up = dp[i-1][j];

                  if(j > 0) left = dp[i][j-1];

                  dp[i][j] = up + left;

              }

          }

      }

      return dp[m-1][n-1];

    }

};

**493. Reverse Pairs**

class Solution {

public:

    int ans = 0;

    int reversePairs(vector<int>& nums) {

      int n = nums.size()-1;

      merge\_sort(nums,0,n);

      return ans;

    }

    void merge\_sort(vector<int>& arr,int left ,int right){

        if(left < right){

            int mid = left + (right-left)/2;

            merge\_sort(arr,left,mid);

            merge\_sort(arr,mid+1,right);

            merge(arr,left,mid,right);

        }

    }

    void merge(vector<int>& arr,int left,int mid,int right){

        int leftSize = mid - left + 1;

        int rightSize = right - mid;

        vector<int> left\_arr(leftSize),right\_arr(rightSize);

        for(int i=0;i<leftSize;i++){

            left\_arr[i] = arr[left + i];

        }

        for(int i=0;i<rightSize;i++)

        {

            right\_arr[i] = arr[mid + 1 + i];

        }

        int i=0;

        int j = 0;

        int cnt = 0;

        for(int i=0;i<leftSize;i++){

            while(j < rightSize && left\_arr[i] > (long) 2 \* right\_arr[j]){

                j++;

            }

            cnt +=j;

        }

        ans += cnt;

        i=0;

        j=0;

        int k = left;

        while(i < leftSize && j < rightSize){

            if(left\_arr[i] > right\_arr[j]){

                arr[k++] = right\_arr[j++];

            }

            else

            {

                arr[k++] = left\_arr[i++];

            }

        }

        while(i < leftSize){

            arr[k++] = left\_arr[i++];

        }

        while( j < rightSize){

            arr[k++] = right\_arr[j++];

        }

    }

};

1. **Two Sum**

**Approach 1:**

class Solution {

public:

    vector<int> twoSum(vector<int>& nums, int target) {

       int n = nums.size();

       vector<pair<int,int>> nums2;

       for(int i=0;i<n;i++){

           nums2.push\_back({nums[i],i});

       }

       sort(nums2.begin(),nums2.end());

       int left = 0;

       int right = n-1;

       while(left < right){

           int sum = nums2[left].first + nums2[right].first;

           if(sum == target){

               return {nums2[left].second,nums2[right].second};

           }

           else if(sum > target){

               right--;

           }

           else

           {

               left++;

           }

       }

       return {-1,-1};

    }

};

**Approach 2:**

class Solution {

public:

    vector<int> twoSum(vector<int>& nums, int target) {

    unordered\_map<int, int> mpp;

    for (int i = 0; i < n; i++) {

        int num = arr[i];

        int complement = target - num;

        if (mpp.find(complement) != mpp.end()) {

            return {mpp[complement], i};

        }

        mpp[num] = i;

    }

    return { -1, -1};

    }

};

18. 4Sum

class Solution {

public:

    vector<vector<int>> fourSum(vector<int>& nums, int target) {

        vector<vector<int>> res;

        if(nums.size() < 4){

            return res;

        }

        int n = nums.size();

        sort(nums.begin(),nums.end());

        for(int i=0;i<n;i++)

        {

            for(int j = i+1;j<n;j++)

            {

               long long int target2 = 1LL\*target - 1LL\*nums[ i ]  -  1LL\*nums[ j ];

                int left = j + 1;

                int right = n-1;

                while(left < right){

                    int twosum = nums[left] + nums[right];

                    if(twosum < target2) {

                        left++;

                    }

                    else if(twosum > target2){

                        right--;

                    }

                    else

                    {

                        vector<int> quadruplet(4,0);

                        quadruplet[0] = nums[i];

                        quadruplet[1] = nums[j];

                        quadruplet[2] = nums[left];

                        quadruplet[3] = nums[right];

                        res.push\_back(quadruplet);

                        while(left < right && nums[left] == quadruplet[2]) left++;

                        while(left < right && nums[right] == quadruplet[3]) right--;

                    }

                }

                while(j+1 < n && nums[j+1] == nums[j]) j++;

            }

            while(i+1 < n && nums[i+1] == nums[i]) i++;

        }

        return res;

    }

};

128. Longest Consecutive Sequence

Approach 1:

class Solution {

public:

    int longestConsecutive(vector<int>& nums) {

      if(nums.size() == 0) return 0;

      sort(nums.begin(),nums.end());

      int ans = 1;

      int cnt = 1;

      int prev = nums[0];

      for(int i=0;i<nums.size();i++)

      {

          if(nums[i] == prev + 1)

          {

              cnt++;

          }

          else if(nums[i] != prev){

              cnt = 1;

          }

          prev = nums[i];

          ans = max(ans,cnt);

      }

      return ans;

    }

};

Approach 2:

class Solution {

public:

    int longestConsecutive(vector<int>& nums) {

      int n = nums.size();

      if( n == 0) return 0;

      int longest = 1;

      unordered\_set<int> set;

      for(int i=0;i<n;i++)

      {

          set.insert(nums[i]);

      }

      for(auto it : set){

        if(set.find(it-1) == set.end()){

            int cnt = 1;

            int x = it;

            while(set.find(x+1)!=set.end()){

                x = x + 1;

                cnt = cnt + 1;

            }

            longest = max(longest,cnt);

        }

      }

      return longest;

    }

};

**Largest subarray with 0 sum**

class Solution{

public:

int maxLen(vector<int>&A, int n)

{

// Your code here

unordered\_map<int,int> map;

int maxi = 0;

int sum = 0;

for(int i=0;i<n;i++){

sum += A[i];

if( sum == 0){

maxi = i + 1;

}

else

{

if(map.find(sum) != map.end()){

maxi = max(maxi,i-map[sum]);

}

else

{

map[sum] = i;

}

}

}

return maxi;

}

};

**Subarray with given XOR**

int Solution::solve(vector<int> &A, int B) {

   vector<int> prefix(A.size(),0);

   prefix[0] = A[0];

   for(int i=1;i<A.size();i++)

   {

       prefix[i] = prefix[i-1] \* A[i];

   }

   int ans = 0;

   unordered\_map<int,int> map;

   for(int i=0;i<prefix.size();i++){

       int xor1 = B ^ prefix[i];

       if(map.find(xor1)!=map.end()){

           ans += map[xor1];

       }

       map[prefix[i]]++;

       if(prefix[i] == B){

           ans++;

       }

   }

   return ans;

}

Longest Substring Without Repeating Characters

class Solution {

public:

    int lengthOfLongestSubstring(string s) {

       vector<int> map(256,-1);

       int left = 0;

       int right = 0;

       int len = 0;

       int n = s.size();

       while(right < n)

       {

           if(map[s[right]] != -1){

               left = max(map[s[right]] + 1,left);

           }

           map[s[right]] = right;

           len = max(len,right -left + 1);

           right++;

       }

       return len;

    }

};

206. Reverse Linked List

Approach 1:

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        if(head == NULL) return NULL;

        ListNode \*prev = NULL;

        ListNode \*curr = head;

        ListNode \*nxt = head->next;

        while(curr != NULL){

            curr->next = prev;

            prev = curr;

            curr = nxt;

            if(nxt != NULL) nxt = nxt->next;

        }

        return prev;

    }

};

Approach 2:

class Solution {

public:

ListNode\* reverse(ListNode\* head){

    if(head->next == NULL) {

        return head;

    }

    ListNode\* reverseHead = reverse(head->next);

    head->next->next = head;

    head->next = NULL;

    return reverseHead;

}

    ListNode\* reverseList(ListNode\* head) {

        if(head == NULL) return NULL;

       return reverse(head);

    }

};

21. Merge Two Sorted Lists

Approach 1:

class Solution {

public:

    ListNode\* merge(ListNode\* l1,ListNode\* l2){

        if(l1 == NULL) return l2;

        if(l2 == NULL) return l1;

        if(l1->val < l2->val){

            l1->next = merge(l1->next,l2);

            return l1;

        }

        else

        {

            l2->next = merge(l1,l2->next);

            return l2;

        }

    }

    ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

        return merge(list1,list2);

    }

};

Approach 2:

class Solution {

public:

    ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

       if(list1 == NULL) return list2;

       if(list2 == NULL) return list1;

       ListNode \*ans,\*tail;

       if(list1->val < list2->val){

          ans = list1;

          tail = list1;

          list1 = list1->next;

       }

       else

       {

           ans = list2;

           tail = list2;

           list2 = list2->next;

       }

       while(list1 != NULL && list2 != NULL){

         if(list1->val < list2->val){

             tail->next = list1;

             tail = list1;

             list1 = list1->next;

         }

         else

         {

             tail->next = list2;

             tail = list2;

             list2 = list2->next;

         }

       }

       if(list1 == NULL) tail->next = list2;

       if(list2 == NULL) tail->next = list1;

       return ans;

    }

};

19. Remove Nth Node From End of List

class Solution {

public:

    ListNode\* removeNthFromEnd(ListNode\* head, int n) {

        ListNode \*start = new ListNode();

        start->next = head;

        ListNode \*fast = start;

        ListNode \*slow = start;

        for(int i=1;i<=n;i++)

        {

            fast = fast->next;

        }

        while(fast->next != NULL){

            fast = fast->next;

            slow = slow->next;

        }

        slow->next = slow->next->next;

        return start->next;

    }

};

1. Add Two Numbers
2. class Solution {
3. public:
4. ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2) {
5. ListNode \*dummy = new ListNode();
6. ListNode \*temp = dummy;
7. int carry = 0;
8. while(l1 != NULL || l2 != NULL || carry){
9. int sum = 0;
10. if(l1 != NULL){
11. sum += l1->val;
12. l1 = l1->next;
13. }
14. if(l2 != NULL){
15. sum += l2->val;
16. l2 = l2->next;
17. }
18. sum += carry;
19. carry = sum / 10;
20. ListNode \*node = new ListNode(sum % 10);
21. temp->next = node;
22. temp = temp ->next;
23. }
24. return dummy->next;
25. }
26. };

237. Delete Node in a Linked List

class Solution {

public:

    void deleteNode(ListNode\* node) {

       ListNode\* curr = node->next;

       node->val = curr->val;

       node->next = curr->next;

    }

};

160. Intersection of Two Linked Lists

Approach 1:

class Solution {

public:

    ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

       unordered\_set<ListNode\*> set;

       while(headA != NULL){

           set.insert(headA);

           headA = headA->next;

       }

       while(headB != NULL){

           if(set.find(headB)!=set.end()) return headB;

           headB = headB->next;

       }

       return NULL;

    }

};

Approach 2:

class Solution {

public:

    ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

        if( headA == NULL || headB == NULL) return NULL;

        ListNode \*d1 = headA;

        ListNode \*d2 = headB;

        while(d1 != d2){

            if(d1 == NULL){

               d1 = headB;

            }

            else{

               d1 = d1->next;

            }

            if(d2 == NULL){

               d2 = headA;

            }

            else{

               d2 = d2->next;

            }

        }

        return d1;

    }

};

141. Linked List Cycle

class Solution {

public:

    bool hasCycle(ListNode \*head) {

       if(head == NULL || head->next == NULL){

           return false;

       }

       ListNode \*slow = head;

       ListNode \*fast = head->next;

       while(slow != fast){

           if(fast == NULL || fast->next == NULL){

               return false;

           }

           slow = slow->next;

           fast = fast->next->next;

       }

       return true;

    }

};

25. Reverse Nodes in k-Group

Extra Space

class Solution {

public:

    void reverse(ListNode \*s, ListNode \*e){

        ListNode \*p = NULL, \*c = s, \*n = s->next;

        while(p != e){

            c->next = p;

            p = c;

            c = n;

            if(n!=NULL) n = n->next;

        }

    }

    ListNode\* reverseKGroup(ListNode\* head, int k) {

       if(head == NULL || head->next == NULL || k==1) return head;

       ListNode \*s = head , \*e = head;

       int inc = k-1;

       while(inc--){

           e = e->next;

           if( e == NULL){

               return head;

           }

       }

       ListNode \*nextHead = reverseKGroup(e->next,k);

       reverse(s,e);

       s->next = nextHead;

       return e;

    }

};

No Extra space

class Solution {

public:

    void reverse(ListNode \*s, ListNode \*e){

        ListNode \*p = NULL, \*c = s, \*n = s->next;

        while(p != e){

            c->next = p;

            p = c;

            c = n;

            if(n!=NULL) n = n->next;

        }

    }

    ListNode\* reverseKGroup(ListNode\* head, int k) {

       if(head == NULL || head->next == NULL || k==1) return head;

       ListNode \*dummy = new ListNode(-1);

       dummy->next = head;

       ListNode \*beforeStart = dummy , \*e = head;

       int i = 0;

       while(e !=NULL){

           i++;

           if( i % k == 0){

               //reversal

               ListNode \*s = beforeStart->next , \*temp = e->next;

               reverse(s,e);

               beforeStart->next = e;

               s->next = temp;

               beforeStart = s;

               e = temp;

           }

           else{

              e = e->next;

           }

       }

       return dummy->next;

    }

};

234. Palindrome Linked List

class Solution {

public:

    ListNode\* reverse(ListNode \*head){

        if(head == NULL) return NULL;

        ListNode \*p = NULL, \*c = head, \*n = head->next;

        while(c != NULL){

            c->next = p;

            p = c;

            c = n;

            if(n!=NULL) n = n->next;

        }

        return p;

    }

    bool isPalindrome(ListNode\* head) {

      ListNode \*slow = head, \*fast = head;

      while(fast->next != NULL && fast->next->next != NULL){

        slow = slow->next;

        fast = fast->next->next;

      }

      slow->next = reverse(slow->next);

      ListNode \*start = head, \*mid = slow->next;

      while(mid != NULL){

         if(mid->val != start->val) return false;

         start = start->next;

         mid = mid->next;

      }

slow->next = reverse(slow->next);

      return true;

    }

};

142. Linked List Cycle II

Approach 1

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

      unordered\_set<ListNode\*> set;

      ListNode \*curr = head;

      while(curr != NULL){

          if(set.find(curr) != set.end()){

              return curr;

          }

          else

          {

              set.insert(curr);

              curr = curr->next;

          }

      }

      return NULL;

    }

};

Approach 2:

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

       if(head == NULL) return NULL;

       ListNode \*slow = head, \*fast = head,  \*entry = head;

       while(fast->next != NULL && fast->next->next != NULL){

         slow = slow->next;

         fast = fast->next->next;

         if(slow == fast){

             while(entry != slow){

                 slow = slow->next;

                 entry = entry->next;

             }

             return slow;

         }

       }

       return NULL;

    }

};

**Flattening a Linked List**

Node \*mergeTwoLists(Node \*a,Node \*b){

Node \*temp = new Node(0);

Node \*res = temp;

while( a != NULL && b != NULL){

if(a->data < b->data){

temp->bottom = a;

temp= temp->bottom;

a = a->bottom;

}

else

{

temp->bottom = b;

temp = temp->bottom;

b = b->bottom;

}

}

if(a) temp->bottom = a;

else temp->bottom = b;

return res->bottom;

}

Node \*flatten(Node \*root)

{

// Your code here

if(root == NULL || root->next == NULL) return root;

root->next = flatten(root->next);

root = mergeTwoLists(root,root->next);

return root;

}

61. Rotate List

class Solution {

public:

    int size(ListNode \*head){

        int n = 0;

        while(head != NULL){

            n++;

            head = head->next;

        }

        return n;

    }

    ListNode\* rotateRight(ListNode\* head, int k) {

        if(head == NULL) return head;

        int n = size(head);

        int loop = k % n;

        loop = n - loop;

        if( n == 1 || loop == n){

            return head;

        }

        int j = 0;

