```
#include <bits/stdc++.h>
using namespace std;
int main() { return 0; }

// Reverse the array
// Find the maximum and minimum element in an array
// Find the "Kth" max and min element of an array
// Sort an array of 0s, 1s and 2s
void sort012(int nums[], int n)
{
 int lo = 0, mid = 0, hi = n - 1;
 while (mid <= hi)
 {
  if (nums[mid] == 0)
   swap(nums[lo++], nums[mid++]);
  else if (nums[mid] == 1)
   mid++;
  else
   swap(nums[hi--], nums[mid]);
 }
}
// Move all the negative elements to one side of the array
void movingNegToLeft(int a[], int n)
{
 int l = 0, r = 0;
 while (r < n)
 {
  if (a[r] < 0)
   swap(a[l++], a[r++]);
  else
   r++;
 }
}
// Find the Union and Intersection of the two sorted arrays.

// Write a program to cyclically rotate an array by one.
void rotate(int arr[], int n)
{
 for (int i = n - 1; i > 0; i--)
 {
  swap(arr[i - 1], arr[i]);
 }
}
// reverse trick

// Kadane's Algorithm
long long maxSubarraySum(int arr[], int n)
{
 long long ans = INT_MIN, temp = 0;
 for (int i = 0; i < n; i++)
 {
  temp += arr[i];
  ans = max(ans, temp);
  if (temp < 0) temp = 0;
 }
```

```
 return ans == INT_MIN ? -1 : ans;
}

// Minimise the maximum difference between heights [V.IMP]
int getMinDiff(int arr[], int n, int k)
{
 sort(arr, arr + n);
 int minele, maxele;
 int result = arr[n - 1] - arr[0]; // b-a
 for (int i = 1; i <= n - 1; i++)
 {
  if (arr[i] >= k){
   maxele = max(arr[i - 1] + k, arr[n - 1] - k);
   minele = min(arr[0] + k, arr[i] - k);
   result = min(result, maxele - minele);
  }
 }
 return result;
}
int smallestRangeII(vector<int> A, int K)
{
 sort(A.begin(), A.end());
 int n = A.size(), mx = A[n - 1], mn = A[0], res = mx - mn;
 for (int i = 0; i < n - 1; ++i)
 {
  mx = max(mx, A[i] + 2 * K);
  mn = min(A[i + 1], A[0] + 2 * K);
  res = min(res, mx - mn);
 }
 return res;
}

// minimum no. of jumps to reach end of an array
// recursion
int jump(vector<int> &nums, int pos = 0)
{
 if (pos >= size(nums) - 1)
  return 0;
 int minJumps = 10001;
 for (int j = 1; j <= nums[pos]; j++)
  minJumps = min(minJumps, 1 + jump(nums, pos + j));
 return minJumps;
}
// dp
int solve(vector<int> &nums, vector<int> &dp, int pos)
{
 if (pos >= size(nums) - 1)
  return 0;
 if (dp[pos] != 10001)
  return dp[pos];
 for (int j = 1; j <= nums[pos]; j++)
  dp[pos] = min(dp[pos], 1 + solve(nums, dp, pos + j));
 return dp[pos];
}
// tabulation
```

```
int jump(vector<int> &nums)
{
 int n = size(nums);
 vector<int> dp(n, 10001);
 dp[n - 1] = 0;
 for (int i = n - 2; i >= 0; i--)
  for (int jumpLen = 1; jumpLen <= nums[i]; jumpLen++)
   dp[i] = min(dp[i], 1 + dp[min(n - 1, i + jumpLen)]);
 return dp[0];
}
// Greedy BFS
int jump(vector<int> &nums)
{
 int n = size(nums), i = 0, maxReachable = 0, lastJumpedPos = 0, jumps = 0;
 while (lastJumpedPos < n - 1)
 {
  maxReachable = max(maxReachable, i + nums[i]);
  if (i == lastJumpedPos)
  {
   lastJumpedPos = maxReachable;
   jumps++;
  }
  i++;
 }
 return jumps;
}

// find duplicate in an array of N+1 Integers
int findDuplicate(vector<int> &nums)
{
 int hare = nums[0], tortoise = nums[0];
 do
 {
  hare = nums[nums[hare]];
  tortoise = nums[tortoise];
 } while (hare != tortoise);

 tortoise = nums[0];
 while (hare != tortoise)
 {
  hare = nums[hare];
  tortoise = nums[tortoise];
 }

 return hare;
}

// Merge Without Extra Space
void merge(int arr1[], int arr2[], int n, int m)
{
 int i = n - 1, j = 0;
 while (i >= 0 && j < m)
 {
  if (arr1[i] > arr2[j])
  {
```

```
   swap(arr1[i], arr2[j]);
   i--;
   j++;
  }
  else
   break;
 }
 sort(arr1, arr1 + n);
 sort(arr2, arr2 + m);
}
// gap method
int nextGap(int gap)
{
 if (gap <= 1)
  return 0;
 return (gap / 2) + (gap % 2);
}

void merge(int *arr1, int *arr2, int n, int m)
{
 int i, j, gap = n + m;
 for (gap = nextGap(gap);
   gap > 0; gap = nextGap(gap))
 {
  for (i = 0; i + gap < n; i++)
   if (arr1[i] > arr1[i + gap])
    swap(arr1[i], arr1[i + gap]);

  for (j = gap > n ? gap - n : 0;
    i < n && j < m;
    i++, j++)
   if (arr1[i] > arr2[j])
    swap(arr1[i], arr2[j]);

  if (j < m)
  {
   for (j = 0; j + gap < m; j++)
    if (arr2[j] > arr2[j + gap])
     swap(arr2[j], arr2[j + gap]);
  }
 }
}

// Merge Intervals
vector<vector<int>> merge(vector<vector<int>> &intervals)
{
 sort(intervals.begin(), intervals.end());
 vector<vector<int>> merged;
 for (auto interval : intervals)
 {
  if (merged.empty() || merged.back()[1] < interval[0])
  {
   merged.push_back(interval);
  }
  else
```

```
  {
   merged.back()[1] = max(merged.back()[1], interval[1]);
  }
 }
 return merged;
}

// Next Permutation
void nextPermutation(vector<int> &nums)
{
 int n = nums.size(), k, l;
 for (k = n - 2; k >= 0; k--)
 {
  if (nums[k] < nums[k + 1])
   break;
 }
 if (k < 0)
  reverse(nums.begin(), nums.end());
 else
 {
  for (l = n - 1; l > k; l--)
  {
   if (nums[l] > nums[k])
    break;
  }
  swap(nums[k], nums[l]);
  reverse(nums.begin() + k + 1, nums.end());
 }
}

// find all pairs on integer array whose sum is equal to given number
int getPairsCount(int arr[], int n, int k)
{
 map<int, int> m;
 int ans = 0;
 for (int i = 0; i < n; i++)
 {
  if (m.find(k - arr[i]) != m.end())
   ans += m[k - arr[i]];
  m[arr[i]]++;
 }
 return ans;
}

// find common elements In 3 sorted arrays
//  Rearrange the array in alternating positive and negative items
void alternatePositiveNegative(int a[], int n)
{
 int n;
 cin >> n;
 long long a[n + 10];
 for (int i = 0; i < n; i++)
  cin >> a[i];
 int l = 0, r = 0;
 while (r < n)
```

```
 {
  if (a[r] < 0)
   swap(a[l++], a[r++]);
  else
   r++;
 }
 for (int i = 0; i < n; i++)
  cout << " " << a[i];
 cout << "\n";
 for (int i = 0; i < n / 2; i += 2)
 {
  cout << " " << a[i] << " " << a[i + (n / 2)] << "\n";
  swap(a[i], a[i + (n / 2)]);
 }
 for (int i = 0; i < n; i++)
  cout << " " << a[i];
 cout << "\n";
}

// Subarray with sum equal to 0
bool subArrayExists(int arr[], int n)
{
 unordered_set<int> sumSet;
 int sum = 0;
 for (int i = 0; i < n; i++)
 {
  sum += arr[i];
  if (sum == 0 || sumSet.find(sum) != sumSet.end())
   return true;
  sumSet.insert(sum);
 }
 return false;
}

// Count Subarray with sum equal to K
int subarraySum(vector<int> &nums, int k)
{
 int count = 0, sum = 0;
 unordered_map<int, int> m;
 m[0] = 1;
 for (int i = 0; i < nums.size(); i++)
 {
  sum += nums[i];
  count += m[sum - k];
  m[sum]++;
 }
 return count;
}

// Count Subarray with sum equal to 0
int subarraySum(vector<int> &nums)
{
 int count = 0, sum = 0;
 unordered_map<int, int> m;
 m[0] = 1;
```

```
 for (int i = 0; i < nums.size(); i++)
 {
  sum += nums[i];
  count += m[sum];
  m[sum]++;
 }
 return count;
}

// Maximum Product SubArray
int maxProduct(vector<int> &nums)
{
 if (nums.size() == 0)
  return 0;
 int maxSub = nums[0];
 int minSub = nums[0];
 int maxProductSub = nums[0];
 for (int i = 1; i < nums.size(); i++)
 {
  if (nums[i] < 0)
   swap(minSub, maxSub);
  maxSub = max(maxSub * nums[i], nums[i]);
  minSub = min(minSub * nums[i], nums[i]);
  maxProductSub = max(maxProductSub, maxSub);
 }
 return maxProductSub;
}

// Longest consecutive subsequence
int findLongestConseqSubseq(int arr[], int n)
{
 unordered_set<int> s(arr, arr + n);
 int c, m = 0;
 for (int i = 0; i < n; i++)
 {
  if (s.find(arr[i] - 1) == s.end())
  {
   c = 1;
   while (s.find(arr[i] + c) != s.end())
   {
    c++;
   }
   m = max(c, m);
  }
 }
 return m;
}

// longest consecutive subarray
int findLength(int arr[], int n)
{
 int max_len = 1;
 for (int i = 0; i < n - 1; i++)
 {
  int mn = arr[i], mx = arr[i];
```

```
  for (int j = i + 1; j < n; j++)
  {
   mn = min(mn, arr[j]);
   mx = max(mx, arr[j]);
   if ((mx - mn) == j - i)
    max_len = max(max_len, mx - mn + 1);
  }
 }
 return max_len;
}
int findLength(int arr[], int n)
{ // duplicate element case
 int max_len = 1;
 for (int i = 0; i < n - 1; i++)
 {
  set<int> s;
  s.insert(arr[i]);
  int mn = arr[i], mx = arr[i];
  for (int j = i + 1; j < n; j++)
  {
   if (myset.find(arr[j]) != myset.end())
    break;
   myset.insert(arr[j]);
   mn = min(mn, arr[j]);
   mx = max(mx, arr[j]);
   if (mx - mn == j - i)
    max_len = max(max_len, mx - mn + 1);
  }
 }
 return max_len;
}

// Given an array of size n and a number k, fin all elements that appear more than " n/k " times.
// Find whether an array is a subset of another array

// Majority Element
int majorityElement(vector<int> &nums)
{
 int c(-1), cnt(0);
 for (auto n : nums)
 {
  if (cnt == 0)
   c = n;
  cnt += (n == c) ? 1 : -1;
 }
 return c;
}

// 3 sum
bool find3Numbers(int a[], int n, int X)
{
 sort(a, a + n);
 for (int i = 0; i < n; i++)
 {
  int tgt = X - a[i];
```

```
  int l = i + 1, r = n - 1;
  while (l < r)
  {
   if (a[l] + a[r] < tgt)
    l++;
   else if (a[l] + a[r] > tgt)
    r--;
   else
    return 1;
  }
 }
 return 0;
}

// Trapping Rain water problem
int maxWater(int arr[], int n)
{
 int res = 0;
 for (int i = 1; i < n - 1; i++)
 {
  int left = arr[i];
  for (int j = 0; j < i; j++)
   left = max(left, arr[j]);
  int right = arr[i];
  for (int j = i + 1; j < n; j++)
   right = max(right, arr[j]);
  res = res + (min(left, right) - arr[i]);
 }
 return res;
}
int findWater(int arr[], int n)
{
 int left[n], right[n];
 int water = 0;
 left[0] = arr[0];
 for (int i = 1; i < n; i++)
  left[i] = max(left[i - 1], arr[i]);
 right[n - 1] = arr[n - 1];
 for (int i = n - 2; i >= 0; i--)
  right[i] = max(right[i + 1], arr[i]);
 for (int i = 1; i < n - 1; i++)
 {
  int var = min(left[i - 1], right[i + 1]);
  if (var > arr[i])
  {
   water += var - arr[i];
  }
 }
 return water;
}
int findWater(int arr[], int n)
{
 int result = 0;
 int left_max = 0, right_max = 0;
 int lo = 0, hi = n - 1;
```

```
 while (lo <= hi)
 {
  if (arr[lo] < arr[hi])
  {
   if (arr[lo] > left_max)
    left_max = max(arr[lo], left_max);
   else
    result += left_max - arr[lo];
   lo++;
  }
  else if (arr[lo] >= arr[hi])
  {
   if (arr[hi] > right_max)
    right_max = max(arr[hi], right_max);
   else
    result += right_max - arr[hi];
   hi--;
  }
 }
 return result;
}
int maxWater(int height[], int n)
{
 stack<int> st;
 int ans = 0;
 for (int i = 0; i < n; i++)
 {
  while ((!st.empty())&&(height[st.top()] < height[i]))
  {
   int pop_height = height[st.top()];
   st.pop();
   if (st.empty())
    break;
   int distance=i-st.top()-1;
   int min_height=min(height[st.top()],height[i])-pop_height;
   ans += distance * min_height;
  }
  st.push(i);
 }
 return ans;
}

// Container with Most Water
int maxArea(int A[], int len)
{
 int area = 0;
 for (int i = 0; i < len; i++)
  for (int j = i + 1; j < len; j++)
   area = max(area, min(A[j], A[i]) * (j - i));
 return area;
}
int maxArea(int A[], int len)
{
 int l = 0, r = len - 1, area = 0;
 while (l < r)
```

```
 {
  area = max(area, min(A[l], A[r]) * (r - l));
  if (A[l] < A[r]) l++;
  else r--;
 }
 return area;
}

// Largest rectangle in a histogram
int largestRectangleArea(vector<int> &heights)
{
 int n = heights.size();
 if (n == 0)
  return 0;
 int maxArea = 0;
 vector<int> left(n);
 vector<int> right(n);
 left[0] = -1;
 right[n - 1] = n;
 for (int i = 1; i < n; i++){
  int prev = i - 1;
  while (prev >= 0 && heights[prev] >= heights[i])
   prev = left[prev];
  left[i] = prev;
 }
 for (int i = n - 2; i >= 0; i--)
 {
  int prev = i + 1;
  while (prev < n && heights[prev] >= heights[i])
   prev = right[prev];
  right[i] = prev;
 }
 for (int i = 0; i < n; i++)
 {
  int width = right[i] - left[i] - 1;
  maxArea = max(maxArea, heights[i] * width);
 }
 return maxArea;
}
// o(n)
int largestRectangleArea(vector<int> &heights)
{
 int n = heights.size();
 int maxArea = 0;
 stack<int> st;
 for (int i = 0; i <= n; i++)
 {
  int currHeight = i == n ? 0 : heights[i];
  while (!st.empty() && currHeight < heights[st.top()])
  {
   int top = st.top();
   st.pop();
   int width = st.empty() ? i : i - st.top() - 1;
   int area = heights[top] * width;
   maxArea = max(area, maxArea);
```

```
  }
  st.push(i);
 }
 return maxArea;
}

// Smallest Subarray with sum greater than a given value
int minSubArrayLen(int tgt, vector<int> &a)
{
 int l = 0, r = 0, t = 0, ans = INT_MAX, n = a.size();
 while (r < n){
  t += a[r++];
  while (t >= tgt)
   t -= a[l++], ans = min(ans, r - l + 1);
 }
 return ans == INT_MAX ? 0 : ans;
}

// Three way partitioning of an array around a given value
void threeWayPartition(vector<int> &t, int a, int b)
{
 int n = t.size(), l = 0, m = 0, r = n - 1;
 while (m <= r)
 {
  if (t[m] < a)