        ListNode \*temp = head;

        ListNode \*firstAddress = head;

        while(temp != NULL){

           j++;

           while(j == loop){

            firstAddress = temp->next;

            temp->next = NULL;

            break;

           }

           temp = temp->next;

        }

        temp = firstAddress;

        while(temp->next != NULL){

            temp = temp->next;

        }

        temp->next = head;

        return firstAddress;

    }

};

138. Copy List with Random Pointer

Approach 1:

class Solution {

public:

    Node\* copyRandomList(Node\* head) {

        unordered\_map<Node\*,Node\*> map;

        Node \*ptr = head;

        while(ptr != NULL){

           map[ptr] = new Node(ptr->val);

           ptr = ptr->next;

        }

        ptr = head;

        while(ptr != NULL){

         map[ptr]->next = map[ptr->next];

         map[ptr]->random = map[ptr->random];

         ptr = ptr->next;

        }

        return map[head];

    }

};

Approach 2:

class Solution {

public:

    Node\* copyRandomList(Node\* head) {

    if (head == NULL)

        return NULL;

    Node \*newHead, \*l1=head, \*l2;

    while (l1 != NULL)

    {

        l2 = new Node(l1->val);

        l2->next = l1->next;

        l1->next = l2;

        l1 = l1->next->next;

    }

    l1 = head;

    newHead = head->next;

    while ( l1 != NULL)

    {

        if (l1->random != NULL)

        {

            l1->next->random = l1->random->next;

        }

         l1 = l1->next->next;

    }

    l1 = head;

    while ( l1 != NULL)

    {

        l2 = l1->next;

        l1->next = l2->next;

        if (l2->next != NULL)

        {

            l2->next = l2->next->next;

        }

        l1 = l1->next;

    }

    return newHead;

    }

};

15. 3Sum

class Solution {

public:

    vector<vector<int>> threeSum(vector<int>& nums) {

      sort(nums.begin(),nums.end());

      vector<vector<int>> res;

      for(int i=0;i<nums.size()-2;i++){

          if(i == 0 || (i > 0 && nums[i] != nums[i-1])){

              int low = i + 1;

              int high = (int)(nums.size())-1;

              int sum = 0 - nums[i];

              while(low < high){

                  if(nums[low] + nums[high] == sum){

                      vector<int> temp;

                      temp.push\_back(nums[i]);

                      temp.push\_back(nums[low]);

                      temp.push\_back(nums[high]);

                      res.push\_back(temp);

                      while (low < high && nums[low] == nums[low+1]) low++;

                      while (low < high && nums[high] == nums[high-1]) high--;

                       low++; high--;

                  }

                  else if (nums[low] + nums[high] < sum) low++;

                  else high--;

              }

          }

      }

       return res;

    }

};

42. Trapping Rain Water

Approach 1

class Solution {

public:

    int trap(vector<int>& height) {

        int n = height.size();

        int waterTrapped = 0;

        for(int i=0;i<n;i++){

            int j = i;

            int leftMax = 0;

            int rightMax = 0;

            while(j >= 0){

               leftMax = max(leftMax,height[j]);

               j--;

            }

            j=i;

            while( j < n){

                rightMax = max(rightMax,height[j]);

                j++;

            }

            waterTrapped += min(leftMax,rightMax) -height[i];

        }

        return waterTrapped;

    }

};

Approach 2:

class Solution {

public:

    int trap(vector<int>& height) {

      int n = height.size();

      int left = 0;

      int right = n-1;

      int res = 0;

      int maxleft = 0;

      int maxright = 0;

      while(left < right){

        if(height[left] <= height[right]){

            if(height[left] >= maxleft){

                maxleft = height[left];

            }

            else

            {

                res += maxleft - height[left];

            }

            left++;

        }

        else

        {

            if(height[right] >= maxright){

                maxright = height[right];

            }

            else

            {

                res += maxright - height[right];

            }

            right--;

        }

      }

      return res;

    }

};

26. Remove Duplicates from Sorted Array

var removeDuplicates = function(nums) {

    let insertIndex = 1;

    for(let i = 1; i < nums.length; i++)

    {

        if(nums[i - 1] != nums[i])

        {

            nums[insertIndex] = nums[i];

            insertIndex++;

        }

    }

    return insertIndex;

};

485. Max Consecutive Ones

class Solution {

public:

    int findMaxConsecutiveOnes(vector<int>& nums) {

       int count = 0;

       int maxOnes = 0;

       for(int i=0;i<nums.size();i++){

           if(nums[i] == 1){

               count++;

           }

           else

           {

               count = 0;

           }

           maxOnes = max(maxOnes,count);

       }

       return maxOnes;

    }

};

**N meetings in one room**

**class Solution**

**{**

**public:**

**//Function to find the maximum number of meetings that can**

**//be performed in a meeting room.**

**int maxMeetings(int start[], int end[], int n)**

**{**

**// Your code here**

**vector<pair<int,int>> meet;**

**for(int i=0;i<n;i++)**

**meet.push\_back({end[i],start[i]});**

**sort(meet.begin(),meet.end());**

**int last = meet[0].first;**

**int task = 1;**

**for(int i=1;i<n;i++)**

**{**

**if(meet[i].second > last)**

**{**

**task++;**

**last = meet[i].first;**

**}**

**}**

**return task;**

**}**

**};**

**Minimum Platforms**

**Approach 1**

**class Solution{**

**public:**

**//Function to find the minimum number of platforms required at the**

**//railway station such that no train waits.**

**int findPlatform(int arr[], int dep[], int n)**

**{**

**// Your code here**

**int ans = 1;**

**for(int i=0;i<n-1;i++){**

**int count = 1;**

**for(int j=i+1;j<=n-1;j++){**

**if((arr[i] >= arr[j] && arr[i] <= dep[j]) || (arr[j] >= arr[i] && arr[j] <= dep[i])){**

**count++;**

**}**

**}**

**ans = max(ans,count);**

**}**

**return ans;**

**}**

**};**

**Approach 2:**

**class Solution{**

**public:**

**//Function to find the minimum number of platforms required at the**

**//railway station such that no train waits.**

**int findPlatform(int arr[], int dep[], int n)**

**{**

**// Your code here**

**sort(arr,arr+n);**

**sort(dep,dep+n);**

**int result = 1;**

**int plat\_needed = 1;**

**int i = 1;**

**int j = 0;**

**while( i < n && j < n){**

**if(arr[i] <= dep[j]){**

**plat\_needed++;**

**i++;**

**}**

**else if(arr[i] > dep[j]){**

**plat\_needed--;**

**j++;**

**}**

**if(plat\_needed > result){**

**result = plat\_needed;**

**}**

**}**

**return result;**

**}**

**};**

**Job Sequencing Problem**

**/\***

**struct Job**

**{**

**int id; // Job Id**

**int dead; // Deadline of job**

**int profit; // Profit if job is over before or on deadline**

**};**

**\*/**

**class Solution**

**{**

**public:**

**//Function to find the maximum profit and the number of jobs done.**

**vector<int> JobScheduling(Job arr[], int n)**

**{**

**// your code here**

**vector<pair<int,int>> v(n);**

**for(int i = 0;i<n;i++){**

**v[i].first = arr[i].dead;**

**v[i].second = arr[i].profit;**

**}**

**sort(v.begin(),v.end());**

**priority\_queue<int,vector<int>,greater<int>> pq;**

**int total = 0;**

**for(int i=0;i<n;i++){**

**if(total<v[i].first){**

**total++;**

**pq.push(v[i].second);**

**}**

**else{**

**if(pq.top()<v[i].second){**

**pq.pop();**

**pq.push(v[i].second);**

**}**

**}**

**}**

**int sum = 0;**

**while(!pq.empty()){**

**sum+=pq.top();**

**pq.pop();**

**}**

**return {total,sum};**

**}**

**};**

**Fractional Knapsack**

**//class implemented**

**/\***

**struct Item{**

**int value;**

**int weight;**

**};**

**\*/**

**class Solution**

**{**

**public:**

**bool static comp(Item a , Item b){**

**double r1 = (double) a.value / (double) a.weight;**

**double r2 = (double) b.value / (double) b.weight;**

**return r1 > r2;**

**}**

**//Function to get the maximum total value in the knapsack.**

**double fractionalKnapsack(int W, Item arr[], int n)**

**{**

**// Your code here**

**sort(arr,arr+n,comp);**

**int currWeight = 0;**

**double finalvalue = 0.0;**

**for(int i=0;i<n;i++){**

**if(currWeight + arr[i].weight <= W){**

**currWeight += arr[i].weight;**

**finalvalue += arr[i].value;**

**}**

**else**

**{**

**int remain = W - currWeight;**

**finalvalue += (arr[i].value / (double) arr[i].weight) \* (double) remain;**

**break;**

**}**

**}**

**return finalvalue;**

**}**

**};**

**Number of Coins**

**class Solution{**

**public:**

**int recur(int sum,int i,int coins[],int M,vector<vector<int>>&dp)**

**{**

**if(sum<0 or i==M)**

**return 1e8;**

**if(sum==0)**

**return 0;**

**if(dp[i][sum]!=-1)**

**return dp[i][sum];**

**int pick=recur(sum-coins[i],i,coins,M,dp)+1;**

**int not\_pick=recur(sum,i+1,coins,M,dp);**

**return dp[i][sum]=min(pick,not\_pick);**

**}**

**int minCoins(int coins[], int M, int V)**

**{**

**vector<vector<int>>dp(M+1,vector<int>(V+1,-1));**

**if(recur(V,0,coins,M,dp)>=1e8)**

**return -1;**

**return recur(V,0,coins,M,dp);**

**}**

**};**

**Subset Sums**

**class Solution**

**{**

**public:**

**void helper(vector<int> &ans,vector<int> arr,int size,int index,int sum){**

**if(index >= size){**

**ans.push\_back(sum);**

**return;**

**}**

**helper(ans,arr,size,index+1,sum);**

**helper(ans,arr,size,index+1,sum+arr[index]);**

**}**

**vector<int> subsetSums(vector<int> arr, int N)**

**{**

**// Write Your Code here**

**vector<int> ans;**

**int sum = 0;**

**int index = 0;**

**helper(ans,arr,N,index,sum);**

**return ans;**

**}**

**};**

**90. Subsets II**

class Solution {

public:

    vector<vector<int>> subsetsWithDup(vector<int>& nums) {

      vector<vector<int>> ans;

      vector<int> ds;

      sort(nums.begin(),nums.end());

      findSubsets(0,nums,ds,ans);

      return ans;

    }

    void findSubsets(int ind,vector<int> &nums,vector<int> &ds,vector<vector<int>> &ans){

        ans.push\_back(ds);

        for(int i = ind;i<nums.size();i++){

            if( i != ind && nums[i] == nums[i-1]) continue;

            ds.push\_back(nums[i]);

            findSubsets(i+1,nums,ds,ans);

            ds.pop\_back();

        }

    }

};

39. Combination Sum

class Solution {

public:

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

        vector<vector<int>> ans;

        vector<int> ds;

        findCombination(0,target,candidates,ans,ds);

        return ans;

    }

    void findCombination(int ind,int target,vector<int> &arr,vector<vector<int>> &ans , vector<int> &ds){

        if(ind == arr.size()){

            if(target == 0){

                ans.push\_back(ds);

            }

            return;

        }

        if(arr[ind] <= target){

            ds.push\_back(arr[ind]);

            findCombination(ind,target - arr[ind],arr,ans,ds);

            ds.pop\_back();

        }

        findCombination(ind + 1, target,arr,ans,ds);

    }

};

40. Combination Sum II

class Solution {

private :

void help(int i, vector<int> &arr, int n, vector<int> &subSet, vector<vector<int>> &powerSet, int sum, int target) {

    // If sum is equal to target, we have reached a Valid Combination

    if(sum == target)

    {

        powerSet.push\_back(subSet);

        return;

    }

    // If at any moment, sum becomes greater than target, we don't need to proceed further

    if(sum > target) return ;

    // If we reach the end of arr[], we cannot go any further so we return back

    if(i == n) return;

    // Include the i-th Element into our Subset & Sum

    subSet.push\_back(arr[i]);

    sum += arr[i];

    // Ask recursion to do rest of the task

    help(i + 1, arr, n, subSet, powerSet, sum, target) ;

    // Backtrack and undo the change we have done

    subSet.pop\_back();

    sum -= arr[i];

    // Use the While Loop to skip all the duplicate occurrences of i-th Element

    while(i + 1 < arr.size() && arr[i] == arr[i + 1]) i++ ;

    // Don't pick the i-th Element and ask recursion to do rest of the task

    help(i + 1, arr, n, subSet, powerSet, sum, target) ;

}

public:

    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

    vector<int> subSet ;

    vector<vector<int>> powerSet ;

    int sum = 0;

    int n = candidates.size() ;

    sort(candidates.begin(), candidates.end()) ;

    help(0, candidates, n, subSet, powerSet, sum, target) ;

    return powerSet ;

}

};

/\*

Time Complexity: O(2^N)

Space Complexity: O(N)

\*/

131. Palindrome Partitioning

class Solution {

public:

    vector<vector<string>> partition(string s) {

       vector<vector<string>> res;

       vector<string> path;

       func(0,s,path,res);

       return res;

    }

    void func(int index,string s,vector<string> &path,vector<vector<string>> &res){

        if(index == s.size()){

            res.push\_back(path);

            return;

        }

        for(int i=index;i<s.size();i++){

            if(isPalindrome(s,index,i)){

                path.push\_back(s.substr(index,i-index+1));

                func(i+1,s,path,res);

                path.pop\_back();

            }

        }

    }

    bool isPalindrome(string s,int start,int end){

        while(start <= end){

            if(s[start++] != s[end--]){

                return false;

            }

        }

        return true;

    }

};

60. Permutation Sequence

class Solution {

public:

    string getPermutation(int n, int k) {

        int fact = 1;

        vector<int> numbers;

        for(int i=1;i<n;i++){

            fact = fact \* i;

            numbers.push\_back(i);

        }

        numbers.push\_back(n);

        string ans = "";

        k = k-1;

        while(true){

            ans = ans + to\_string(numbers[k/fact]);

            numbers.erase(numbers.begin() + k / fact);

            if(numbers.size() == 0){

                break;

            }

            k = k % fact;

            fact = fact / numbers.size();

        }

        return ans;

    }

};

46. Permutations

class Solution {

public:

    vector<vector<int>> permute(vector<int>& nums) {

      vector<vector<int>> ans;

      vector<int> ds;

      int freq[nums.size()];

      for(int i=0;i<nums.size();i++) freq[i] = 0;

      recurPermute(ds,nums,ans,freq);

      return ans;

    }

    void recurPermute(vector<int> &ds,vector<int> &nums,vector<vector<int>> &ans,int freq[]){

        if(ds.size() == nums.size()){

            ans.push\_back(ds);

            return;

        }

        for(int i=0;i<nums.size();i++){

            if(!freq[i]){

                ds.push\_back(nums[i]);

                freq[i] = 1;

                recurPermute(ds,nums,ans,freq);

                freq[i] = 0;

                ds.pop\_back();

            }

        }

    }

};

N Queens

class Solution {

public:

    vector<vector<string>> ret;

    bool is\_valid(vector<string> &board, int row, int col){

        // check col

        for(int i=row;i>=0;--i)

            if(board[i][col] == 'Q') return false;

        // check left diagonal

        for(int i=row,j=col;i>=0&&j>=0;--i,--j)

            if(board[i][j] == 'Q') return false;

        //check right diagonal

        for(int i=row,j=col;i>=0&&j<board.size();--i,++j)

            if(board[i][j] == 'Q') return false;

        return true;

    }

    void dfs(vector<string> &board, int row){

        // exit condition

        if(row == board.size()){

            ret.push\_back(board);

            return;