   swap(t[l++], t[m++]);
  else if (t[m] > b)
   swap(t[m], t[r--]);
  else
   m++;
 }
}

// Minimum swaps required bring elements less equal K together
int minSwap(int a[], int n, int k)
{
 int count = 0;
 for (int i = 0; i < n; i++)
  if (a[i] <= k)
   count++;
 if (!count || count == n)
  return 0;
 int l = 0, r = 0, ans = INT_MAX, req = 0;

 while (r - l + 1 < count)
 {
  if (a[r] > k)
   req++;
  r++;
 }
 if (a[r] > k)
  req++;
 while (r < n)
 {
  ans = min(req, ans);
```

```
  if (a[l++] > k)
   req--;
  if (a[++r] > k)
   req++;
 }
 return ans;
}

// Median of 2 sorted arrays of different size
// way1: 2Ptr Method
int getMedian(int ar1[], int ar2[], int n, int m)
{
 int i = 0; /* Current index of input array ar1[] */
 int j = 0; /* Current index of input array ar2[] */
 int count;
 int m1 = -1, m2 = -1;
 /*loop till (m+n)/2*/
 for (count = 0; count <= (m + n) / 2; count++)
 {
  // store (n+m)/2-1 in m2
  m2 = m1;
  if (i != n && j != m)
  {
   m1 = (ar1[i] > ar2[j]) ? ar2[j++] : ar1[i++];
  }
  else if (i < n)
  {
   m1 = ar1[i++];
  }
  // for case when j<m,
  else
  {
   m1 = ar2[j++];
  }
 }
 if ((m + n) % 2 == 1)
 {
  return m1;
 }
 else
 {
  return (m1 + m2) / 2;
 }
}
// Way 2: Binary Search
double findMedianSortedArrays(vector<int> &nums1,
         vector<int> &nums2)
{
 if (nums2.size() < nums1.size())
  return findMedianSortedArrays(nums2, nums1);
 int n1 = nums1.size();
 int n2 = nums2.size();
 int low = 0, high = n1;
 while (low <= high)
 {
```

```
  int cut1 = (low + high) >> 1;
  int cut2 = (n1 + n2 + 1) / 2 - cut1;
  int left1 = cut1 == 0 ? INT_MIN : nums1[cut1 - 1];
  int left2 = cut2 == 0 ? INT_MIN : nums2[cut2 - 1];
  int right1 = cut1 == n1 ? INT_MAX : nums1[cut1];
  int right2 = cut2 == n1 ? INT_MAX : nums2[cut2];
  if (left1 <= right2 && left2 <= right1)
  {
   if ((n1 + n2) % 2 == 0)
    return (max(left1, left2) + min(right1,right2))/2.0;
   else
    return max(left1, left2);
  }
  else if (left1 > right2)
  {
   high = cut1 - 1;
  }
  else
  {
   low = cut1 + 1;
  }
 }
 return 0.0;
}

// Kth element of two Sorted Arrays
double kthElement(vector<int> &nums1,
      vector<int> &nums2)
{
 if (nums2.size() < nums1.size())
  return fkthElement(nums2, nums1);
 int n1 = nums1.size();
 int n2 = nums2.size();
 int low = 0, high = n1;
 while (low <= high)
 {
  int cut1 = (low + high) >> 1;
  int cut2 = k - cut1;
  int left1 = cut1 == 0 ? INT_MIN : nums1[cut1 - 1];
  int left2 = cut2 == 0 ? INT_MIN : nums2[cut2 - 1];
  int right1 = cut1 == n1 ? INT_MAX : nums1[cut1];
  int right2 = cut2 == n1 ? INT_MAX : nums2[cut2];
  if (left1 <= right2 && left2 <= right1)
  {
   return max(left1, left2);
  }
  else if (left1 > right2)
  {
   high = cut1 - 1;
  }
  else
  {
   low = cut1 + 1;
  }
 }
```

```
 return 0.0;
}

// Sorted Matrix Median
// Kth Element in sorted Matrix
int binaryMedian(int m[][], int r, int c)
{
 int min = INT_MAX, max = INT_MIN;
 for (int i = 0; i < r; i++)
 {
  if (m[i][0] < min)
   min = m[i][0];
  if (m[i][c - 1] > max)
   max = m[i][c - 1];
 }

 int desired = (r * c + 1) / 2;
 while (min < max)
 {
  int mid = min + (max - min) / 2;
  int place = 0;
  for (int i = 0; i < r; ++i)
   place += upper_bound(m[i], m[i] + c, mid) - m[i];
  if (place < desired)
   min = mid + 1;
  else
   max = mid;
 }
 return min;
}

// Spiral traversal on a Matrix
vector<int> spiralOrder(vector<vector<int>>
       &matrix)
{
 vector<int> ans;
 if (matrix.size() == 0)
  return ans;
 int R = matrix.size(), C = matrix[0].size();
 vector<vector<bool>> seen(R, vector<bool>(C,false));
 int dr[] = {0, 1, 0, -1};
 int dc[] = {1, 0, -1, 0};
 int r = 0, c = 0, di = 0;
 // Iterate from 0 to R * C - 1
 for (int i = 0; i < R * C; i++)
 {
  ans.push_back(matrix[r]);
  seen[r] = true;
  int cr = r + dr[di];
  int cc = c + dc[di];
  if (0 <= cr && cr < R && 0 <= cc && cc < C && !seen[cr][cc])
  {
   r = cr;
   c = cc;
  }
```

```
 else
 {
  di = (di + 1) % 4;
  r += dr[di];
  c += dc[di];
 }
}
return ans;
}
void spiralPrint(int m, int n, int a[R][C])
{
int i, k = 0, l = 0;
/* k - starting row index
m - ending row index
l - starting column index
n - ending column index
i - iterator
*/
while (k < m && l < n)
{
 /* Print the first row from
 the remaining rows */
 for (i = l; i < n; ++i)
 {
  cout << a[k][i] << " ";
 }
 k++;
 /* Print the last column
 from the remaining columns */
 for (i = k; i < m; ++i)
 {
  cout << a[i][n - 1] << " ";
 }
 n--;
 /* Print the last row from
 the remaining rows */
 if (k < m)
 {
  for (i = n - 1; i >= l; --i)
  {
   cout << a[m - 1][i] << " ";
  }
  m--;
 }
 /* Print the first column from
 the remaining columns */
 if (l < n)
 {
  for (i = m - 1; i >= k; --i)
  {
   cout << a[i][l] << " ";
  }
  l++;
 }
}
```

```
}

// Search an element in a matriix
bool searchMatrix(vector<vector<int>> &matrix, int
               target)
{
 int m = matrix.size(), n = matrix[0].size(), x, y;
 int lo = 0, hi = m * n - 1, mid;
 if (hi == 0)
  return (matrix[0][0] == target);
 while (lo <= hi)
 {
  mid = lo + (hi - lo) / 2;
   x = mid / n;
   y = mid % n;
  if (matrix[x][y] == target)
   return true;
  else if (matrix[x][y] > target)
   hi = mid - 1;
  else
   lo = mid + 1;
 }
 return false;
}

// Find median in a row wise sorted matrix
int binaryMedian(int m[][MAX], int r, int c)
{
 int min = INT_MAX, max = INT_MIN;
 for (int i = 0; i < r; i++)
 {
  // Finding the minimum element
  if (m[i][0] < min)
   min = m[i][0];
  // Finding the maximum element
  if (m[i][c - 1] > max)
   max = m[i][c - 1];
 }
 int desired = (r * c + 1) / 2;
 while (min < max)
 {
  int mid = min + (max - min) / 2;
  int place = 0;
  for (int i = 0; i < r; ++i)
   place += upper_bound(m[i], m[i] + c, mid) -
      m[i];
  if (place < desired)
   min = mid + 1;
  else
   max = mid;
 }
 return min;
}

// Find row with maximum no. of 1's
```

```
int first(bool arr[], int low, int high)
{
 if (high >= low)
 {
  int mid = low + (high - low) / 2;
  if ((mid == 0 || arr[mid - 1] == 0) && arr[mid] == 1) return mid;
  else if (arr[mid] == 0) return first(arr, (mid + 1), high);
  else return first(arr, low, (mid - 1));
 }
 return -1;
}
// Function that returns index of row
// with maximum number of 1s.