        }

        // iterate every possible position

        for(int i=0;i<board.size();++i){

            if(is\_valid(board,row,i)){

                // make decision

                board[row][i] = 'Q';

                // next iteration

                dfs(board,row+1);

                // back-tracking

                board[row][i] = '.';

            }

        }

    }

    vector<vector<string>> solveNQueens(int n) {

        // return empty if n <= 0

        if(n <= 0) return {{}};

        vector<string> board(n,string(n,'.'));

        dfs(board,0);

        return ret;

    }

};

[**37. Sudoku Solver**](https://leetcode.com/problems/sudoku-solver/)

class Solution {

public:

    void solveSudoku(vector<vector<char>>& board) {

        solve(board);

    }

    bool solve(vector<vector<char>>& board){

        for(int i=0; i<board.size(); i++){

            for(int j=0; j<board[0].size(); j++){

                if(board[i][j] == '.'){

                    for(char c='1'; c<='9'; c++){

                        if(valid(c, board, i, j)){

                            board[i][j] = c;

                            if(solve(board) == true) return true;

                            else board[i][j]='.';

                        }

                    }

                    return false;

                }

            }

        }

        return true;

    }

    bool valid(char c, vector<vector<char>>& board, int row, int col){

        for(int i=0; i<9; i++){

            if(board[i][col] == c) return false;

            if(board[row][i] == c) return false;

            if(board[3\*(row/3)+(i/3)][3\*(col/3)+(i%3)] == c) return false;

        }

        return true;

    }

};

### M-Coloring Problem

class Solution{

public:

// Function to determine if graph can be coloured with at most M colours such

// that no two adjacent vertices of graph are coloured with same colour.

bool graphColoring(bool graph[101][101], int m, int n) {

// your code here

int color[n] = {0};

if(solve(0,color,m,n,graph)) return true;

return false;

}

bool solve(int node,int color[],int m,int n,bool graph[101][101]){

if(node == n) return true;

for(int i = 1 ;i<=m ; i++)

{

if(isSafe(node,color,graph,n,i))

{

color[node] = i;

if(solve(node + 1,color,m,n,graph)) return true;

color[node] = 0;

}

}

return false;

}

bool isSafe(int node ,int color[],bool graph[101][101],int n,int col)

{

for(int k=0;k<n;k++)

{

if(k != node && graph[k][node] == 1 && color[k] == col){

return false;

}

}

return true;

}

};

### Rat in a Maze Problem - I

// User function template for C++

class Solution{

public:

vector<string> findPath(vector<vector<int>> &m, int n) {

// Your code goes here

vector<string> ans;

vector<vector<int>> visited(n,vector<int>(n,0));

if(m[0][0] == 1) solve(0,0,m,n,ans,"",visited);

return ans;

}

void solve(int i,int j,vector<vector<int>> &arr,int n , vector<string> &ans, string move , vector<vector<int>> &visited){

if( i == n-1 && j == n-1){

ans.push\_back(move);

return;

}

//down

if(i+1 < n && !visited[i+1][j] && arr[i+1][j] == 1){

visited[i][j] = 1;

solve(i+1,j,arr,n,ans,move + "D" , visited);

visited[i][j] = 0;

}

//left

if(j-1 >= 0 && !visited[i][j-1] && arr[i][j-1] == 1){

visited[i][j] = 1;

solve(i,j-1,arr,n,ans,move + "L",visited);

visited[i][j] = 0;

}

//right

if(j+1 < n && !visited[i][j+1] && arr[i][j+1] == 1){

visited[i][j] = 1;

solve(i,j+1,arr,n,ans,move + "R",visited);

visited[i][j] = 0;

}

//up

if(i-1 >=0 && !visited[i-1][j] && arr[i-1][j] == 1){

visited[i][j] = 1;

solve(i-1,j,arr,n,ans,move + "U" ,visited);

visited[i][j] = 0;

}

}

};

[139. Word Break](https://leetcode.com/problems/word-break/)

TLE

class Solution {

public:

    unordered\_set<string> setOfWords;

    bool wordBreak(string s, vector<string>& wordDict) {

        for(auto word : wordDict){

            setOfWords.insert(word);

        }

        return solve(s,0);

    }

    bool solve(string s,int position){

        if(position == s.size()) return true;

        for(int i=position;i<s.size();i++)

        {

           if(setOfWords.count(s.substr(position, i+1-position)) && solve(s,i+1)){

               return true;

           }

        }

        return false;

    }

};

Accepted

class Solution {

public:

    unordered\_map<int,bool> memoized;

    unordered\_set<string> setOfWords;

    bool wordBreak(string s, vector<string>& wordDict) {

        for(auto word : wordDict){

            setOfWords.insert(word);

        }

        return solve(s,0);

    }

    bool solve(string s,int position){

        if(position == s.size()) return true;

        if(memoized.count(position)) return memoized[position];

        for(int i=position;i<s.size();i++)

        {

           if(setOfWords.count(s.substr(position, i+1-position)) && solve(s,i+1)){

               return  memoized[position] = true;

           }

        }

        return memoized[position] = false;

    }

};

BFS-SOLUTION

class Solution {

public:

    bool wordBreak(string s, vector<string>& wordDict) {

        unordered\_set<string> setOfWords;

        unordered\_set<int> visited;

        queue<int> q;

        q.push(0);

        for(string word: wordDict){

            setOfWords.insert(word);

        }

        while(!q.empty()){

            int curr = q.front();

            q.pop();

            if(!visited.count(curr)){

                visited.insert(curr);

                string temp = "";

                for(int start = curr;start < s.size();start++)

                {

                    temp.push\_back(s[start]);

                    if(setOfWords.count(temp)){

                        q.push(start + 1);

                        if(start == s.size()-1){

                            return true;

                        }

                    }

                }

            }

        }

        return false;

    }

};

### Word Break - Part 2

// User function Template for C++

class Solution{

public:

unordered\_set<string> setOfWords;

vector<string> ans;

vector<string> wordBreak(int n, vector<string>& dict, string s)

{

// code here

ans.clear();

for(string word : dict){

setOfWords.insert(word);

}

string temp = "";

solve(0,temp,s);

return ans;

}

void solve(int index , string &temp , string &s){

if(index >= s.size())

{

temp.pop\_back();

ans.push\_back(temp);

temp.push\_back(' ');

}

string curr = temp;

for(int i=index;i<s.size();i++)

{

string new\_string = s.substr(index,i+1 -index);

if(setOfWords.count(new\_string))

{

temp += new\_string;

temp.push\_back(' ');

solve(i+1,temp,s);

temp = curr;

}

}

}

};

#include <bits/stdc++.h>

using namespace std;

// Power exponential method:

long long func(int b, int exp) {

long long ans = 1;

long long base = b;

while (exp > 0) {

if (exp % 2) {

exp--;

ans = ans \* base;

}

else {

exp /= 2;

base = base \* base;

}

}

return ans;

}

**Nth Root of a Number**

**BRUTE\_FORCE**

int NthRoot(int n, int m) {

//Use linear search on the answer space:

for (int i = 1; i <= m; i++) {

long long val = func(i, n);

if (val == m \* 1ll) return i;

else if (val > m \* 1ll) break;

}

return -1;

}

int main()

{

int n = 3, m = 27;

int ans = NthRoot(n, m);

cout << "The answer is: " << ans << "\n";

return 0;

}

**OPTIMAL(BINARY SEARCH)**

int NthRoot(int n, int m) {

  // Write your code here.

  int low = 1;

  int high = m;

   while(low <= high){

     int mid = low + (high-low)/2;

     int midN = func(mid,n);

     if(midN == 1) return mid;

     else if(midN == 0) low = mid + 1;

     else high = mid-1;

   }

   return -1;

}

int func(int mid,int m,int n){

  long long ans = 1;

  for(int i = 1;i<=n;i++){

     ans = ans \* mid;

     if(ans > m) return 2;

  }

  if(ans == m) return 1;

  return 0;

}

### Median in a row-wise sorted Matrix

//User function template for C++

class Solution{

public:

int median(vector<vector<int>> &matrix, int R, int C){

// code here

int low = 1;

int high = 2000;

while(low <= high){

int mid = low + (high-low)/2;

int count = 0;

for(int i=0;i<R;i++){

count += countSmallerMid(matrix[i],C,mid);

}

if(count <= (R \* C) / 2) low = mid + 1;

else high = mid -1;

}

return low;

}

int countSmallerMid(vector<int> &row ,int C ,int mid){

int low =0;

int high = C-1;

while(low <= high){

int mid2 = low + (high - low)/2;

if(row[mid2] <= mid){

low = mid2 + 1;

}

else if(row[mid2] > mid){

high = mid2 -1;

}

}

return low;

}

};

[540. Single Element in a Sorted Array](https://leetcode.com/problems/single-element-in-a-sorted-array/)

class Solution {

public:

    int singleNonDuplicate(vector<int>& nums) {

       for(int i=0;i<nums.size();i++)

       {

           if(i == 0){

              if(nums[i] != nums[i+1]) return nums[0];

           }

           else if( i == nums.size()-1){

               if(nums[i] != nums[i-1]) return nums[nums.size()-1];

           }

           else{

              if(nums[i] != nums[i+1] && nums[i] != nums[i-1]) return nums[i];

           }

       }

       return -1;

    }

};

1. [Search in Rotated Sorted Array](https://leetcode.com/problems/search-in-rotated-sorted-array/)

class Solution {

public:

    int search(vector<int>& nums, int target) {

      int n = nums.size();

      int low = 0;

      int high = n-1;

      while(low <= high){

          int mid = low + ( high - low)/2;

          if(nums[mid] == target) return mid;

          //left half sorted

          if(nums[low] <= nums[mid]){

              if(nums[low] <= target && target <= nums[mid]){

                  high = mid -1;

              }

              else{

                  low = mid + 1;

              }

          }

          else{//right half sorted

              if(nums[mid] <= target && target <= nums[high]){

                  low = mid + 1;

              }

              else{

                  high = mid -1;

              }

          }

        }

        return -1;

    }

};

# [Median of Two Sorted Arrays](https://leetcode.com/problems/median-of-two-sorted-arrays/)

Brute force

class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

       vector<int> arr3;

       int n1 = nums1.size();

       int n2 = nums2.size();

       int i = 0;

       int j = 0;

       while(i < n1 && j < n2){

           if(nums1[i] < nums2[j] ) arr3.push\_back(nums1[i++]);

           else arr3.push\_back(nums2[j++]);

       }

       while( i < n1 ) arr3.push\_back(nums1[i++]);

       while( j < n2 ) arr3.push\_back(nums2[j++]);

       int n = n1 + n2;

       if( n % 2 == 1){

           return arr3[n/2];

       }

       return (double)((double)(arr3[n/2]) + (double)(arr3[n/2 -1])) / 2.0;

    }

};

Better Approach

class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

    //size of two given arrays:

    int n1 = nums1.size(), n2 = nums2.size();

    int n = n1 + n2; //total size

    //required indices:

    int ind2 = n / 2;

    int ind1 = ind2 - 1;

    int cnt = 0;

    int ind1el = -1, ind2el = -1;

    //apply the merge step:

    int i = 0, j = 0;

    while (i < n1 && j < n2) {

        if (nums1[i] < nums2[j]) {

            if (cnt == ind1) ind1el = nums1[i];

            if (cnt == ind2) ind2el = nums1[i];

            cnt++;

            i++;

        }

        else {

            if (cnt == ind1) ind1el = nums2[j];

            if (cnt == ind2) ind2el = nums2[j];

            cnt++;

            j++;

        }

    }

    //copy the left-out elements:

    while (i < n1) {

        if (cnt == ind1) ind1el = nums1[i];

        if (cnt == ind2) ind2el = nums1[i];

        cnt++;

        i++;

    }

    while (j < n2) {

        if (cnt == ind1) ind1el = nums2[j];

        if (cnt == ind2) ind2el = nums2[j];

        cnt++;

        j++;

    }

    //Find the median:

    if (n % 2 == 1) {

        return (double)ind2el;

    }

    return (double)((double)(ind1el + ind2el)) / 2.0;

    }

};

Optimal

class Solution {

public:

    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

        int n1 = nums1.size();

        int n2 = nums2.size();

        if(n1 > n2) return findMedianSortedArrays(nums2,nums1);

        int low = 0;

        int high = n1;

        int left = (n1 + n2 + 1)/2;

        int n = n1 + n2;

        while(low <= high){

            int mid1 = low + (high-low)/2;

            int mid2 = left - mid1;

            int l1 = INT\_MIN;

            int l2 = INT\_MIN;

            int r1 = INT\_MAX;

            int r2 = INT\_MAX;

            if(mid1 < n1) r1 = nums1[mid1];

            if(mid2 < n2) r2 = nums2[mid2];

            if(mid1 -1 >= 0) l1 = nums1[mid1-1];

            if(mid2 -1 >= 0) l2 = nums2[mid2-1];

            if(l1 <= r2 && l2 <= r1) {

                if(n % 2 == 1) return max(l1,l2);

                return ((double)(max(l1,l2) + min(r1,r2)))/2.0;

            }

            else if(l1 > r2) high = mid1 -1;

            else low = mid1 + 1;

        }

        return 0;

    }

};

### K-th element of two Arrays

class Solution{

public:

int kthElement(int arr1[], int arr2[], int n1, int n2, int k)

{

if(n1 > n2) return kthElement(arr2,arr1,n2,n1,k);

int low = max(0,k-n2);

int high = min(k,n1);

int left = k;

int n = n1 + n2;

while(low <= high){

int mid1 = low + (high-low)/2;

int mid2 = left - mid1;

int l1 = INT\_MIN;

int l2 = INT\_MIN;

int r1 = INT\_MAX;

int r2 = INT\_MAX;

if(mid1 < n1) r1 = arr1[mid1];

if(mid2 < n2) r2 = arr2[mid2];

if(mid1 -1 >= 0) l1 = arr1[mid1-1];

if(mid2 -1 >= 0) l2 = arr2[mid2-1];

if(l1 <= r2 && l2 <= r1) {

return max(l1,l2);

}

else if(l1 > r2) high = mid1 -1;

else low = mid1 + 1;

}

return 0;

}

};

### Allocate minimum number of pages

//User function template in C++

class Solution

{

public:

int isPossible(int maxi,int arr[],int n,int m){

int count = 0;

int sum = 0;

for(int i=0;i<n;i++){

if(sum + arr[i] <= maxi){

sum += arr[i];

}

else{

count++;

sum = arr[i];

}

}

return count;

}

//Function to find minimum number of pages.

int findPages(int A[], int N, int M)

{

//code here

if( N < M) return -1;

int high = A[0];

int low = A[0];

for(int i=1;i<N;i++){

high += A[i];

low = max(low,A[i]);

}

while(low <= high){

int mid = low + (high - low)/2;

if(isPossible(mid,A,N,M) >= M){

low = mid + 1;

}else{

high = mid -1;

}

}

return low;

}

};

### Aggressive Cows

// User function Template for C++

class Solution {

public:

int solve(int n, int k, vector<int> &stalls) {

// Write your code here

sort(stalls.begin(),stalls.end());

int low = 0;

int high = stalls[n-1];

int best = 0;

while(low <= high){

int mid = (low + high + 1)/2;

int count = 1;

int left = 0;

for(int i = 1 ;i < n && count < k;i++){

if(stalls[i] - stalls[left] >= mid){

left = i;

count++;

}

}

if(count >= k){

best = mid;

low = mid + 1;

}

else{

high = mid -1;

}

}

return best;

}

};

### Kth smallest element

class Solution{

public:

// arr : given array

// l : starting index of the array i.e 0

// r : ending index of the array i.e size-1

// k : find kth smallest element and return using this function

int kthSmallest(int arr[], int l, int r, int k) {

//code here

priority\_queue<int> maxh;

for(int i=0;i<=r;i++){

maxh.push(arr[i]);

if(maxh.size() > k) {

maxh.pop();

}

}

return maxh.top();

}

};

### Is Binary Tree Heap

class Solution {

public:

int countNodes(struct Node\* root){

if(root == NULL) return 0;

int ans = 1 + countNodes(root->left) + countNodes(root->right);

return ans;

}

bool isCBT(struct Node\* root, int index, int totalCount){

if(root == NULL) return true;

if(index >= totalCount){

return false;

}

bool left = isCBT(root->left,2\*index + 1,totalCount);

bool right = isCBT(root->right,2\*index + 2,totalCount);

return (left && right);

}

bool isMaxHeap(struct Node\* root){

if(root == NULL) return true;

if(root->left != NULL && root->data < root->left->data) return false;

if(root->right != NULL && root->data < root->right->data) return false;

bool left = isMaxHeap(root->left);

bool right = isMaxHeap(root->right);

return left && right;

}

bool isHeap(struct Node\* tree) {

// code here

int index = 0;

int totalCount = countNodes(tree);

if(isCBT(tree,index,totalCount) && isMaxHeap(tree)){

return true;

}

return false;

}

};

### Merge two binary Max heaps

class Solution{

public:

void heapify(vector<int> &arr,int n,int i){

int largest = i;

int left = 2\*i + 1;

int right = 2\*i + 2;

if(left < n && arr[largest] < arr[left]){

largest = left;

}

if(right < n && arr[largest] < arr[right]){

largest = right;

}

if(largest != i){

swap(arr[largest],arr[i]);

heapify(arr,n,largest);

}

}

vector<int> mergeHeaps(vector<int> &a, vector<int> &b, int n, int m) {

// your code here

vector<int> c;

for(auto i : a){

c.push\_back(i);

}

for(auto i : b){

c.push\_back(i);

}

int size=c.size();

for(int i = size /2 -1;i>=0;i--){

heapify(c,size,i);

}

return c;

}

};

### Minimum Cost of ropes

### class Solution

### {

### public:

### //Function to return the minimum cost of connecting the ropes.

### long long minCost(long long arr[], long long n) {

### // create a min heap

### priority\_queue<long long, vector<long long>, greater<long long>> pq;

### for(long long i = 0; i < n; i++)

### pq.push(arr[i]);

### long long cost = 0;

### while(pq.size() > 1)

### {

### long long a = pq.top();

### pq.pop();

### long long b = pq.top();

### pq.pop();

### long long sum = a + b;

### cost += sum;

### pq.push(sum);

### }

### return cost;

### }

### };

### K-th Largest Sum Contiguous Subarray

class Solution{

public:

int kthLargest(vector<int> &Arr,int N,int K){

priority\_queue<int, vector<int>, greater<int>> pq;

for(int i=0;i<N;i++){

int sum = 0;

for(int j = i;j < N; j++){

sum += Arr[j];

if(pq.size() < K){

pq.push(sum);

}

else if(sum > pq.top()){

pq.pop();

pq.push(sum);

}

}

}

return pq.top();

}

};

### Merge k Sorted Arrays

class Solution

{

public:

//Function to merge k sorted arrays.

vector<int> mergeKArrays(vector<vector<int>> arr, int K)

{

//code here

vector<int> ans;

priority\_queue<int, vector<int>, greater<int>> pq;

for(auto it: arr){

for(int i=0;i<it.size();i++){

pq.push(it[i]);

}

}

while(!pq.empty()){

ans.push\_back(pq.top());

pq.pop();

}

return ans;

}

};

Sol2:

class Solution

{

public:

class Node{

public:

int data;

int row;

int col;

Node(int a,int row,int col){

this->data = a;

this->row = row;

this->col = col;

}

};

class compare{

public:

bool operator()(Node\* a , Node\* b){

return a->data > b->data;

}

};

//Function to merge k sorted arrays.

vector<int> mergeKArrays(vector<vector<int>> arr, int K)

{

//code here

priority\_queue<Node\*, vector<Node\*>,compare> pq;

vector<int> ans;

for(int i=0;i<K;i++){

Node\* temp = new Node(arr[i][0],i,0);

pq.push(temp);

}

while(pq.size() > 0 ){

Node\* temp = pq.top();

ans.push\_back(temp->data);

pq.pop();

int row = temp->row;

int col = temp->col;

if(col + 1 < arr[row].size()){

Node\* next = new Node(arr[row][col+1],row,col+1);

pq.push(next);

}

}

return ans;

}

};

### Merge K sorted linked lists

class Solution{

public:

class compare{

public:

bool operator()(Node\* a , Node\* b){

return a->data > b->data;

}

};

//Function to merge K sorted linked list.

Node \* mergeKLists(Node \*arr[], int K)

{

// Your code here

if(K == 0) return NULL;

priority\_queue<Node\*,vector<Node\*>,compare>minHeap;

for(int i=0;i<K;i++){

if(arr[i]!=NULL){

minHeap.push(arr[i]);

}

}

Node\* head = NULL;

Node\* tail = NULL;

while(minHeap.size() > 0){

Node\* top = minHeap.top();

minHeap.pop();

if(top->next!=NULL){

minHeap.push(top->next);

}

if(head == NULL){

head = top;

tail = top;

}

else{

tail->next = top;

tail = top;

}

}

return head;

}

};

[**232. Implement Queue using Stacks**](https://leetcode.com/problems/implement-queue-using-stacks/)

class MyQueue {

public:

    stack<int> input;

    stack<int> output;

    MyQueue() {

    }

    void push(int x) {

       while(!input.empty()) {

           output.push(input.top());

           input.pop();

       }

       input.push(x);

       while(!output.empty()){

           input.push(output.top());

           output.pop();

       }

    }

    int pop() {

        int val = input.top();

        input.pop();

        return val;

    }

    int peek() {

       return input.top();

    }

    bool empty() {

       return input.empty() && output.empty();

    }

};

Code

Testcase

Testcase

Result

[**20. Valid Parentheses**](https://leetcode.com/problems/valid-parentheses/)

class Solution {

public:

    bool isValid(string s) {

        stack<char> stk;

        for(int i=0;i<s.length();i++){

            char ch = s[i];

            if(ch == '(' || ch == '{' || ch == '['){

                stk.push(ch);

            }

            else{

                if(!stk.empty()){

                   char top = stk.top();

                   if((ch == ')' && top == '(') || (ch == '}' && top == '{') || (ch == ']' && top == '[')){

                       stk.pop();

                   }

                   else{

                       return false;

                   }

                }

                else{

                    return false;

                }

            }

        }

        if(stk.empty()) {

            return true;

        }

        return false;

    }

};

Sol2:

class Solution {

public:

    map<char,char> Pm = {

        {'}' , '{'}, {')' , '('}, {']' , '['}

    };

    bool isValid(string s) {

        stack<char> stk;

        for(const char& c : s){

            if(c == '(' || c == '{' || c == '['){

                stk.push(c);

            }

            else{

                if(!stk.empty() && (Pm.at(c) == stk.top())){

                    stk.pop();

                }

                else{

                    return false;

                }

            }

        }

        return stk.empty();

    }

};

[**155. Min Stack**](https://leetcode.com/problems/min-stack/)

class MinStack {

public:

   vector<pair<int,int>> v;

    MinStack() {

    }

    void push(int val) {

        if(v.empty()){

            v.push\_back({val,val});

        }

        else{

            v.push\_back({val,min(v.back().second,val)});

        }

    }

    void pop() {

        v.pop\_back();

    }

    int top() {

       return v.back().first;

    }

    int getMin() {

         return v.back().second;

    }

};

### Prefix to Postfix Conversion

bool isOperand(char ch){

if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z') || (ch>='0' && ch<='9'))

return 1;

else return 0;

}

string preToPost(string pre\_exp) {

stack<string>st;

for(int i=pre\_exp.size()-1; i>=0; i--){

if(isOperand(pre\_exp[i])){

st.push(string(1, pre\_exp[i]));

}else{

string A=st.top();

st.pop();

string B=st.top();

st.pop();

st.push(A+B+pre\_exp[i]);

}

}

return st.top();

}

### Prefix to Infix Conversion

// User function Template for C++

class Solution {

public:

string preToInfix(string pre\_exp) {

// Write your code here

stack<string> stk;

reverse(begin(pre\_exp),end(pre\_exp));

for(int i=0;i<pre\_exp.size();i++){

if(isalpha(pre\_exp[i])){

string curr = "";

curr += pre\_exp[i];

stk.push(curr);

}

else{

string s2 = stk.top();

stk.pop();

string s1 = stk.top();

stk.pop();

stk.push('(' + s2 + pre\_exp[i] + s1 + ')');

}

}

return stk.top();

}

};

### Postfix to Prefix Conversion

class Solution {

public:

bool isOperand(char ch){

if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9')){

return 1;

}

else return 0;

}

string postToPre(string post\_exp) {

// Write your code here

stack<string> stk;

for(int i=0;i<post\_exp.size();i++){

if(isOperand(post\_exp[i])){

stk.push(string(1,post\_exp[i]));

}

else{

string s1 = stk.top();

stk.pop();

string s2 = stk.top();

stk.pop();

stk.push(post\_exp[i] + s2 + s1);

}

}

return stk.top();

}

};

### Postfix to Infix Conversion

class Solution {

public:

bool isOperand(char ch){

if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '1' && ch <= '9')){

return true;

}

return false;

}

string postToInfix(string exp) {

// Write your code here

stack<string> stk;

for(int i=0;i<exp.size();i++){

if(isOperand(exp[i])){

stk.push(string(1,exp[i]));

}

else{

string s1 = stk.top();

stk.pop();

string s2 = stk.top();

stk.pop();

stk.push('(' + s2 + exp[i] + s1 + ')');

}

}

return stk.top();

}

};

**Convert Infix To Prefix Notation**

// C++ program to convert infix to prefix

#include <bits/stdc++.h>

using namespace std;

// Function to check if the character is an operator

bool isOperator(char c)

{

return (!isalpha(c) && !isdigit(c));

}

// Function to get the priority of operators

int getPriority(char C)

{

if (C == '-' || C == '+')

return 1;

else if (C == '\*' || C == '/')

return 2;

else if (C == '^')

return 3;

return 0;

}

// Function to convert the infix expression to postfix

string infixToPostfix(string infix)

{

infix = '(' + infix + ')';

int l = infix.size();

stack<char> char\_stack;

string output;

for (int i = 0; i < l; i++) {

// If the scanned character is an

// operand, add it to output.

if (isalpha(infix[i]) || isdigit(infix[i]))

output += infix[i];

// If the scanned character is an

// ‘(‘, push it to the stack.

else if (infix[i] == '(')

char\_stack.push('(');

// If the scanned character is an

// ‘)’, pop and output from the stack

// until an ‘(‘ is encountered.

else if (infix[i] == ')') {

while (char\_stack.top() != '(') {

output += char\_stack.top();

char\_stack.pop();

}

// Remove '(' from the stack

char\_stack.pop();

}

// Operator found

else {

if (isOperator(char\_stack.top())) {

if (infix[i] == '^') {

while (

getPriority(infix[i])

<= getPriority(char\_stack.top())) {

output += char\_stack.top();

char\_stack.pop();

}

}

else {

while (

getPriority(infix[i])

< getPriority(char\_stack.top())) {

output += char\_stack.top();

char\_stack.pop();

}

}

// Push current Operator on stack

char\_stack.push(infix[i]);

}

}

}

while (!char\_stack.empty()) {

output += char\_stack.top();

char\_stack.pop();

}

return output;

}

// Function to convert infix to prefix notation

string infixToPrefix(string infix)

{

// Reverse String and replace ( with ) and vice versa

// Get Postfix

// Reverse Postfix

int l = infix.size();

// Reverse infix

reverse(infix.begin(), infix.end());

// Replace ( with ) and vice versa

for (int i = 0; i < l; i++) {

if (infix[i] == '(') {

infix[i] = ')';

}

else if (infix[i] == ')') {

infix[i] = '(';

}

}

string prefix = infixToPostfix(infix);

// Reverse postfix

reverse(prefix.begin(), prefix.end());

return prefix;

}

// Driver code

int main()

{

string s = ("x+y\*z/w+u");

// Function call

cout << infixToPrefix(s) << std::endl;

return 0;

}

### Next Greater Element

### class Solution

### {

### public:

### //Function to find the next greater element for each element of the array.

### vector<long long> nextLargerElement(vector<long long> arr, int n){

### // Your code here

### stack<long long > s;

### vector<long long> ans;

### 

### for(int i=n-1;i>=0;i--){

### while(!s.empty() && arr[i] >= s.top()){

### s.pop();

### }

### 

### if(s.empty()){

### ans.push\_back(-1);

### }

### else{

### ans.push\_back(s.top());

### }

### 

### s.push(arr[i]);

### }

### 

### reverse(ans.begin(),ans.end());

### 

### return ans;

### }

### };

**Leetcode solution**

class Solution {

public:

    vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {

        vector<int> res (nums1.size(),-1);

        stack<int> stk;

        unordered\_map<int,int> next\_greater\_map;

        for(int i=0;i<nums2.size();i++){

            int ele = nums2[i];

            while(!stk.empty() && ele > stk.top()){

                next\_greater\_map[stk.top()] = ele;

                stk.pop();

            }

            stk.push(ele);

        }

        for(int i=0;i<nums1.size();i++){

            int ele = nums1[i];

            if(next\_greater\_map.find(ele) != next\_greater\_map.end()){

                int nge = next\_greater\_map[ele];

                res[i] = nge;

            }

        }

        return res;

    }

};

### Sort a stack

void SortedStack :: sort()

{

//Your code here

priority\_queue<int,vector<int>,greater<int>> q;

while(!s.empty()){

q.push(s.top());

s.pop();

}

while(!q.empty()){

s.push(q.top());

q.pop();

}

}

[**1475. Final Prices With a Special Discount in a Shop**](https://leetcode.com/problems/final-prices-with-a-special-discount-in-a-shop/)

class Solution {

public:

    vector<int> finalPrices(vector<int>& prices) {

       vector<int> final;

       int i = 0, j = 1;

       while(i < prices.size() -1 && j < prices.size()){

           if(prices[i] >= prices[j]){

               prices[i] = prices[i] - prices[j];

               i++;

               j = i + 1;

           }

           else{

               j++;

           }

           if(j == prices.size()) {

               i++;

               j = i + 1;

           }

       }

       return prices;

    }

};

# [Largest Rectangle In Histogram](https://leetcode.com/problems/largest-rectangle-in-histogram/)

Given array of heights, return area of largest rectangle

Ex. heights = [2,1,5,6,2,3] -> 10 (5 x 2 at index 2 and 3)

Monotonic incr stack, if curr height lower extend back, find max area

Time: O(n)

Space: O(n)

\*/

class Solution {

public:

int largestRectangleArea(vector<int>& heights) {

// pair: [index, height]

stack<pair<int, int>> stk;

int result = 0;

for (int i = 0; i < heights.size(); i++) {

int start = i;

while (!stk.empty() && stk.top().second > heights[i]) {

int index = stk.top().first;

int width = i - index;

int height = stk.top().second;

stk.pop();

result = max(result, height \* width);

start = index;

}

stk.push({start, heights[i]});

}

while (!stk.empty()) {

int width = heights.size() - stk.top().first;

int height = stk.top().second;

stk.pop();

result = max(result, height \* width);

}

return result;

}

};

[**239. Sliding Window Maximum**](https://leetcode.com/problems/sliding-window-maximum/)

class Solution {

public:

    vector<int> maxSlidingWindow(vector<int>& nums, int k) {

       deque<int> q;

       vector<int> result;

       int i = 0;

       int j = 0;

       while(j < nums.size()){

           while(!q.empty() && nums[q.back()] < nums[j]){

               q.pop\_back();