int rowWithMax1s(bool mat[R][C])
{
 int max_row_index = 0, max = -1;
 int i, index;
 for (i = 0; i < R; i++)
 {
  index = first(mat[i], 0, C - 1);
  if (index != -1 && C - index > max)
  {
   max = C - index;
   max_row_index = i;
  }
 }
 return max_row_index;
}

// WAY 2:
int rowWithMax1s(bool mat[R][C])
{
 int j, max_row_index = 0;
 j = C - 1;
 for (int i = 0; i < R; i++)
 {
  bool flag = false;
  while (j >= 0 && mat[i][j] == 1)
  {
   j = j - 1;
   flag = true;
  }
  if (flag)
  {
   max_row_index = i;
  }
 }
 if (max_row_index == 0 && mat[0][C - 1] == 0)
  return -1;
 return max_row_index;
}

// Print elements in sorted order using row-column wise sorted matrix

// Rotate 90 Clockwise
```

```
void rotate90Clockwise(int a[N][N])
{
 for (int i = 0; i < N / 2; i++)
 {
  for (int j = i; j < N - i - 1; j++)
  {
   int temp = a[i][j];
   a[i][j] = a[N - 1 - j][i];
   a[N - 1 - j][i] = a[N - 1 - i][N - 1 - j];
   a[N - 1 - i][N - 1 - j] = a[j][N - 1 - i];
   a[j][N - 1 - i] = temp;
  }
 }
}
void rotate90Clockwise(int arr[N][N])
{
 for (int j = 0; j < N; j++)
 {
  for (int i = N - 1; i >= 0; i--)
   cout << arr[i][j] << " ";
  cout << '\n';
 }
}
void rotate(int arr[N][N])
{
 for (int i = 0; i < N; ++i) // transpose
  for (int j = 0; j < i; ++j)
   swap(arr[i][j], arr[j][i]);
 for (int i = 0; i < N; ++i) // reverse
  for (int j = 0; j < N / 2; ++j)
   swap(arr[i][j], arr[i][N - j - 1]);
}

// Reverse a String
// Check whether a String is Palindrome or not
// Find Duplicate characters in a string
// Write a Code to check whether one string is a rotation of another
bool areRotations(string s1, string s2)
{
 if (s1.length() != s2.length())
  return false;
 s1 = s1 + s1;
 return (s1.find(s2) != -1) ? true : false;
}

// Write a Program to check whether a string is a valid shuffle of two strings or not
bool compare(char arr1[], char arr2[])
{
 for (int i = 0; i < MAX; i++)
  if (arr1[i] != arr2[i])
   return false;
 return true;
}
bool search(char *pat, char *txt)
{
```

```
 int M = strlen(pat), N = strlen(txt);
 int countP[MAX] = {0}, countTW[MAX] = {0};
 for (int i = 0; i < M; i++)
 {
  countP[pat[i]]++;
  countTW[txt[i]]++;
 }
 for (int i = M; i < N; i++)
 {
  if (compare(countP, countTW))
   return true;
  (countTW[txt[i]])++;
  countTW[txt[i - M]]--;
 }
 return compare(countP, countTW) ? 1 : 0;
}

// Count and Say
string countAndSay(int n)
{
 string ans = "1";
 n--;
 while (n--)
 {
  string res = ans, subAns;
  for (int i = 0; i < res.size(); i++)
  {
   int count = 1;
   while (i + 1 < res.size() && res[i] == res[i + 1])
   {
    count++;
    i++;
   }
   subAns += to_string(count) + res[i];
  }
  ans = subAns;
 }
 return ans;
}

// Longest Palindrome substring
string longestPalindrome(string s)
{
 int n = s.length();
 if (n == 1) return s;
 bool dp[n][n];
 int start = 0, end = 0;
 for (int g = 0; g < s.length(); g++)
 {
  for (int i = 0, j = g; j < s.length(); i++, j++)
  {
   if (g == 0)
    dp[i][j] = true;
   else if (g == 1)
   {
```

```
    if (s[i] == s[j])
     dp[i][j] = true;
    else
     dp[i][j] = false;
   }
   else
   {
    if (s[i] == s[j] && dp[i + 1][j - 1] == true)
     dp[i][j] = true;
    else
     dp[i][j] = false;
   }
   if (dp[i][j] == true)
   {
    start = i;
    end = g + 1;
   }
  }
 }
 return s.substr(start, end);
}

// Longest Palindromic Subsequence
int longestPalindromeSubseq(string s)
{
 string w = s;
 reverse(s.begin(), s.end());
 int m = s.size();
 int dp[m + 1][m + 1];
 for (int i = 0; i <= m; i++)
  for (int j = 0; j <= m; j++)
   if (i == 0 || j == 0)
    dp[i][j] = 0;
   else if (s[i - 1] == w[j - 1])
    dp[i][j] = 1 + dp[i - 1][j - 1];
   else
    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
 return dp[m][m];
}

// Count of Palindromic substrings
int countSubstrings(string s)
{
 vector<vector<int>> mem(s.size(), vector<int>(s.size(), -1));
 int count = 0;
 for (int i = 0; i < s.size(); ++i)
 {
  for (int j = i; j < s.size(); ++j)
  {
   count += solve(mem, s, i, j);
  }
 }
 return count;
}
int solve(vector<vector<int>> &mem, string &s, int i, int j)
```

```
{
 if (i >= j)
  return 1;
 if (mem[i][j] >= 0)
  return mem[i][j];
 return mem[i][j] = s[i] == s[j] ? solve(mem, s, i + 1, j - 1) : 0;
}
int tabulation(string &s)
{
 vector<vector<int>> tab(s.size(), vector<int>(s.size()));
 int count = 0;
 for (int i = s.size() - 1; i >= 0; --i)
 {
  for (int j = i; j < s.size(); ++j)
  {
   if (i == j) tab[i][j] = 1;
   else if (i+1==j) tab[i][j]= s[i]==s[j]?1:0;
   else
    tab[i][j] = s[i] == s[j] ? tab[i + 1][j - 1] : 0;
   count += tab[i][j];
  }
 }
 return count;
}

// Count Different Palindromic Subsequences
int dp[1001][1001];
long long int f(string &s, int l, int r)
{
 const long long int M = 1e9 + 7;
 if (l > r)
  return 0;
 if (dp[l][r] != -1)
  return dp[l][r];
 if (s[l] == s[r])
  return dp[l][r] = (1 + f(s, l + 1, r) + f(s, l, r - 1)) % M;
 else
  return dp[l][r] = (M + f(s, l + 1, r) + f(s, l, r - 1) - f(s, l + 1, r - 1)) % M;
}

long long int countPS(string str)
{
 memset(dp, -1, sizeof(dp));
 return f(str, 0, str.size() - 1);
}

int countPalindromicSubsequences(string s)
{
 int i, j, n = s.size();
 const long long int M = 1e9 + 7;
 vector<vector<int>> dp(n, vector<int>(n, 0));
 for (i = 0; i < n; i++)
  dp[i][i] = 1;
 for (int k = 1; k < n; k++)
 {
```

```
  for (i = 0; i < n - k; i++)
  {
   j = i + k;
   if (s[i] == s[j])
   {
    int low = i + 1, high = j - 1;
    while (low <= high && s[low] != s[i])
     low++;
    while (low <= high && s[high] != s[j])
     high--;

    if (low > high)
     dp[i][j] = dp[i + 1][j - 1] * 2 + 2;
    else if (low == high)
     dp[i][j] = dp[i + 1][j - 1] * 2 + 1;
    else
     dp[i][j] = dp[i + 1][j - 1] * 2 - dp[low + 1][high - 1];
   }
   else
    dp[i][j] = dp[i][j - 1] + dp[i + 1][j] - dp[i + 1][j - 1];

   dp[i][j] = dp[i][j] < 0 ? dp[i][j] + M : dp[i][j] % M;
  }
 }
 return dp[0][n - 1];
}

// Longest Repeating Subsequence
int LongestRepeatingSubsequence(string s)
{
 int n = s.size();
 vector<vector<int>> dp(n + 1, vector<int>(n + 1, 0));
 for (int i = 1; i <= n; i++)
 {
  for (int j = 1; j <= n; j++)
  {
   if (s[i - 1] == s[j - 1] && i != j)
    dp[i][j] = 1 + dp[i - 1][j - 1];
   else
    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
  }
 }
 return dp[n][n];
}

// Subsets
// Way 1
vector<vector<int>> ans;
void sub(vector<int> &nums, int i, vector<int> temp)
{
 if (i == nums.size())
 {
  ans.push_back(temp);
  return;
 }
```

```
 sub(nums, i + 1, temp);
 temp.push_back(nums[i]);
 sub(nums, i + 1, temp);
}
vector<vector<int>> subsets(vector<int> &nums)
{
 vector<int> temp;
 sub(nums, 0, temp); // or sub(nums, 0, vector<int> {});
 return ans;
}
// Way 2
vector<vector<int>> subsets(vector<int> &nums)
{
 int n = nums.size(), p = 1 << n;
 vector<vector<int>> subs(p);
 for (int i = 0; i < p; i++)
 {
  for (int j = 0; j < n; j++)
  {
   if ((i >> j) & 1)
   {
    subs[i].push_back(nums[j]);
   }
  }
 }
 return subs;
}
// way 3
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();
 }
}
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;
}

// Permutations
vector<vector<int>> result;
vector<vector<int>> permute(vector<int> &nums)
{
 helper(nums, 0, nums.size() - 1);
 return result;
```

```
}
void helper(vector<int> &nums, int l, int r)
{
 if (l == r)
  result.push_back(nums);
 for (int i = l; i <= r; i++)
 {
  swap(nums[l], nums[i]);
  helper(nums, l + 1, r);
  swap(nums[l], nums[i]);
 }
}
// avoid copies case
void help(vector<int> &nums, int i)
{
 if (i == nums.size())
 {
  ans.push_back(nums);
  return;
 }
 unordered_set<int> s;
 for (int j = i; j < nums.size(); j++)
 {
  if (s.find(nums[j]) != s.end())
   continue;
  s.insert(nums[j]);
  swap(nums[j], nums[i]);
  help(nums, i + 1);
  swap(nums[j], nums[i]);
 }
}

// Split the Binary string into two substring with equal 0's and 1's
int maxSubStr(string s)
{
 int c1 = 0, c0 = 0, c = 0;
 for (auto &v : s)
  v == '1' ? c1++ : c0++, c1 == c0 ? c++ : c;

 return c1 != c0 ? -1 : c;
}

// Balanced Parenthesis problem.