           }

           q.push\_back(j);

           if(i > q.front()){

               q.pop\_front();

           }

           if(j + 1 >= k){

               result.push\_back(nums[q.front()]);

               i++;

           }

           j++;

       }

       return result;

    }

};

[Rotting Oranges](https://leetcode.com/problems/rotting-oranges/)

/\*

Given grid: 0 empty cell, 1 fresh orange, 2 rotten orange

Return min # of minutes until no cell has a fresh orange

BFS: rotten will contaminate neighbors first, then propagate out

Time: O(m x n)

Space: O(m x n)

\*/

class Solution {

public:

int orangesRotting(vector<vector<int>>& grid) {

int m = grid.size();

int n = grid[0].size();

// build initial set of rotten oranges

queue<pair<int, int>> q;

int fresh = 0;

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

if (grid[i][j] == 2) {

q.push({i, j});

} else if (grid[i][j] == 1) {

fresh++;

}

}

}

// mark the start of a minute

q.push({-1, -1});

int result = -1;

// start rotting process via BFS

while (!q.empty()) {

int row = q.front().first;

int col = q.front().second;

q.pop();

if (row == -1) {

// finish 1 minute of processing, mark next minute

result++;

if (!q.empty()) {

q.push({-1, -1});

}

} else {

// rotten orange, contaminate its neighbors

for (int i = 0; i < dirs.size(); i++) {

int x = row + dirs[i][0];

int y = col + dirs[i][1];

if (x < 0 || x >= m || y < 0 || y >= n) {

continue;

}

if (grid[x][y] == 1) {

// contaminate

grid[x][y] = 2;

fresh--;

// this orange will now contaminate others

q.push({x, y});

}

}

}

}

if (fresh == 0) {

return result;

}

return -1;

}

private:

vector<vector<int>> dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

};

[**901. Online Stock Span**](https://leetcode.com/problems/online-stock-span/)

class StockSpanner {

public:

   stack<pair<int, int>> st;

   pair<int,int> pr;

    StockSpanner() {

       pr = {0,0};

    }

    int next(int price) {

        int ret = 1;

        while(!st.empty() && price >= pr.first){

            ret += pr.second;

            st.pop();

            if(!st.empty()){

                pr = st.top();

            }

        }

        st.push(make\_pair(price,ret));

        pr = st.top();

        return (ret);

    }

};

# Binary tree post order traversal

class Solution {

public:

    vector<int> postorderTraversal(TreeNode\* root) {

       vector<int> postOrder;

       if(root == NULL) return postOrder;

       stack<TreeNode\*> s1,s2;

       s1.push(root);

       while(!s1.empty()){

           TreeNode\* curr = s1.top();

           s1.pop();

           s2.push(curr);

           if(curr->left != NULL){

               s1.push(curr->left);

           }

           if(curr->right != NULL){

               s1.push(curr->right);

           }

       }

       while(!s2.empty()){

           postOrder.push\_back(s2.top()->val);

           s2.pop();

       }

       return postOrder;

    }

};

# [Morris Inorder Traversal](https://takeuforward.org/data-structure/morris-inorder-traversal-of-a-binary-tree/)

class Solution {

public:

    vector<int> inorderTraversal(TreeNode\* root) {

        vector<int> ans;

        TreeNode\* temp;

        while(root){

            if(root->left){

                temp = root->left;

                while(temp->right){

                    temp = temp->right;

                }

                temp->right = root;

                temp = root->left;

                root->left = NULL;

                root = temp;

            }

            else{

                ans.push\_back(root->val);

                root = root->right;

            }

        }

        return ans;

    }

};

# Left View of Binary Tree

## void checkView(Node\* root,vector<int> &ans, int level){

## if(root == NULL) return;

## 

## if(level == ans.size()) ans.push\_back(root->data);

## 

## checkView(root->left,ans,level+1);

## checkView(root->right,ans,level+1);

## }

## //Function to return a list containing elements of left view of the binary tree.

## vector<int> leftView(Node \*root)

## {

## // Your code here

## vector<int> ans;

## checkView(root,ans,0);

## return ans;

## }

Iterative approach

vector<int> leftView(Node \*root)

{

// Your code here

vector<int> res;

if(root == NULL) return res;

queue<Node\*> q;

q.push(root);

while(!q.empty()){

int size = q.size();

for(int i=0;i<size;i++){

Node \*curr = q.front();

q.pop();

if(i == 0) res.push\_back(curr->data);

if(curr->left != NULL) {

q.push(curr->left);

}

if(curr->right != NULL){

q.push(curr->right);

}

}

}

return res;

}

# Bottom View of Binary Tree

class Solution {

public:

vector <int> bottomView(Node \*root) {

// Your Code Here

vector<int> ans;

if(root == NULL)

return ans;

map<int, int> topNode;

queue<pair<Node \*, int>>q;

q.push(make\_pair(root, 0));

while(!q.empty())

{

pair<Node \*, int> temp = q.front();

q.pop();

Node \*frontNode = temp.first;

int line\_no = temp.second;

topNode[line\_no] = frontNode -> data;

if(frontNode -> left)

q.push(make\_pair(frontNode -> left, line\_no - 1));

if(frontNode -> right)

q.push(make\_pair(frontNode -> right, line\_no + 1));

}

for(auto i : topNode)

ans.push\_back(i.second);

return ans;

}

};

# Top View of Binary Tree

Recursion

class Solution

{

public:

void checkLevels(map<int,pair<int,int>> &mapper,Node\* root,int line\_no,int level){

if(root == NULL) return;

if(mapper.find(line\_no) == mapper.end() || mapper[line\_no].second > level){

mapper[line\_no] = {root->data,level};

}

checkLevels(mapper,root->left,line\_no-1,level+1);

checkLevels(mapper,root->right,line\_no+1,level+1);

}

//Function to return a list of nodes visible from the top view

//from left to right in Binary Tree.

vector<int> topView(Node \*root)

{

//Your code here

vector<int> ans;

if(root == NULL) return ans;

map<int,pair<int,int>> mapper;//data,{line\_no,level}

checkLevels(mapper,root,0,0);

for(auto it : mapper){

ans.push\_back(it.second.first);

}

return ans;

}

};

Iterative

class Solution

{

public:

//Function to return a list of nodes visible from the top view

//from left to right in Binary Tree.

vector<int> topView(Node \*root)

{

//Your code here

vector<int> ans;

if(root == NULL)

return ans;

map<int, int> topNode;

queue<pair<Node \*, int>>q;

q.push(make\_pair(root, 0));

while(!q.empty())

{

pair<Node \*, int> temp = q.front();

q.pop();

Node \*frontNode = temp.first;

int line\_no = temp.second;

if(topNode.find(line\_no) == topNode.end()){

topNode[line\_no] = frontNode -> data;

}

if(frontNode -> left)

q.push(make\_pair(frontNode -> left, line\_no - 1));

if(frontNode -> right)

q.push(make\_pair(frontNode -> right, line\_no + 1));

}

for(auto i : topNode)

ans.push\_back(i.second);

return ans;

}

};

[**987. Vertical Order Traversal of a Binary Tree**](https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/)

class Solution {

public:

    vector<vector<int>> verticalTraversal(TreeNode\* root) {

        map<int,map<int,multiset<int>>> nodes;

        queue<pair<TreeNode\*,pair<int,int>>> levels;

        levels.push({root,{0,0}});

        while(!levels.empty()){

            auto node = levels.front();

            levels.pop();

            TreeNode\* curr = node.first;

            int x = node.second.first;

            int y = node.second.second;

            nodes[x][y].insert(curr->val);

            if(curr->left){

                levels.push({curr->left,{x-1,y+1}});

            }

            if(curr->right){

                levels.push({curr->right,{x+1,y+1}});

            }

        }

         vector < vector < int >> ans;

         for (auto p: nodes) {

           vector < int > col;

           for (auto q: p.second) {

            col.insert(col.end(), q.second.begin(), q.second.end());

            }

            ans.push\_back(col);

         }

         return ans;

    }

};

[**257. Binary Tree Paths**](https://leetcode.com/problems/binary-tree-paths/)

class Solution {

    vector<string> res;

public:

    void recur(TreeNode\* root,string s){

        if(root == NULL) return;

        if(s.empty()){

            s+=(to\_string(root->val));

        }

        else{

            s = s + "->" + (to\_string(root->val));

        }

        if(root->left == NULL && root->right == NULL){

            res.push\_back(s);

        }

        recur(root->left,s);

        recur(root->right,s);

    }

    vector<string> binaryTreePaths(TreeNode\* root) {

      recur(root,"");

      return res;

    }

};

# Maximum width of Binary Tree

class Solution {

public:

    int widthOfBinaryTree(TreeNode\* root) {

         if (!root) return 0;

         int ans = 0;

         queue < pair < TreeNode\* , int >> q;

         q.push({root,0});

        while (!q.empty()) {

          int size = q.size();

          int curMin = q.front().second;

          int leftMost, rightMost;

          for (int i = 0; i < size; i++) {

             int cur\_id = q.front().second - curMin; // subtracted to prevent integer overflow

             TreeNode \* temp = q.front().first;

             q.pop();

             if (i == 0) leftMost = cur\_id;

             if (i == size - 1) rightMost = cur\_id;

             if (temp -> left)

             q.push({ temp -> left,(long long) cur\_id \* 2 + 1});

             if (temp -> right)

             q.push({temp -> right,(long long) cur\_id \* 2 + 2});

          }

          ans = max(ans, rightMost - leftMost + 1);

        }

        return ans;

    }

};

# Level order Traversal of binary tree

class Solution {

public:

    vector<vector<int>> levelOrder(TreeNode\* root) {

        vector<vector<int>> ans;

        if(root == NULL) return ans;

        queue<TreeNode\*> q;

        q.push(root);

        while(!q.empty()){

            int size = q.size();

            vector<int> level;

            while(size--){

                TreeNode\* curr = q.front();

                q.pop();

                level.push\_back(curr->val);

                if(curr->left){

                    q.push(curr->left);

                }

                if(curr->right){

                    q.push(curr->right);

                }

            }

            ans.push\_back(level);

        }

        return ans;

    }

};

## Recursive solution

class Solution {

public:

  void helper(TreeNode\* root, vector<vector<int>> &ans,int level){

      if(root == NULL) return;

      if(level >= ans.size()) ans.push\_back({});

      ans[level].push\_back(root->val);

      helper(root->left,ans,level + 1);

      helper(root->right,ans,level + 1);

  }

    vector<vector<int>> levelOrder(TreeNode\* root) {

        vector<vector<int>> ans;

        if(root == NULL) return ans;

         helper(root,ans,0);

        return ans;

    }

};

# Maximum depth of binary tree

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if(root == NULL) return 0;

        int leftHeight = maxDepth(root->left);

        int rightHeight = maxDepth(root->right);

        return max(leftHeight,rightHeight) + 1;

    }

};

# Diameter of Binary Tree

class Solution {

public:

    int diameterOfBinaryTree(TreeNode\* root) {

      int res = 0;

      dfs(root,res);

      return res;

    }

    int dfs(TreeNode\* root, int& res){

        if(root == NULL) return 0;

        int left = dfs(root->left,res);

        int right = dfs(root->right,res);

        res = max(res,left + right);

        return 1 + max(left, right);

    }

};

# [110. Balanced Binary Tree](https://leetcode.com/problems/balanced-binary-tree/)

class Solution {

public:

    bool isBalanced(TreeNode\* root) {

       return dfsHeight(root) != -1;

    }

    int dfsHeight(TreeNode\* root){

        if(root == NULL) return 0;

        int lh = dfsHeight(root->left);

        if(lh == -1) return -1;

        int rh = dfsHeight(root->right);

        if(rh == -1) return -1;

        if(abs(lh - rh) > 1) return -1;

        return max(lh,rh) + 1;

    }

};

# [236. Lowest Common Ancestor of a Binary Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if(root == NULL || root == p || root == q){

            return root;

        }

        TreeNode\* left = lowestCommonAncestor(root->left,p,q);

        TreeNode\* right = lowestCommonAncestor(root->right,p,q);

        if(left == NULL){

            return right;

        }

        else if(right == NULL){

            return left;

        }

        else{

            return root;

        }

    }

};

[**100. Same Tree**](https://leetcode.com/problems/same-tree/)

class Solution {

public:

    bool isSameTree(TreeNode\* p, TreeNode\* q) {

        if(p == NULL && q == NULL) return true;

        if(p == NULL || q == NULL) return false;

        if(p->val != q->val) return false;

        return isSameTree(p->right,q->right) && isSameTree(p->left,q->left);

    }

};

class Solution {

public:

    bool isSameTree(TreeNode\* p, TreeNode\* q) {

        queue<TreeNode\*> Q;

        Q.push(p);

        Q.push(q);

        while(Q.size() != 0){

            TreeNode\* p1 = Q.front();

            Q.pop();

            TreeNode\* q1 = Q.front();

            Q.pop();

            if(p1 == NULL && q1 == NULL) continue;

            if(p1 == NULL || q1 == NULL) return false;

            if(p1->val != q1->val) return false;

            Q.push(p1->left);

            Q.push(q1->left);

            Q.push(p1->right);

            Q.push(q1->right);

        }

        return true;

    }

};

[**103. Binary Tree Zigzag Level Order Traversal**](https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/)

class Solution {

public:

    vector<vector<int>> zigzagLevelOrder(TreeNode\* root) {

       vector<vector<int>> res;

       if(root == NULL) return res;

       queue<TreeNode\*> q;

       q.push(root);

       while(!q.empty()){

          int size = q.size();

          vector<int> level;

          for(int i=0;i<size;i++){

              TreeNode\* node = q.front();

              q.pop();

              level.push\_back(node->val);

              if(node->left){

                  q.push(node->left);

              }

              if(node->right){

                  q.push(node->right);

              }

          }

          if(res.size() % 2 == 1){

              reverse(level.begin(),level.end());

          }

          res.push\_back(level);

       }

       return res;

    }

};

**Boundary Traversal of binary tree**

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

bool isLeaf(TreeNode\* root) {

return !root->left && !root->right;

}

void addLeaves(TreeNode\* root, vector<int>& res) {

if (isLeaf(root)) {

res.push\_back(root->val);

return;

}

if (root->left) addLeaves(root->left, res);

if (root->right) addLeaves(root->right, res);

}

public:

vector<int> boundaryOfBinaryTree(TreeNode\* root) {

if (!root) return {};

vector<int> res;

if (!isLeaf(root)) res.push\_back(root->val);

// add left boundary

TreeNode\* cur = root->left;

while (cur) {

if (!isLeaf(cur)) res.push\_back(cur->val);

if (cur->left) cur = cur->left;

else cur = cur->right;

}

// add leaf nodes

addLeaves(root, res);

cur = root->right;

vector<int> tmp;

// add right boundary

while (cur) {

if (!isLeaf(cur)) tmp.push\_back(cur->val);

if (cur->right) cur = cur->right;

else cur = cur->left;

}

for (int i = tmp.size()-1; i >= 0; --i) {

res.push\_back(tmp[i]);

}

return res;

}

};

[**124. Binary Tree Maximum Path Sum**](https://leetcode.com/problems/binary-tree-maximum-path-sum/)

class Solution {

public:

    int maxPathSum(TreeNode\* root) {

      int maxPath = INT\_MIN;

      dfs(root,maxPath);

      return maxPath;

    }

    int dfs(TreeNode\* root,int& maxPath){

        if(root == NULL) return 0;

        int left = max(dfs(root->left,maxPath),0);

        int right = max(dfs(root->right,maxPath),0);

        int currPath = root->val + left + right;

        maxPath = max(maxPath,currPath);

        return root->val + max(left,right);

    }

};

[**105. Construct Binary Tree from Preorder and Inorder Traversal**](https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

class Solution {

public:

    TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

        int index = 0;

        return build(preorder,inorder,index,0,inorder.size()-1);

    }

    TreeNode\* build(vector<int>& preorder, vector<int>& inorder, int& index, int i, int j){

        if(i > j) return NULL;

        TreeNode\* root = new TreeNode(preorder[index]);

        int mid = 0;

        for(int i=0;i<inorder.size();i++){

            if(preorder[index] == inorder[i]){

                mid = i;

                break;

            }

        }

        index++;

        root->left = build(preorder,inorder,index,i,mid-1);

        root->right = build(preorder,inorder,index,mid+1,j);

        return root;

    }

};

[**106. Construct Binary Tree from Inorder and Postorder Traversal**](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/)

class Solution {

     TreeNode\* buildTree\_rec(vector<int>& inorder,int i1, int i2, vector<int>& postorder, int p1, int p2) {

         if(i1 >= i2 || p1 >= p2) return NULL;

         TreeNode\* root = new TreeNode(postorder[p2-1]);

         auto it = find(inorder.begin() + i1 , inorder.begin() + i2, postorder[p2-1]);

         int diff = it - inorder.begin() - i1;

         root->left = buildTree\_rec(inorder, i1, i1 + diff, postorder, p1, p1 + diff);

         root->right = buildTree\_rec(inorder, i1 + diff + 1, i2, postorder, p1 + diff, p2-1);

         return root;

    }

public:

    TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {

        int n = inorder.size();

        if(n == 0) return NULL;

       return buildTree\_rec(inorder, 0, n, postorder, 0, n);

    }

};

[**101. Symmetric Tree**](https://leetcode.com/problems/symmetric-tree/)

class Solution {

    bool isSameTree(TreeNode\* leftTree, TreeNode\* rightTree){

       if(leftTree == NULL && rightTree == NULL) return true;

       if(leftTree == NULL || rightTree == NULL) return false;

       if(leftTree->val != rightTree->val) return false;

       return  isSameTree(leftTree->right,rightTree->left) && isSameTree(leftTree->left,rightTree->right);

    }

public:

    bool isSymmetric(TreeNode\* root) {

       return isSameTree(root->left, root->right);

    }

};

[**114. Flatten Binary Tree to Linked List**](https://leetcode.com/problems/flatten-binary-tree-to-linked-list/)

class Solution {

public:

    void flatten(TreeNode\* root) {

        TreeNode\* curr = root;

        while(curr!= NULL){

            if(curr->left != NULL){

                TreeNode\* temp = curr->left;

                while(temp->right){

                    temp = temp->right;

                }

                temp->right = curr->right;

                curr->right = curr->left;

                curr->left = NULL;

            }

            curr = curr->right;

        }

    }

};

# Mirror Tree

class Solution {

void traverse(Node\* node){

if(node == NULL) return;

if(node->left){

traverse(node->left);

}

if(node->right){

traverse(node->right);

}

swap(node->left, node->right);

}

public:

// Function to convert a binary tree into its mirror tree.

void mirror(Node\* node) {

// code here

traverse(node);

}

};

[**116. Populating Next Right Pointers in Each Node**](https://leetcode.com/problems/populating-next-right-pointers-in-each-node/)

class Solution {

public:

    Node\* connect(Node\* root) {

         if(root==NULL){

            return NULL;

        }

        Node\* current=root;

        while(current->left!=NULL){

            Node\* temp=current;

            while(current!=NULL){

                current->left->next=current->right;

                current->right->next=current->next==NULL?NULL: current->next->left;

                current=current->next;

            }

            current=temp->left;

        }

        return root;

    }

};

[**700. Search in a Binary Search Tree**](https://leetcode.com/problems/search-in-a-binary-search-tree/)

class Solution {

public:

    TreeNode\* searchBST(TreeNode\* root, int val) {

       if(root == NULL) return root;

       if(root->val == val) return root;

       else if(root->val > val) return searchBST(root->left,val);

       else return searchBST(root->right,val);

    }

};

[**108. Convert Sorted Array to Binary Search Tree**](https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/)

class Solution {

    TreeNode\* createBST(vector<int> &nums,int left,int right){

        if(left > right) return NULL;

        int mid = left + (right-left)/2;

        TreeNode\* root = new TreeNode(nums[mid]);

        root->left = createBST(nums,left,mid-1);

        root->right = createBST(nums,mid+1,right);

        return root;

    }

public:

    TreeNode\* sortedArrayToBST(vector<int>& nums) {

        return createBST(nums,0,nums.size()-1);

    }

};

[**1008. Construct Binary Search Tree from Preorder Traversal**](https://leetcode.com/problems/construct-binary-search-tree-from-preorder-traversal/)

class Solution {

    TreeNode\* build(vector<int> preorder,int& i, int bound){

        if(i == preorder.size() || preorder[i] > bound) return NULL;

        TreeNode\* root = new TreeNode(preorder[i++]);

        root->left = build(preorder,i,root->val);

        root->right = build(preorder,i,bound);

        return root;

    }

public:

    TreeNode\* bstFromPreorder(vector<int>& preorder) {

        int i = 0;

        return build(preorder,i,INT\_MAX);

    }

};

[**98. Validate Binary Search Tree**](https://leetcode.com/problems/validate-binary-search-tree/)

class Solution {

    bool helper(TreeNode\* root, long long minVal , long long maxVal){

        if(root == NULL) return true;

        if(root->val >= maxVal || root->val <= minVal) return false;

        return helper(root->left,minVal,root->val) && helper(root->right,root->val,maxVal);

    }

public:

    bool isValidBST(TreeNode\* root) {

        return helper(root,LONG\_MIN,LONG\_MAX);

    }

};

[**235. Lowest Common Ancestor of a Binary Search Tree**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/)

class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

      if(root == NULL || root->val == p->val || root->val == q->val){

          return root;

      }

      TreeNode \*left = NULL, \*right = NULL;

      if(root->val > p->val && root->val > q->val){

          left = lowestCommonAncestor(root->left,p,q);

      }

      else if(root->val < p->val && root->val < q->val){

          right = lowestCommonAncestor(root->right,p,q);

      }

      else{

          left = lowestCommonAncestor(root->left,p,q);

          right = lowestCommonAncestor(root->right,p,q);

      }

      if(left == NULL && right == NULL) return NULL;

      if(left == NULL && right!=NULL) return right;

      if(left!=NULL && right == NULL) return left;

      if(left!=NULL && right!=NULL) return root;

      return root;

    }

};

# Floor in BST

class Solution{

void inorder(Node\* root, Node\* ans, int x){

if(root == NULL){

return;

}

inorder(root->left,ans,x);

if(root->data <= x){

ans->data = root->data;

}

inorder(root->right,ans,x);

}

public:

int floor(Node\* root, int x) {

// Code here

Node\* ans = new Node(-1);

inorder(root,ans,x);

return ans->data;

}

};

## Iterative

class Solution{

public:

int floor(Node\* root, int x) {

// Code here

int ans=-1;

while(root!=NULL){

if(root->data == x){

ans = root->data;

break;

}else if(root->data > x){

root = root->left;

}else{

ans = root->data;

root = root->right;

}

}

return ans;

}

};

# Ceil in BST

int findCeil(Node\* root, int input) {

if (root == NULL) return -1;

// Your code here

int ans = -1;

while(root != NULL){

if(root->data == input){

ans = root->data;

break;

}

else if(root->data < input){

root = root->right;

}

else{

ans = root->data;

root = root->left;

}

}

return ans;

}

[**230. Kth Smallest Element in a BST**](https://leetcode.com/problems/kth-smallest-element-in-a-bst/)

class Solution {

    void inorder(TreeNode\* root, vector<int> &inArr){

        TreeNode\* temp;

        while(root){

            if(root->left){

                temp = root->left;

                while(temp->right){

                    temp = temp->right;

                }

                temp->right = root;

                temp = root->left;

                root->left = NULL;

                root = temp;

            }

            else{

                inArr.push\_back(root->val);

                root = root->right;

            }

        }

    }

public:

    int kthSmallest(TreeNode\* root, int k) {

        vector<int> inArr;

        inorder(root,inArr);

        return inArr[k-1];

    }

};

# Kth largest element in BST

# class Solution

# {

# void inorder(Node\* root, vector<int> &inArr){

# Node\* temp;

# while(root){

# if(root->left){

# temp = root->left;

# while(temp->right){

# temp = temp->right;

# }

# temp->right = root;

# temp = root->left;

# root->left = NULL;

# root = temp;

# }

# else{

# inArr.push\_back(root->data);

# root = root->right;

# }

# }

# }

# public:

# int kthLargest(Node \*root, int K)

# {

# //Your code here

# vector<int> inArr;

# inorder(root,inArr);

# int n = inArr.size();

# return inArr[n-K];

# }

# };

# [653. Two Sum IV - Input is a BST](https://leetcode.com/problems/two-sum-iv-input-is-a-bst/)

class Solution {

public:

    unordered\_set<int> s;

    bool findTarget(TreeNode\* root, int k) {

        if(root == NULL) return false;

        if(s.count(k - root->val)) return true;

        s.insert(root->val);

        return findTarget(root->left,k) || findTarget(root->right,k);

    }

};

# [173. Binary Search Tree Iterator](https://leetcode.com/problems/binary-search-tree-iterator/)

class BSTIterator {

    stack<TreeNode\*> stk;

    void pushAll(TreeNode\* node){

        while(node!= NULL){

            stk.push(node);

            node = node->left;

        }

    }

public:

    BSTIterator(TreeNode\* root) {

        pushAll(root);

    }

    int next() {

        TreeNode\* temp = stk.top();

        stk.pop();

        pushAll(temp->right);

        return temp->val;

    }

    bool hasNext() {

       return !stk.empty();

    }

};

# Size of the largest BST in a Binary Tree

class NodeData{

  private:

  int mini;

  int maxi;

  int sum;

  int maxSum;

  public:

  NodeData(int mini, int maxi, int sum, int maxSum){

      this->mini = mini;

      this->maxi = maxi;

      this->sum = sum;

      this->maxSum = maxSum;

  }

  int getsum(){

      return sum;

  }

  int getmaxi(){

      return maxi;

  }

  int getmini(){

      return mini;

  }

  int getmaxSum(){

      return maxSum;

  }

 };

class Solution {

public:

    NodeData maxBST(TreeNode\* root){

        if(!root) return NodeData(INT\_MAX, INT\_MIN, 0, INT\_MIN);

        NodeData left = maxBST(root->left);

        NodeData right = maxBST(root->right);

        if(left.getmaxi() < root->val && right.getmini() > root->val){

            return NodeData(min(left.getmini(), root->val), max(right.getmaxi(), root->val), root->val + left.getsum() + right.getsum(), max(max(left.getmaxSum(), right.getmaxSum()),root->val + left.getsum() + right.getsum()));

        }

        return NodeData(INT\_MIN, INT\_MAX, max(left.getsum(),right.getsum()), max(left.getmaxSum(),right.getmaxSum()));

    }

    int maxSumBST(TreeNode\* root) {

        int ans = maxBST(root).getmaxSum();

        return ans >= 0 ? ans : 0;

    }

};

# Serialize and Deserialize a Binary Tree

class Codec {

public:

    // Encodes a tree to a single string.

    string serialize(TreeNode\* root) {

        ostringstream out;

        encode(root, out);

        return out.str();

    }

    // Decodes your encoded data to tree.

    TreeNode\* deserialize(string data) {

        istringstream in(data);

        return decode(in);

    }

private:

    void encode(TreeNode\* root, ostringstream& out) {

        if (root == NULL) {

            out << "N ";

            return;

        }

        out << root->val << " ";

        encode(root->left, out);

        encode(root->right, out);

    }

    TreeNode\* decode(istringstream& in) {

        string value = "";

        in >> value;

        if (value == "N") {

            return NULL;

        }

        TreeNode\* root = new TreeNode(stoi(value));

        root->left = decode(in);

        root->right = decode(in);

        return root;

    }

};

# BFS of graph

class Solution {

public:

// Function to return Breadth First Traversal of given graph.

vector<int> bfsOfGraph(int V, vector<int> adj[]) {

// Code here

int vis[V] = {0};

vis[0] = 1;

queue<int> q;

q.push(0);

vector<int> bfs;

while(!q.empty()){

int node = q.front();

q.pop();

bfs.push\_back(node);

for(auto it : adj[node]){

if(!vis[it]){

vis[it] = 1;

q.push(it);

}

}

}

return bfs;

}

};

# DFS of Graph

class Solution {

private:

void dfs(int node, vector<int> adj[], int vis[], vector<int> &ls ){

vis[node] = 1;

ls.push\_back(node);

//traverse all its neighbours

for(auto it : adj[node]){

if(!vis[it]){

dfs(it,adj,vis,ls);

}

}

}

public:

// Function to return a list containing the DFS traversal of the graph.

vector<int> dfsOfGraph(int V, vector<int> adj[]) {

// Code here

int vis[V] = {0};

int start = 0;

vector<int> ls;

dfs(start,adj,vis,ls);

return ls;

}

};

# Number of Provinces

class Solution {

private:

void dfs(int node,vector<int> adjList[], int vis[]){

vis[node] = 1;

for(auto it: adjList[node]) {

if(!vis[it]) {

dfs(it, adjList, vis);

}

}

}

public:

int numProvinces(vector<vector<int>> adj, int V) {

vector<int> adjList[V];

// code here

for(int i=0;i<V;i++){

for(int j=0;j<V;j++){

if(adj[i][j] == 1 && i !=j){

adjList[i].push\_back(j);

adjList[j].push\_back(i);

}

}

}

int vis[V] = {0};

int provinces = 0;

for(int i=0;i<V;i++){

if(vis[i] == 0){

provinces++;

dfs(i,adjList,vis);

}

}

return provinces;

}

};

# Find the number of islands

class Solution {

private:

void bfs(int row,int col, vector<vector<int>> &vis,vector<vector<char>> &grid ){

queue<pair<int,int>> q;

int n = grid.size();

int m = grid[0].size();

vis[row][col] = 1;

q.push({row,col});

while(!q.empty()){

int row = q.front().first;

int col = q.front().second;

q.pop();

//traverse the neightbours

for(int delrow = -1; delrow <= 1; delrow++){

for(int delcol = -1; delcol <= 1; delcol++){

int nrow = row + delrow;

int ncol = col + delcol;

if(nrow >=0 && nrow < n && ncol >= 0 && ncol < m && grid[nrow][ncol] == '1' && !vis[nrow][ncol]){

vis[nrow][ncol] = 1;

q.push({nrow,ncol});

}

}

}

}

}

public:

// Function to find the number of islands.

int numIslands(vector<vector<char>>& grid) {

// Code here

int n = grid.size();

int m = grid[0].size();

int islands = 0;

vector<vector<int>> vis(n, vector<int> (m,0));

for(int row = 0; row < n; row++){

for(int col = 0; col < m; col++){

if(!vis[row][col] && grid[row][col] == '1'){

bfs(row,col,vis,grid);

islands++;

}

}

}

return islands;

}

};

# Flood fill Algorithm

class Solution {

private:

void dfs(int row,int col, vector<vector<int>> &ans, vector<vector<int>> &image, int newColor, int delRow[],int delCol[], int iniColor){

int n = image.size();

int m = image[0].size();

ans[row][col] = newColor;

for(int i=0;i<4;i++){

int nrow = row + delRow[i];

int ncol = col + delCol[i];

if(nrow >= 0 && nrow < n && ncol >= 0 && ncol < m && image[nrow][ncol] == iniColor && ans[nrow][ncol] != newColor){

dfs(nrow,ncol,ans,image,newColor,delRow, delCol,iniColor);