bool ispar(string s)
{
 unordered_map<char, int> symbols = {{'[', -1}, {'(', -2}, {'{', -3}, {']', 1}, {')', 2}, {'}', 3}};
 stack<char> st;
 for (char bracket : s)
 {
  if (symbols[bracket] < 0)
  {
   st.push(bracket);
  }
  else
  {
```

```
   if (st.empty())
    return 0;
   char top = st.top();
   st.pop();
   if (symbols[top] + symbols[bracket] != 0)
    return 0;
  }
 }
 if (st.empty())
  return 1;
 else
  return 0;
}

//word break
unordered_map<string, int> m;
bool solve(string A, vector<string> &B, int idx)
{
 if (!A.size()) return 1;
 for (int i = 1; i <= A.size(); i++){
  string l = A.substr(idx, i);
  string r = A.substr(i);
  if (m.find(l) != m.end()){
   if (solve(r, B, 0))
    return 1;
  }
 }
 return 0;
}
int wordBreak(string A, vector<string> &B){
 for (auto v : B)m[v]++;
 return solve(A, B, 0);
}

////////////////////////////////////
bool isMatch(string s1, string s2, int i){
 if (s1.substr(i, s2.size()) == s2) return true;
 return false;
}
int solve(int i, int n, string A, vector<string> &B)
{
 if (i == n) return 1;
 if (dp[i] != -1) return dp[i];
 for (int j = 0; j < B.size(); j++)
  if (isMatch(A, B[j], i))
   if (solve(i + B[j].size(), n, A, B))
    return dp[i] = 1;
 return dp[i] = 0;
}
int wordBreak(string A, vector<string> &B){
 for (int i = 0; i <= A.size(); i++) dp[i] = -1;
 return solve(0, A.size(), A, B);
}

///////////////////////////////////
```

```
bool wordBreak(string s, vector<string>& wordDict) {
 vector<bool> dp(s.size()+1, false);
 dp[0] = true;
 for (int i = 1; i <= s.size(); i++)
  for (int j = 0; j < i; j++)
   if ((dp[j]) && (find(wordDict.begin(), wordDict.end(), s.substr(j, i-j)) != wordDict.end())) {
    dp[i] = true;
    break;
   }
 return dp.back();
}



// Convert a sentence into its equivalent mobile numeric keypad sequence
string printSequence(string S)
{
 string str[] = {"2", "22", "222",
     "3", "33", "333",
     "4", "44", "444",
     "5", "55", "555",
     "6", "66", "666",
     "7", "77", "777", "7777",
     "8", "88", "888",
     "9", "99", "999", "9999"};
 string ans;
 for (int i = 0; i < S.size(); i++)
 {
  if (S[i] == ' ')
   ans += '0';
  ans += (str[S[i] - 'A']);
 }
 return ans;
}

// Count the Reversals-
// reversal means changing '{' to '}'
int countRev(string s)
{
 int n = s.size(), ans = 0;
 if (n % 2 == 1)
  return -1;
 stack<char> st;
 for (int i = 0; i < n; i++)
 {
  if (st.empty() && s[i] == '}')
  {
   ans++;
   st.push('{');
  }
  else
  {
   if (s[i] == '}')
    st.pop();
   else
```

```
    st.push('{');
   }
  }
  int x = st.size();
  x = x / 2;
  ans += x;
  return ans;
}

long long int f(string &s, int l, int r)
{
 const long long int M = 1e9 + 7;
 if (l > r)
  return 0;
 if (dp[l][r] != -1)
  return dp[l][r];
 if (s[l] == s[r])
  return dp[l][r] = (1 + f(s, l + 1, r) + f(s, l, r - 1)) % M;
 else
  return dp[l][r] = (f(s, l + 1, r) + f(s, l, r - 1) - f(s, l + 1, r - 1) + M) % M;
}
long long int countPS(string str)
{
 int dp[1001][1001];
 memset(dp, -1, sizeof(dp));
 return f(str, 0, str.size() - 1);
}


// FInd count of words in a grid
bool isValid(int x, int y, int n, int m, vector<vector<char>> &mat, vector<vector<bool>> &vis, char ch)
{
 if (x < 0 || x >= n || y < 0 || y >= m)
  return (false);
 if (vis[x][y] || mat[x][y] != ch)
  return (false);
 return (true);
}

void dfs(int x, int y, int n, int m, vector<vector<char>> &mat, vector<vector<bool>> &vis, string &target, int i, int
&cnt)
{
 if (i == target.length())
 {
  cnt++;
  return;
 }

 int dx[] = {-1, 0, 1, 0};
 int dy[] = {0, 1, 0, -1};

 for (int j = 0; j < 4; j++)
 {
  int nx = x + dx[j];
  int ny = y + dy[j];
```

```
  if (isValid(nx, ny, n, m, mat, vis, target[i]))
  {
   vis[nx][ny] = true;
   dfs(nx, ny, n, m, mat, vis, target, i + 1, cnt);
   vis[nx][ny] = false;
  }
 }
}

int findOccurrence(vector<vector<char>> &mat, string target)
{
 int n = mat.size(), m = mat[0].size();
 int ans = 0;
 for (int i = 0; i < n; i++)
 {
  for (int j = 0; j < m; j++)
  {
   if (mat[i][j] == target[0])
   {
    vector<vector<bool>> vis(n, vector<bool>(m, false));
    int cnt = 0;
    vis[i][j] = true;
    dfs(i, j, n, m, mat, vis, target, 1, cnt);
    ans += cnt;
   }
  }
 }
 return (ans);
}

// Search a Word in a 2D Grid of characters. only one direction
bool solve(vector<vector<char>> &grid, string &word, int i, int j, int di, int dj, int l)
{
 if (i < 0 || j < 0 || i > m - 1 || j > n - 1 || grid[i][j] != word[l])
 {
  return false;
 }
 if (l == word.size() - 1)
  return true;
 return solve(grid, word, i + di, j + dj, di, dj, l + 1);
}
vector<vector<int>> searchWord(vector<vector<char>> grid, string word)
{
 vector<vector<int>> ans;
 m = grid.size();
 n = grid[0].size();
 int di[] = {-1, -1, -1, 0, 0, 1, 1, 1};
 int dj[] = {-1, 0, 1, -1, 1, -1, 0, 1};
 for (int i = 0; i < m; i++)
 {
  for (int j = 0; j < n; j++)
  {
   for (int k = 0; k < 8; k++)
   {
    if (solve(grid, word, i, j, di[k], dj[k], 0))
```

```
    {
     ans.push_back({i, j});
     break;
    }
   }
  }
 }
 return ans;
}

// Boyer Moore Algorithm for Pattern Searching.

// Converting Roman Numerals to Decimal
int romanToInt(string s)
{
 int ans = 0;
 for (int i = s.size() - 1; i >= 0; i--)
 {
  if (s[i] == 'I')
  {
   if (ans >= 5)
    ans -= 1;
   else
    ans += 1;
  }
  else if (s[i] == 'V')
   ans += 5;
  else if (s[i] == 'X')
  {
   if (ans >= 50)
    ans -= 10;
   else
    ans += 10;
  }
  else if (s[i] == 'L')
   ans += 50;
  else if (s[i] == 'C')
  {
   if (ans >= 500)
    ans -= 100;
   else
    ans += 100;
  }
  else if (s[i] == 'D')
   ans += 500;
  else if (s[i] == 'M')
   ans += 1000;
 }
 return ans;
}

// Integer to Roman
string intToRoman(int N)
{
 const int val[13] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
```

```
 const string rom[13] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};

 string ans = "";
 for (int i = 0; N; i++)
  while (N >= val[i])
   ans += rom[i], N -= val[i];
 return ans;
}

// Longest Common Prefix
string longestCommonPrefix(vector<string> &s)
{
 sort(s.begin(), s.end());
 string ans, s1 = s[0], s2 = s[s.size() - 1];
 for (int i = 0; i < s1.size(); i++)
 {
  if (s1[i] == s2[i])
   ans += s1[i];
  else
   break;
 }
 return ans;
}



// Min Number of Flips
int minFlips(string s)
{
 int c0 = 0, c1 = 0;
 for (int i = 0; i < s.size(); i++)
 {
  if (i % 2 == 0 && s[i] == '1')
   c0++;
  else if (i % 2 != 0 && s[i] == '0')
   c0++;
 }
 for (int i = 0; i < s.size(); i++)
 {
  if (i % 2 == 0 && s[i] == '0')
   c1++;
  else if (i % 2 != 0 && s[i] == '1')
   c1++;
 }
 return min(c0, c1);
}

// Second most repeated string in a sequence

// Minimum Swaps for Bracket Balancing
int minSwaps(string s)
{
 int bal = 0, cnt = 0;
 for (auto c : s)
 {
```

```
  if (bal == 0 && c == ']')
   ++cnt;
  else
   bal += c == '[' ? 1 : -1;
 }
 return (cnt + 1) / 2;
}

// Restore IP Addresses
vector<string> result;
string solution;
vector<string> restoreIpAddresses(string s)
{
 backtracking(s, 0, 0);
 return result;
}
void backtracking(string s, int start, int part)
{
 if (start == s.size() && part == 4)
 {
  result.push_back(solution);
  return;
 }
 for (int i = start; i < s.size(); i++)
 {
  if (part < 4 && i - start < 3 && validIP(s, start, i))
  {
   solution.append(s.substr(start, i - start + 1));
   part++;
   if (part < 4)
    solution.push_back('.');
   backtracking(s, i + 1, part);

   if (part < 4)
    solution.pop_back();
   part--;
   for (int j = 0; j < i - start + 1; j++)
    solution.pop_back();
  }
 }
}
bool validIP(string s, int start, int end)
{
 string temp = s.substr(start, end - start + 1);
 int ip = stoi(temp);
 if (s[start] == '0' && start != end)
  return false;
 else if (ip >= 0 && ip <= 255)
  return true;
 return false;
}

// Rearrange characters in a string such that no two adjacent are same
string reorganizeString(string S)
{
```

```
 vector<int> cnt(26);
 int mostFreq = 0, i = 0;
 for (char c : S)
  if (++cnt[c - 'a'] > cnt[mostFreq])
   mostFreq = (c - 'a');
 if (2*cnt[mostFreq] - 1 > S.size()) return "";
 while (cnt[mostFreq]){
  S[i] = ('a' + mostFreq);
  i += 2;
  cnt[mostFreq]--;
 }
 for (int j = 0; j < 26; j++){
  while (cnt[j]){
   if (i >= S.size()) i = 1;
   S[i] = ('a' + j);
   cnt[j]--;
   i += 2;
  }
 }
 for (auto &v:S) cout<<v;
 return S;
}

// Minimum characters to be added at front to make string palindrome
string shortestPalindrome(string str)
{
 int n = str.length();
 if (n == 0) return "";
 int comlen = 1;
 int end = 0;
 // find LPS from mid
 for (int i = n/2; i >= 0; i--){
  int s=i-1, e=i+1;
  while(s>=0 && str[s]==str[i]) s--;
  while(e<n && str[e]==str[i]) e++;
  while(s>=0 && e<n && str[s]==str[e]){
   s--; e++;
  }
  if (s == -1){
   string left = str.substr(e);
   reverse(left.begin(), left.end());
   return left + str;
  }
 }
 return "";
}
// LPS Method
vector<int> generateLPS(string ns)
{
 int n = ns.size();
 vector<int> lps(n, 0);
 for (int i = 1; i < n; i++){
  int len = lps[i - 1];

  while (len > 0 && ns[i] != ns[len])
```

```
  len = lps[len - 1];

 if (ns[i] == ns[len])
  len++;

 lps[i] = len;
}
return lps;
}
int minCharsforPalindrome(string s)
{
string rev = s;
reverse(rev.begin(), rev.end());
string ns = s + '$' + rev;
auto lps = generateLPS(ns);
return s.size() - lps.back();
}


// Given a sequence of words, print all anagrams together
vector<vector<string>> Anagrams(vector<string> &s){
map<string, vector<int>> m;
for (int i = 0; i < s.size(); i++){
 string v = s[i];
 sort(v.begin(), v.end());
 m[v].push_back(i);
}
vector<vector<string>> ans;
for (auto &v : m){
 vector<string> subAns;
 for (auto &x : v.second){
  subAns.push_back(s[x]);
 }
 ans.push_back(subAns);
}
return ans;
}

// Recursively remove all adjacent duplicates
string removeConsecutiveCharacter(string s)
{
stack<char> st;
for (auto &v : s){
 if (st.empty() || st.top() != v){
  st.push(v);
 }
}
string ans = "";
while (st.size())
{
 char k=st.top(); st.pop();
 ans = k + ans;
}
return ans;
}
```

```
// Function to find Number of customers who could not get a computer
int runCustomerSimulation(int n, char *seq){
 char seen[26] = {0};
 int res = 0, occupied = 0;
 for (int i = 0; seq[i]; i++){
  int ind = seq[i] - 'A';
  if (seen[ind] == 0){
   seen[ind] = 1;
   if (occupied < n){
    occupied++;
    seen[ind] = 2;
   }
   else
    res++;
  }
  else
  {
   if (seen[ind] == 2)
    occupied--;
   seen[ind] = 0;
  }
 }
 return res;
}

// Transform One String to Another Minimum Operation
// The only operation allowed is to put any character from A and insert it at front.