}

}

}

public:

vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {

// Code here

int iniColor = image[sr][sc];

vector<vector<int>> ans = image;

int delRow[] = {-1,0,+1,0};

int delCol[] = {0,+1,0,-1};

dfs(sr,sc,ans,image,newColor, delRow, delCol,iniColor);

return ans;

}

};

# Rotten Oranges

class Solution

{

public:

//Function to find minimum time required to rot all oranges.

int orangesRotting(vector<vector<int>>& grid) {

// Code here

queue<pair<pair<int,int>,int>>q;

int n=grid.size();

int m=grid.front().size();

int i=0,j=0;

for(i=0;i<n;i++)

{

for(j=0;j<m;j++)

{

if(grid[i][j]==2)

{

q.push({{i,j},0});

}

}

}

int time=0,nrow=0,ncol=0;

i=0;

int r[4]={-1,0,1,0};

int c[4]={0,1,0,-1};

while(q.size()>0)

{

for(i=0;i<4;i++)

{

nrow=q.front().first.first+r[i];

ncol=q.front().first.second+c[i];

if(nrow>=0 && nrow<n && ncol>=0 && ncol<m && grid[nrow][ncol]==1)

{

grid[nrow][ncol]=2;

time=q.front().second;

time++;

q.push({{nrow,ncol},time});

}

}

q.pop();

}

i=0,j=0;

for(i=0;i<n;i++)

{

for(j=0;j<m;j++)

{

if(grid[i][j]==1)

{

return -1;

}

}

}

return time;

}

};

# Detect cycle in an undirected graph

class Solution {

private:

bool detect(int src,vector<int> adj[], int vis[]){

vis[src] = 1;

queue<pair<int,int>> q;

q.push({src,-1});

while(!q.empty()){

int node = q.front().first;

int parent = q.front().second;

q.pop();

for(auto adjNode : adj[node]){

if(!vis[adjNode]){

vis[adjNode] = 1;

q.push({adjNode,node});

}

else if(parent != adjNode){

return true;

}

}

}

return false;

}

public:

// Function to detect cycle in an undirected graph.

bool isCycle(int V, vector<int> adj[]) {

// Code here

int vis[V] = {0};

for(int i=0;i<V;i++){

if(!vis[i]){

if(detect(i,adj,vis)) return true;

}

}

return false;

}

};

## DFS SOLUTION

class Solution {

private:

bool dfs(int node,int parent,int vis[], vector<int> adj[]){

vis[node] = 1;

for(auto adjNode : adj[node]){

if(!vis[adjNode]){

if(dfs(adjNode,node, vis, adj) == true) return true;

}

else if(adjNode != parent) return true;

}

return false;

}

public:

// Function to detect cycle in an undirected graph.

bool isCycle(int V, vector<int> adj[]) {

// Code here

int vis[V] = {0};

for(int i=0;i<V;i++){

if(!vis[i]){

if(dfs(i,-1,vis,adj) == true) return true;

}

}

return false;

}

};

# Distance of nearest cell having 1

class Solution

{

public:

//Function to find distance of nearest 1 in the grid for each cell.

vector<vector<int>>nearest(vector<vector<int>>grid)

{

// Code here

int n = grid.size();

int m = grid[0].size();

vector<vector<int>> vis(n,vector<int>(m,0));

vector<vector<int>> dist(n,vector<int>(m,0));

queue<pair<pair<int,int>,int>> q;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(grid[i][j] == 1){

q.push({{i,j},0});

vis[i][j] = 1;

}

else{

vis[i][j] = 0;

}

}

}

int delRow[] = {-1,0,+1,0};

int delCol[] = {0,+1,0,-1};

while(!q.empty()){

int row = q.front().first.first;

int col = q.front().first.second;

int steps = q.front().second;

q.pop();

dist[row][col] = steps;

for(int i=0;i<4;i++){

int nrow = row + delRow[i];

int ncol = col + delCol[i];

if(nrow >= 0 && nrow < n && ncol >= 0 && ncol < m && vis[nrow][ncol] == 0){

vis[nrow][ncol] = 1;

q.push({{nrow,ncol},steps + 1});

}

}

}

return dist;

}

};

# Replace O's with X's

class Solution{

private:

void dfs(int row,int col, vector<vector<int>> &vis, vector<vector<char>> &mat, int delRow[], int delCol[]){

int n = mat.size();

int m = mat[0].size();

vis[row][col] = 1;

//check for top, right, bottom, left

for(int i=0;i<4;i++){

int nrow = row + delRow[i];

int ncol = col + delCol[i];

if(nrow >= 0 && nrow < n && ncol >= 0 && ncol < m && !vis[nrow][ncol] && mat[nrow][ncol] == 'O'){

dfs(nrow,ncol,vis,mat,delRow, delCol);

}

}

}

public:

vector<vector<char>> fill(int n, int m, vector<vector<char>> mat)

{

// code here

int delRow[] = {-1,0,1,0};

int delCol[] = {0,1,0,-1};

vector<vector<int>> vis(n,vector<int>(m,0));

//traverse first and last row

for(int j = 0; j < m ; j++){

//first row

if(!vis[0][j] && mat[0][j] == 'O'){

dfs(0,j,vis,mat,delRow, delCol);

}

//last row

if(!vis[n-1][j] && mat[n-1][j] == 'O'){

dfs(n-1,j,vis,mat,delRow, delCol);

}

}

for(int i=0;i<n;i++){

//first column

if(!vis[i][0] && mat[i][0] == 'O'){

dfs(i,0,vis,mat,delRow, delCol);

}

//last column

if(!vis[i][m-1] && mat[i][m-1] == 'O'){

dfs(i,m-1,vis,mat, delRow, delCol);

}

}

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(!vis[i][j] && mat[i][j] == 'O'){

mat[i][j] = 'X';

}

}

}

return mat;

}

};

# Number Of Enclaves

class Solution {

public:

int numberOfEnclaves(vector<vector<int>> &grid) {

// Code here

queue<pair<int,int>> q;

int n = grid.size();

int m = grid[0].size();

int vis[n][m] = {0};

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

//boundary ones

if(i == 0 || j == 0 || i == n-1 || j == m-1){

if(grid[i][j] == 1){

q.push({i,j});

vis[i][j] = 1;

}

}

}

}

int delRow[] = {-1,0,+1,0};

int delCol[] = {0,+1,0,-1};

while(!q.empty()){

int row = q.front().first;

int col = q.front().second;

q.pop();

//traverse all four direction

for(int i=0;i<4;i++){

int nrow = row + delRow[i];

int ncol = col + delCol[i];

if(nrow >= 0 && nrow < n && ncol >= 0 && ncol < m && vis[nrow][ncol] == 0 && grid[nrow][ncol] == 1){

q.push({nrow,ncol});

vis[nrow][ncol] = 1;

}

}

}

int count = 0;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(grid[i][j] == 1 && vis[i][j] == 0){

count++;

}

}

}

return count;

}

};

# Number of Distinct Islands

class Solution {

private:

void dfs(int row,int col, vector<vector<int>> &grid,vector<vector<int>> &vis,vector<pair<int,int>> &vec, int row0, int col0){

int n = grid.size();

int m = grid[0].size();

vis[row][col] = 1;

vec.push\_back({row - row0, col - col0});

int delRow[] = {-1,0,+1,0};

int delCol[] = {0,-1,0,+1};

for(int i=0;i<4;i++){

int nrow = row + delRow[i];

int ncol = col + delCol[i];

if(nrow >=0 && nrow < n && ncol >= 0 && ncol < m && !vis[nrow][ncol] && grid[nrow][ncol] == 1){

vis[nrow][ncol] = 1;

dfs(nrow,ncol,grid,vis,vec,row0,col0);

}

}

}

public:

int countDistinctIslands(vector<vector<int>>& grid) {

// code here

int n = grid.size();

int m = grid[0].size();

vector<vector<int>> vis(n,vector<int>(m,0));

set<vector<pair<int,int>>> st;

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(!vis[i][j] && grid[i][j] == 1){

vector<pair<int,int>> vec;

dfs(i,j,grid,vis,vec,i,j);

st.insert(vec);

}

}

}

return st.size();

}

};

# Bipartite Graph

class Solution {

private:

bool check(int start,int V,vector<int>adj[], int color[]){

queue<int> q;

q.push(start);

color[start] = 0;

while(!q.empty()){

int node = q.front();

q.pop();

for(auto it: adj[node]){

if(color[it] == -1){

color[it] = !color[node];

q.push(it);

}

else if(color[it] == color[node]){

return false;

}

}

}

return true;

}

public:

bool isBipartite(int V, vector<int>adj[]){

// Code here

int color[V];

for(int i=0;i<V;i++) color[i] = -1;

for(int i=0;i<V;i++){

if(color[i] == -1){

if(check(i,V,adj,color) == false){

return false;

}

}

}

return true;

}

};

# Bipartite Graph

class Solution {

private:

bool dfs(int s,int color,vector<int>&visited,vector<int>adj[]){

visited[s]=color;

for(auto x:adj[s]){

if(visited[x]==-1){

if(dfs(x,!color,visited,adj)==false)

return false;

}

else if(visited[x]==color)

return false;

}

return true;

}

public:

bool isBipartite(int V, vector<int>adj[]){

// Code here

vector<int>visited(V,-1);

for(int i=0;i<V;i++){

if(visited[i]==-1)

if(dfs(i,0,visited,adj)==false)

return false;

}

return true;

}

};

# Detect cycle in a directed graph

class Solution {

private:

bool check(int node, int vis[], int pathVis[], vector<int> adj[]){

vis[node] = 1;

pathVis[node] = 1;

for(auto it : adj[node]){

if(vis[it] == 0){

if(check(it,vis,pathVis,adj) == true) return true;

}

else if(pathVis[it] == 1){

return true;

}

}

pathVis[node] = 0;

return false;

}

public:

// Function to detect cycle in a directed graph.

bool isCyclic(int V, vector<int> adj[]) {

// code here

int vis[V] = {0};

int pathVis[V] = {0};

for(int i=0;i<V;i++){

if(vis[i] == 0){

if(check(i,vis,pathVis,adj) == true) return true;

}

}

return false;

}

};

# Eventual Safe States

class Solution {

private:

bool dfs\_check(int node, int vis[], int pathVis[], vector<int> adj[],int check[]){

vis[node] = 1;

pathVis[node] = 1;

check[node] = 0;

for(auto it : adj[node]){

if(vis[it] == 0){

if(dfs\_check(it,vis,pathVis,adj,check) == true) {

check[node] = 0;

return true;

}

}

else if(pathVis[it] == 1){

check[node] = 0;

return true;

}

}

check[node] = 1;

pathVis[node] = 0;

return false;

}

public:

vector<int> eventualSafeNodes(int V, vector<int> adj[]) {

// code here

int vis[V] = {0};

int pathVis[V] = {0};

int check[V] = {0};

vector<int> safeNodes;

for(int i=0;i<V;i++){

if(vis[i] == 0){

dfs\_check(i,vis,pathVis,adj,check);

}

}

for(int i=0;i<V;i++){

if(check[i] == 1) safeNodes.push\_back(i);

}

return safeNodes;

}

};

## BFS solution

class Solution {

public:

vector<int> eventualSafeNodes(int V, vector<int> adj[]) {

// code here

vector<int> adjRev[V];

int indegree[V] = {0};

for(int i=0;i<V;i++){

for(auto it : adj[i]){

adjRev[it].push\_back(i);

indegree[i]++;

}

}

queue<int> q;

vector<int> safeNodes;

for(int i=0;i<V;i++){

if(indegree[i] == 0){

q.push(i);

}

}

while(!q.empty()){

int node = q.front();

q.pop();

safeNodes.push\_back(node);

for(auto it: adjRev[node]){

indegree[it]--;

if(indegree[it] == 0){

q.push(it);

}

}

}

sort(safeNodes.begin(),safeNodes.end());

return safeNodes;

}

};

# Topological sort

class Solution

{

private:

void dfs(int node,int vis[],stack<int> &st,vector<int> adj[]){

vis[node] = 1;

for(auto it : adj[node]){

if(!vis[it]){

dfs(it,vis,st,adj);

}

}

st.push(node);

}

public:

//Function to return list containing vertices in Topological order.

vector<int> topoSort(int V, vector<int> adj[])

{

// code here

int vis[V] = {0};

stack<int> st;

for(int i=0;i<V;i++){

if(!vis[i]){

dfs(i, vis, st, adj);

}

}

vector<int> ans;

while(!st.empty()){

ans.push\_back(st.top());

st.pop();

}

return ans;

}

};

# Topological sort

class Solution

{

public:

//Function to return list containing vertices in Topological order.

vector<int> topoSort(int V, vector<int> adj[])

{

// code here

int indegree[V] = {0};

for(int i=0;i<V;i++){

for(auto it : adj[i]){

indegree[it]++;

}

}

queue<int> q;

for(int i=0;i<V;i++){

if(indegree[i] == 0){

q.push(i);

}

}

vector<int> topo;

while(!q.empty()){

int node = q.front();

q.pop();

topo.push\_back(node);

for(auto it: adj[node]){

indegree[it]--;

if(indegree[it] == 0){

q.push(it);

}

}

}

return topo;

}

};

# Prerequisite Tasks

class Solution {

public:

bool isPossible(int N,int P, vector<pair<int, int> >& prerequisites) {

// Code here

vector<int> adj[N];

for(auto it: prerequisites){

adj[it.first].push\_back(it.second);

}

int indegree[N] = {0};

for(int i=0;i<N;i++){

for(auto it : adj[i]){

indegree[it]++;

}

}

queue<int> q;

for(int i=0;i<N;i++){

if(indegree[i] == 0){

q.push(i);

}

}

vector<int> topo;

while(!q.empty()){

int node = q.front();

q.pop();

topo.push\_back(node);

for(auto it: adj[node]){

indegree[it]--;

if(indegree[it] == 0){

q.push(it);

}

}

}

if(topo.size() == N) return true;

return false;

}

};

# Course Schedule

class Solution

{

public:

vector<int> findOrder(int V, int m, vector<vector<int>> prerequisites)

{

//code here

vector<int> adj[V];

for(auto it: prerequisites){

adj[it[1]].push\_back(it[0]);

}

int indegree[V] = {0};

for(int i=0;i<V;i++){

for(auto it : adj[i]){

indegree[it]++;

}

}

queue<int> q;

for(int i=0;i<V;i++){

if(indegree[i] == 0){

q.push(i);

}

}

vector<int> topo;

while(!q.empty()){

int node = q.front();

q.pop();

topo.push\_back(node);

for(auto it: adj[node]){

indegree[it]--;

if(indegree[it] == 0){

q.push(it);

}

}

}

if(topo.size() == V) return topo;

return {};

}

};

# Alien Dictionary

class Solution{

private:

vector<int> topoSort(int V, vector<int> adj[])

{

// code here

int indegree[V] = {0};

for(int i=0;i<V;i++){

for(auto it : adj[i]){

indegree[it]++;

}

}

queue<int> q;

for(int i=0;i<V;i++){

if(indegree[i] == 0){

q.push(i);

}

}

vector<int> topo;

while(!q.empty()){

int node = q.front();

q.pop();

topo.push\_back(node);

for(auto it: adj[node]){

indegree[it]--;

if(indegree[it] == 0){

q.push(it);

}

}

}

return topo;

}

public:

string findOrder(string dict[], int N, int K) {

//code here

vector<int> adj[K];

for(int i=0;i<N-1;i++){

string s1 = dict[i];

string s2 = dict[i+1];

int len = min(s1.size(),s2.size());

for(int ptr = 0; ptr < len; ptr++){

if(s1[ptr] != s2[ptr]){

adj[s1[ptr]-'a'].push\_back(s2[ptr]-'a');

break;