int minOps(string &A, string &B)
{
 int m = A.length(), n = B.length();
 if (n != m) return -1;
 int count[256];
 memset(count, 0, sizeof(count));
 for (int i = 0; i < n; i++)
  count[A[i]]++;
 for (int i = 0; i < n; i++)
  count[B[i]]--;
 for (int i = 0; i < 256; i++)
  if (count[i])
   return -1;

 int res = 0;
 for (int i = n - 1, j = n - 1; i >= 0;)
 {
  while (i >= 0 && A[i] != B[j])
  {
   i--;
   res++;
  }
  if (i >= 0)
  {
   i--;
   j--;
  }
```

```
 }
 return res;
}

// Isomorphic Strings
string f(string s){
 map<char, char> m;
 char C='a';
 for (auto v : s)
  if (m.find(v)==m.end())
   m[v] = C, C++;

 string ans;
 for (int i=0; i<s.size(); i++)
  ans += m[s[i]];
 return ans;
}
bool areIsomorphic(string s1, string s2){
 return f(s1) == f(s2);
}

// Recursively print all sentences that can be formed from list of word lists
void print(vector<vector<string>> &list, vector<vector<string>> &res, vector<string> &s, int start, int N)
{
 if (start >= N){
  res.push_back(s);
  return;
 }
 vector<string> str = list[start];
 for (int i = 0; i < str.size(); i++){
  s.push_back(str[i]);
  print(list, res, s, start + 1, N);
  s.pop_back();
 }
}
vector<vector<string>> sentences(vector<vector<string>> &list)
{
 vector<vector<string>> res;
 vector<string> s;
 print(list, res, s, 0, list.size());
 return res;
}

// Find a Fixed Point (Value equal to index) in a given array

// Square root of a integer
int mySqrt(int x){
 const int E = 1e-6;
 int left = 1, right = x;
 while (left <= right){
  int mid = left + (right - left) / 2;
  if (mid == x / mid) return mid;
  else if (mid < x / mid) left = mid + 1;
  else right = mid - 1;
 }
```

```
 return right;
}

// Search in a rotated sorted array V.IMP
int search(vector<int> &nums, int target)
{
 int low = 0, high = nums.size() - 1;
 while (low <= high)
 {
  int mid = (low + high) / 2;
  if (nums[mid] == target)
   return mid;
  if (nums[mid] >= nums[low])
  {
   // left half is sorted
   if (target >= nums[low] && target < nums[mid])
    high = mid - 1;
   else
    low = mid + 1;
  }
  else
  {
   // right half is sorted
   if (target > nums[mid] && target <= nums[high])
    low = mid + 1;
   else
    high = mid - 1;
  }
 }
 return -1;
}

// Find First and Last Position of Element in Sorted Array
vector<int>searchRange(vector<int> &nums, int target){
 vector<int> ans;
 int i = 0, j = nums.size() - 1;
 int start = -1, end = -1;
 while(i<=j){
  int mid = (i + j) / 2;
  if (nums[mid] == target){
   int temp = mid;
   while (mid>0 && nums[mid-1]==target)
    mid--;
   start = mid;
   while (temp<nums.size()-1 && nums[temp+1]==target)
    temp++;
   end = temp;
   ans.push_back(start);
   ans.push_back(end);
   return ans;
  }
  else if (nums[mid] > target)
   j = mid - 1;
  else
   i = mid + 1;
```

```
 }
 ans.push_back(-1); ans.push_back(-1);
 return ans;
}

// Maximum and minimum of an array using minimum number of comparisons

// Optimum location of point to minimize total
// distance
// Given a set of points as and a line as ax+by+c = 0. We need to find a point on given line for which sum of distance
s from given set of points is minimum.
struct point
{
 int x, y;
 point() {}
 point(int x, int y):x(x), y(y)
  {}
};
struct line
{
 int a, b, c;
 line(int a, int b, int c)
  : a(a), b(b), c(c) {}
};
double dist(double x, double y, point p){
 return sqrt(sq(x - p.x) + sq(y - p.y));
}
double compute(point p[], int n, line l, double X){
 double res = 0;
 double Y = -1 * (l.c + l.a * X) / l.b;
 for (int i = 0; i < n; i++)
  res += dist(X, Y, p[i]);
 return res;
}
double findOptimumCostUtil(point p[], int n, line l)
{ // ternary search
 double low = -1e6;
 double high = 1e6;
 while ((high - low) > EPS)
  {
  double mid1 = low + (high - low) / 3;
  double mid2 = high - (high - low) / 3;
  double dist1 = compute(p, n, l, mid1);
  double dist2 = compute(p, n, l, mid2);
  if (dist1 < dist2)
   high = mid2;
  else
   low = mid1;
  }
 return compute(p, n, l, (low + high) / 2);
}

// Find the repeating and the missing
// way: Use elements as Index and mark the visited places
void printTwoElements(int arr[], int size)
```

```
{
 int i;
 cout << " The repeating element is ";
 for (i = 0; i < size; i++)
 {
  if (arr[abs(arr[i]) - 1] > 0)
   arr[abs(arr[i]) - 1] = -arr[abs(arr[i]) - 1];
  else
   cout << abs(arr[i]) << "\n";
 }
 cout << "and the missing element is ";
 for (i = 0; i < size; i++)
 {
  if (arr[i] > 0)
   cout << (i + 1);
 }
}

// Pivot Min in rotated sorted array
int findMin(vector<int> &nums)
{
 int left = 0, right = nums.size() - 1;
 while (left < right)
 {
  if (nums[left] < nums[right])
   return nums[left];
  int mid = (left + right) / 2;
  if (nums[mid] > nums[right])
   left = mid + 1;
  else if (nums[mid] < nums[left])
   right = mid;
  else
  { // contain duplicates case
   hi--;
  }
 }
 return nums[left];
}

// Smallest Substring Of A String Containing All Unique Characters of Itself
int findSubString(string str)
{
 int len = 0;
 map<int, int> s;
 for (int i = 0; i < str.size(); i++) s[str[i]]++;
 int i = -1, j = -1;
 map<int, int> check;
 while (1){
  bool f1 = 0, f2 = 0;
  while (i<s.size() && check.size()<s.size()){
   i++;
   check[str[i]]++;
   f1 = 1;
  }
  while (j<i && s.size()==check.size()){
```

```
   int plen = i - j;
   len = min(len, plen);
   j++;
   check[str[j]]--;
   f2 = 1;
  }
  if (f1 == 0 && f2 == 0)
   break;
 }
 return len;
}

// Searching in an array where adjacent differ by at most k
int search(int arr[], int n, int x, int k)
{
 int i = 0;
 while (i < n)
 {
  if (arr[i] == x)
   return i;
  i = i + max(1, abs(arr[i] - x) / k);
 }
 cout << "number is not present!";
 return -1;
}

// pair of elements in the array whose difference is N.
bool findPair(int arr[], int size, int n)
{
 map<int, int> m;
 for (int i = 0; i < size; i++)
 {
  if (m.find(arr[i] + n) != m.end() || m.find(arr[i] - n) != m.end())
   return 1;
  m[arr[i]]++;
 }
 return 0;
}

// Four sum
vector<vector<int>> fourSum(vector<int> &nums, int target)
{
 int n = nums.size();
 sort(nums.begin(), nums.end());
 set<vector<int>> sv;
 for (int i = 0; i < n; i++)
 {
  for (int j = i + 1; j < n; j++)
  {
   for (int k = j + 1; k < n; k++)
   {
    int chk = target - (nums[i] + nums[j] + nums[k]);
    if (binary_search(nums.begin() + k + 1, nums.end(), chk))
    {
     vector<int> v;
```

```
    v.push_back(nums[i]);
    v.push_back(nums[j]);
    v.push_back(nums[k]);
    v.push_back(chk);

    sort(v.begin(), v.end());
    sv.insert(v);
   }
  }
 }
 }
 vector<vector<int>> ans(sv.begin(), sv.end());
 return ans;
}

// Count triplet with sum smaller than a given value
long long countTriplets(long long arr[], int n, long long sum)
{
 sort(arr, arr + n);
 long long cnt = 0;
 for (int i = 0; i < n - 2; i++){
  int j = i + 1;
  int k = n - 1;
  while (j < k){
   long long s = arr[i] + arr[j] + arr[k];
   if (s < sum){
    cnt += (k - j);
    j++;
   }
   else k--;
  }
 }
 return cnt;
}

// Product array puzzle- product of all except nums[i].