}

}

}

vector<int> topo = topoSort(K,adj);

string ans = "";

for(auto it : topo){

ans = ans + char(it + 'a');

}

return ans;

}

};

# Shortest path in Directed Acyclic Graph

// User function Template for C++

class Solution {

private:

void topoSort(int node,vector<pair<int,int>> adj[], int vis[], stack<int> &stk){

vis[node] = 1;

for(auto it : adj[node]){

int v = it.first;

if(!vis[v]){

topoSort(v, adj, vis,stk);

}

}

stk.push(node);

}

public:

vector<int> shortestPath(int N,int M, vector<vector<int>>& edges){

// code here

vector<pair<int,int>> adj[N];

//initiate the adjacency list

for (int i = 0; i < M; i++) {

int u = edges[i][0];

int v = edges[i][1];

int wt = edges[i][2];

adj[u].push\_back({v, wt});

}

//find the topo sort

int vis[N] = {0};

stack<int> stk;

for (int i = 0; i < N; i++) {

if (!vis[i]) {

topoSort(i, adj, vis, stk);

}

}

//do the distance calculation

vector<int> dist(N);

for (int i = 0; i < N; i++) {

dist[i] = 1e9;

}

dist[0] = 0;

while (!stk.empty()) {

int node = stk.top();

stk.pop();

for (auto it: adj[node]) {

int v = it.first;

int wt = it.second;

if (dist[node] + wt < dist[v]) {

dist[v] = wt + dist[node];

}

}

}

for (int i = 0; i < N; i++) {

if (dist[i] == 1e9) dist[i] = -1;

}

return dist;

}

};

# Shortest path in Undirected Graph having unit distance

class Solution {

public:

vector<int> shortestPath(vector<vector<int>>& edges, int N,int M, int src){

// code here

vector<int> adj[N];

for(auto it: edges){

adj[it[0]].push\_back(it[1]);

adj[it[1]].push\_back(it[0]);

}

vector<int> dist(N);

for(int i=0;i<N;i++) dist[i] = 1e9;

dist[src] = 0;

queue<int> q;

q.push(src);

while(!q.empty()){

int node = q.front();

q.pop();

for(auto it : adj[node]){

if(dist[node] + 1 < dist[it]){

dist[it] = dist[node] + 1;

q.push(it);

}

}

}

for(int i=0;i<N;i++){

if(dist[i] == 1e9){

dist[i] = -1;

}

}

return dist;

}

};

# Word Ladder I

class Solution {

public:

int wordLadderLength(string startWord, string targetWord, vector<string>& wordList) {

// Code here

queue<pair<string,int>> q;

q.push({startWord,1});

unordered\_set<string> st(wordList.begin(),wordList.end());

st.erase(startWord);

while(!q.empty()){

string word = q.front().first;

int steps = q.front().second;

q.pop();

if(word == targetWord) return steps;

for(int i=0;i<word.size();i++){

char original = word[i];

for(char ch = 'a'; ch <= 'z'; ch++){

word[i] = ch;

if(st.find(word) != st.end()){

st.erase(word);

q.push({word,steps + 1});

}

}

word[i] = original;

}

}

return 0;

}

};

# Word Ladder II

class Solution {

public:

vector<vector<string>> findSequences(string beginWord, string endWord, vector<string>& wordList) {

// code here

unordered\_set<string> st(wordList.begin(),wordList.end());

queue<vector<string>> q;

q.push({beginWord});

vector<string> usedOnLevel;

usedOnLevel.push\_back(beginWord);

int level = 0;

vector<vector<string>> ans;

while(!q.empty()){

vector<string> vec = q.front();

q.pop();

if(vec.size() > level){

level++;

for(auto it: usedOnLevel){

st.erase(it);

}

}

string word = vec.back();

if(word == endWord){

//first sequence where we reached the end

if(ans.size() == 0){

ans.push\_back(vec);

}

else if(ans[0].size() == vec.size()){

ans.push\_back(vec);

}

}

for(int i=0;i<word.size();i++){

char original = word[i];

for(char c = 'a'; c <= 'z';c++){

word[i] = c;

if(st.count(word) > 0){

vec.push\_back(word);

q.push(vec);

//mark as visited on the level

usedOnLevel.push\_back(word);

vec.pop\_back();

}

}

word[i] = original;

}

}

return ans;

}

};

# Implementing Dijkstra Algorithm

class Solution

{

public:

//Function to find the shortest distance of all the vertices

//from the source vertex S.

vector <int> dijkstra(int V, vector<vector<int>> adj[], int S)

{

// Create a priority queue for storing the nodes as a pair {dist,node}

// where dist is the distance from source to the node.

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

// Initialising distTo list with a large number to

// indicate the nodes are unvisited initially.

// This list contains distance from source to the nodes.

vector<int> distTo(V, INT\_MAX);

// Source initialised with dist=0.

distTo[S] = 0;

pq.push({0, S});

// Now, pop the minimum distance node first from the min-heap

// and traverse for all its adjacent nodes.

while (!pq.empty())

{

int node = pq.top().second;

int dis = pq.top().first;

pq.pop();

// Check for all adjacent nodes of the popped out

// element whether the prev dist is larger than current or not.

for (auto it : adj[node])

{

int v = it[0];

int w = it[1];

if (dis + w < distTo[v])

{

distTo[v] = dis + w;

// If current distance is smaller,

// push it into the queue.

pq.push({dis + w, v});

}

}

}

// Return the list containing shortest distances

// from source to all the nodes.

return distTo;

}

};

Solution-2

class Solution

{

public:

//Function to find the shortest distance of all the vertices

//from the source vertex S.

vector <int> dijkstra(int V, vector<vector<int>> adj[], int S)

{

// Code here

set<pair<int,int>> st;

vector<int> dist(V,1e9);

st.insert({0,S});

dist[S] = 0;

while(!st.empty()){

auto it = \*(st.begin());

int node = it.second;

int dis = it.first;

st.erase(it);

for(auto it: adj[node]){

int adjNode = it[0];

int edgeWeight = it[1];

if(dis + edgeWeight < dist[adjNode]){

if(dist[adjNode] != 1e9){

st.erase({dist[adjNode], adjNode});

}

dist[adjNode] = dis + edgeWeight;

st.insert({dist[adjNode],adjNode});

}

}

}

return dist;

}

};

# Shortest Distance in a Binary Maze

class Solution {

public:

int shortestPath(vector<vector<int>> &grid, pair<int, int> source,

pair<int, int> destination) {

// code here

if (source.first == destination.first &&

source.second == destination.second){

return 0;

}

queue<pair<int,pair<int,int>>> q;

int n = grid.size();

int m = grid[0].size();

vector<vector<int>> dist(n,vector<int>(m,1e9));

dist[source.first][source.second] = 0;

q.push({0,{source.first,source.second}});

int dr[] = {-1,0,1,0};

int dc[] = {0,1,0,-1};

while(!q.empty()){

auto it = q.front();

q.pop();

int dis = it.first;

int r = it.second.first;

int c = it.second.second;

for(int i=0;i<4;i++){

int newr = r + dr[i];

int newc = c + dc[i];

if(newr >= 0 && newr < n && newc >= 0 && newc < m &&

grid[newr][newc] == 1 && dis + 1 < dist[newr][newc])

{

dist[newr][newc] = 1 + dis;

if(newr == destination.first && newc == destination.second)

{

return dis + 1;

}

q.push({1 + dis, {newr,newc}});

}

}

}

return -1;

}

};

# Path With Minimum Effort

class Solution {

public:

int MinimumEffort(vector<vector<int>>& heights) {

priority\_queue<pair<int, pair<int, int>>,

vector<pair<int, pair<int, int>>>,

greater<pair<int, pair<int, int>>>>

pq;

int n = heights.size();

int m = heights[0].size();

vector<vector<int>> dist(n, vector<int>(m, 1e9));

dist[0][0] = 0;

pq.push({0, {0, 0}});

int dr[] = {-1, 0, 1, 0};

int dc[] = {0, 1, 0, -1};

while (!pq.empty())

{

auto it = pq.top();

pq.pop();

int diff = it.first;

int row = it.second.first;

int col = it.second.second;

if (row == n - 1 && col == m - 1)

return diff;

for (int i = 0; i < 4; i++)

{

int newr = row + dr[i];

int newc = col + dc[i];

if (newr >= 0 && newc >= 0 && newr < n && newc < m)

{

int newEffort = max(abs(heights[row][col] - heights[newr][newc]), diff);

if (newEffort < dist[newr][newc])

{

dist[newr][newc] = newEffort;

pq.push({newEffort, {newr, newc}});

}

}

}

}

return 0;

}

};

# Cheapest Flights Within K Stops

class Solution {

public:

int CheapestFLight(int n, vector<vector<int>>& flights, int src, int dst, int K) {

// Code here

vector<pair<int,int>> adj[n];

for(auto it : flights){

adj[it[0]].push\_back({it[1],it[2]});

}

queue<pair<int,pair<int,int>>> q;

q.push({0,{src,0}});

vector<int> dist(n,1e9);

dist[src] = 0;

while(!q.empty()){

auto it = q.front();

q.pop();

int stops = it.first;

int node = it.second.first;

int cost = it.second.second;

if(stops > K) continue;

for(auto iter : adj[node]){

int adjNode = iter.first;

int edgeWeight = iter.second;

if(cost + edgeWeight < dist[adjNode] && stops <= K){

dist[adjNode] = cost + edgeWeight;

q.push({stops + 1,{adjNode , cost + edgeWeight}});

}

}

}

if(dist[dst] == 1e9) return -1;

return dist[dst];

}

};

# Minimum Multiplications to reach End

class Solution {

public:

int minimumMultiplications(vector<int>& arr, int start, int end) {

// code here

if(start == end) return 0;

queue<pair<int,int>> q;

q.push({start,0});

vector<int> dist(100000,1e9);

dist[start] = 0;

int mod = 100000;

while(!q.empty()){

int node = q.front().first;

int steps = q.front().second;

q.pop();

for(auto it : arr){

int num = (it \* node) % mod;

if(steps + 1 < dist[num]){

dist[num] = steps + 1;

if(num == end) return steps + 1;

q.push({num,steps + 1});

}

}

}

return -1;

}

};

# Number of Ways to Arrive at Destination

class Solution {

public:

int countPaths(int n, vector<vector<int>>& roads) {

// code here

vector<pair<int,int>>adj[n];

for(auto it:roads)

{

adj[it[0]].push\_back({it[1],it[2]});

adj[it[1]].push\_back({it[0],it[2]});

}

int mod=int(1e9+7);

priority\_queue<pair<long long,int>,vector<pair<long long,int>>,

greater<pair<long long,int>>> q;

vector< long long>dist(n,1e18);

dist[0]=0;

vector<long long>ways(n,0);

ways[0]=1;

q.push({0,0});

while(!q.empty())

{

int dis=q.top().first;

int node=q.top().second;

q.pop();

for(auto it:adj[node])

{

int adjNode=it.first;

long long edW=it.second;

if(edW+dis<dist[adjNode])

{

dist[adjNode]=edW+dis;

ways[adjNode]=ways[node];

q.push({edW+dis,adjNode});

}

else if(edW+dis==dist[adjNode])

{

ways[adjNode]=(ways[adjNode]+ways[node])%mod;

}

}

}

return ways[n-1]%mod;

}

};

[**198. House Robber**](https://leetcode.com/problems/house-robber/)

class Solution{

public:

    int recur(int ind, vector<int> &nums, vector<int> &dp){

        if(ind == 0) return nums[ind];

        if(ind < 0) return 0;

        if(dp[ind] != -1){

            return dp[ind];

        }

        int pick = nums[ind] + recur(ind - 2, nums, dp);

        int notPick = 0 + recur(ind - 1, nums, dp);

        return dp[ind] = max(pick, notPick);

    }

    int rob(vector<int>& nums) {

        int n = nums.size();

        vector<int> dp(n,-1);

        return recur(n-1,nums,dp);

    }

};

[**213. House Robber II**](https://leetcode.com/problems/house-robber-ii/)

class Solution {

public:

    int solve(vector<int>& arr){

     int n = arr.size();

     int prev = arr[0];

     int prev2 =0;

     for(int i=1;i<n;i++){

        int pick = arr[i];

        if(i>1)

            pick += prev2;

        int nonPick = 0 + prev;

        int cur\_i = max(pick, nonPick);

        prev2 = prev;

        prev= cur\_i;

     }

      return prev;

    }

    int rob(vector<int>& nums) {

       vector<int> temp1, temp2;

       int n = nums.size();

       if(n == 1) return nums[0];

       for(int i=0;i<n;i++){

           if(i != 0) temp1.push\_back(nums[i]);

           if(i != n-1) temp2.push\_back(nums[i]);

       }

       return max(solve(temp1),solve(temp2));

    }

};

# Permutations of a given string

class Solution

{

private:

void permute(string ip,string op,vector<string> &ans){

if(ip.size() == 0){

ans.push\_back(op);

return;

}

unordered\_set<char> mp;

for(int i=0;i<ip.size();i++){

if(mp.find(ip[i]) == mp.end()){

mp.insert(ip[i]);

string newip = ip.substr(0,i) + ip.substr(i+1);

string newop = op + ip[i];

permute(newip,newop,ans);

}

}

}

public:

vector<string>find\_permutation(string S)

{

// Code here there

vector<string> ans;

permute(S, "", ans);

return ans;

}

};

## Number of Unique Paths

class Solution

{

public:

int solve(int i, int j,vector<vector<int>> &dp){

if(i == 0 && j == 0) return 1;

if(i < 0 || j < 0) return 0;

if (dp[i][j] != -1){

return dp[i][j];

}

int up = solve(i-1,j,dp);

int right = solve(i, j-1,dp);

return dp[i][j] = up + right;

}

//Function to find total number of unique paths.

int NumberOfPath(int a, int b)

{

//code here

vector<vector<int>> dp(a,vector<int>(b,-1));

return solve(a-1,b-1,dp);

}

};

[**63. Unique Paths II**](https://leetcode.com/problems/unique-paths-ii/)

class Solution {

public:

    int helper(vector<vector<int>>& dp, vector<vector<int>>& obstacleGrid, int i, int j) {

        if (i < 0 || j < 0) {

            return 0;

        }

        if (obstacleGrid[i][j] == 1) {

            return 0;

        }

        if (i == 0 && j == 0) {

            return 1;

        }

        if (dp[i][j] != -1) {

            return dp[i][j];

        }

        return dp[i][j] = helper(dp, obstacleGrid, i - 1, j) + helper(dp, obstacleGrid, i, j - 1);

    }

    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {

        int m = obstacleGrid.size();

        int n = obstacleGrid[0].size();

        vector<vector<int>> dp(m, vector<int>(n, -1));

        return helper(dp, obstacleGrid, m - 1, n - 1);

    }

};

[**64. Minimum Path Sum**](https://leetcode.com/problems/minimum-path-sum/)

class Solution {

public:

    int minPathSum(vector<vector<int>>& grid) {

        int m = grid.size();

        int n = grid[0].size();

        vector<vector<int>> dp(m,vector<int>(n,-1));

        return solve(m-1,n-1,grid,dp);

    }

    int solve(int i,int j,vector<vector<int>> &matrix,vector<vector<int>> &dp){

        if(i == 0 && j == 0) return matrix[0][0];

        if(i < 0 || j < 0) return 1e9;

        if(dp[i][j] !=-1) return dp[i][j];

        int up = matrix[i][j] + solve(i-1,j,matrix,dp);

        int left = matrix[i][j] + solve(i,j-1,matrix,dp);

        return dp[i][j] = min(up,left);

    }

};