vector<long long int> productExceptSelf(vector<long long int> &nums, int n){
 vector<long long int> ans(n, 0);
 long long int product = 1, ct0 = 0, idx = 0;
 for (int i = 0; i < n; i++){
  if (!nums[i])
   ct0++, idx = i;
  else
   product *= nums[i];
 }
 if(ct0>1) return ans;
 else if(ct0) ans[idx]=product;
 else{
  for (int i = 0; i < n; i++)
   ans[i] = (product / nums[i]);
 }
 return ans;
}

// Sort array according to count of set bits
```

```
static bool compare(int x, int y){
 return __builtin_popcount(x) > __builtin_popcount(y);
}
void sortBySetBitCount(int arr[], int n){
 sort(arr, arr + n, compare);
}

// Minimum number of swaps required to sort an array
int minSwaps(int arr[], int n)
{
 pair<int, int> arrPos[n];
 for (int i = 0; i < n; i++){
  arrPos[i].first = arr[i];
  arrPos[i].second = i;
 }
 sort(arrPos, arrPos + n);
 vector<bool> vis(n, false);
 int ans = 0;
 for (int i=0; i<n; i++){
  if (vis[i] || arrPos[i].second==i)
   continue;
  int cycle_size = 0;
  int j = i;
  while (!vis[j]){
   vis[j] = 1;
   j = arrPos[j].second;
   cycle_size++;
  }
  if (cycle_size > 0){
   ans += (cycle_size - 1);
  }
 }
 return ans;
}

// Aggressive Cows
bool isPossible(vector<int> &stalls, int minDist, int k){
 int cows = 1;
 int lastCowPosition = stalls[0];
 for (int i = 1; i < stalls.size(); i++)
 {
  if (stalls[i] - lastCowPosition >= minDist)
  {
   cows++;
   lastCowPosition = stalls[i];
   if (cows >= k)
    return true;
  }
 }
 return false;
}
int aggressiveCows(vector<int> &stalls, int k)
{
 int n = stalls.size();
 sort(stalls.begin(), stalls.end());
```

```
 int low = 1, high = stalls[n - 1] - stalls[0];
 int res;
 while (low <= high)
 {
  int mid = (low + high) / 2;
  if (isPossible(stalls, mid, k))
  {
   res = mid;
   low = mid + 1;
  }
  else
   high = mid - 1;
 }
 return res;
}
// Allocate Minimum No of pages
bool isPossible(int arr[], int n, int m, int curr_min)
{
 int studentsRequired = 1;
 int curr_sum = 0;
 for (int i = 0; i < n; i++)
 {
  if (arr[i] > curr_min)
   return false;
  if (curr_sum + arr[i] > curr_min)
  {
   studentsRequired++;
   curr_sum = arr[i];
   if (studentsRequired > m)
    return false;
  }
  else
   curr_sum += arr[i];
 }
 return true;
}
int findPages(int arr[], int n, int m)
{
 long long sum = 0;
 if (n < m)
  return -1;
 for (int i = 0; i < n; i++)
  sum += arr[i];
 int start = 0, end = sum;
 int result = INT_MAX;
 while (start <= end)
 {
  int mid = (start + end) / 2;
  if (isPossible(arr, n, m, mid))
  {
   result = mid;
   end = mid - 1;
  }
  else
   start = mid + 1;
```

```
 }
 return result;
}
// EKO SPOJ
int main(int argc, char *argv[])
{
 int i = 0, max = -1, *array, n, k;
 cin >> n >> k;
 for (i = 0; i < n; i++)
 {
  cin >> array[i];
  if (array[i] > max)
   max = array[i];
 }
 long long int low = 0, high = max, count = 0,
      mid, h = 0;
 while (low <= high)
 {
  mid = (high + low) / 2;
  count = 0;
  for (i = 0; i < n; i++)
  {
   long long int temp =
    array[i] - mid;
   count += (temp > 0 ? temp : 0);
  }
  if (count == k)
  {
   h = mid;
   break;
  }
  else if (count < k)
  {
   high = mid - 1;
  }
  else
  {
   low = mid + 1;
   if (mid > h)
    h = mid;
  }
 }
 cout << h;
 return 0;
}

// The painter's partition problem
int numberOfPainters(int arr[], int n, int maxLen)
{
 int total = 0, numPainters = 1;
 for (int i = 0; i < n; i++)
 {
  total += arr[i];
  if (total > maxLen)
  {
```

```
   total = arr[i];
   numPainters++;
  }
 }
 return numPainters;
}
int partition(int arr[], int n, int k)
{
 int lo = getMax(arr, n);
 int hi = getSum(arr, n);
 while (lo < hi)
 {
  int mid = lo + (hi - lo) / 2;
  int requiredPainters = numberOfPainters(arr, n, mid);
  if (requiredPainters <= k)
   hi = mid;
  else
   lo = mid + 1;
 }
 return lo;
}

// Smallest number with at least n trailing zeroes infactorial
bool isValid(int mid, int n)
{
 int num = mid;
 int cnt5 = 0;
 while (num)
 {
  num /= 5;
  cnt5 += num;
 }
 return (cnt5 >= n);
}

int findNum(int n)
{
 int s = 5, e = 5 * n;
 int ans = 5 * n;
 while (s <= e)
 {
  int mid = s + (e - s) / 2;
  if (isValid(mid, n))
  {
   ans = mid;
   e = mid - 1;
  }
  else s = mid + 1;
 }
 return (ans);
}

// Find the inversion count
int _mergeSort(int arr[], int temp[], int left, int right)
{
```

```
 int mid, inv_count = 0;
 if (right > left)
 {
  mid = (right + left) / 2;
  inv_count += _mergeSort(arr, temp, left, mid);
  inv_count += _mergeSort(arr, temp, mid + 1, right);
  inv_count += merge(arr, temp, left, mid + 1, right);
 }
 return inv_count;
}
int merge(int arr[], int temp[], int left, int mid, int right)
{
 int i = left, j = mid, k = left, inv_count = 0;
 while ((i <= mid - 1) && (j <= right))
 {
  if (arr[i] <= arr[j])
  {
   temp[k++] = arr[i++];
  }
  else
  {
   temp[k++] = arr[j++];
   inv_count = inv_count + (mid - i);
  }
 }
 while (i <= mid - 1)
  temp[k++] = arr[i++];
 while (j <= right)
  temp[k++] = arr[j++];
 for (i = left; i <= right; i++)
  arr[i] = temp[i];
 return inv_count;
}

struct Node
{
 int data;
 struct Node *next;
 Node(int x)
 {
  data = x;
  next = NULL;
 }
};

// Reverse the Linked List. (Both Iterative and recursive)
Node *reverseList(Node *head){
 Node *prev = NULL, *curr = head, *next = NULL;
 while (curr != NULL){
  Node *temp = curr->next;
  curr->next = prev;
  prev = curr;
  curr = temp;
 }
 head = prev;
```

```
 return head;
}
Node *reverseList( Node *head){
 if (!head || head->next == NULL)
  return head;
 Node *temp = reverseList(head->next);
 head->next->next = head;
 head->next = NULL;
 return temp;
}

// Reverse a Linked List in group of Given Size. [Very Imp]
struct node *reverse(struct node *head, int k){
 int count = 0;
 struct node *prev = NULL;
 struct node *curr = head;
 struct node *temp = NULL;
 while (count < k && curr != NULL){
  temp = curr->next;
  curr->next = prev;
  prev = curr;
  curr = temp;
  count++;
 }
 if (curr != NULL){
  head->next = reverse(curr, k);
 }
 return prev;
}

// Detect Loop in linked list
bool detectLoop(Node *head)
{
 int cnt = 0;
 auto t = head;
 map<Node *, int> m;
 while (t != NULL){
  if (m[t] == 1){
   return true;
  }
  m[t]++;
  t = t->next;
 }
 return false;
}
bool detectLoop(Node *head)
{
 if (head == NULL || head->next == NULL)
  return NULL;
 Node *slow = head, *fast = head;
 slow = slow->next;
 fast = fast->next->next;
 while (fast && fast->next){
  if (slow == fast) return 1;
  slow = slow->next;
```

```
  fast = fast->next->next;
 }
 if (slow != fast) return 0;
}

// Remove loop in a linked list.
void removeLoop(Node *head)
{
 if (head == NULL || head->next == NULL)
  return;
 Node *slow = head, *fast = head;
 while (fast->next != NULL && fast->next->next != NULL)
 {
  slow = slow->next;
  fast = fast->next->next;
  if (fast->next == NULL || fast->next->next == NULL)
   return;
  if (slow == fast)
   break;
 }
 slow = head;
 while (slow != fast)
 {
  slow = slow->next;
  fast = fast->next;
 }
 while (slow->next != fast)
 {
  slow = slow->next;
 }
 slow->next = NULL;
 return;

 // Node *low = slow, *high = fast;
 // if (low == head)
 // {
 //  while (high->next != low)
 //  {
 //   high = high->next;
 //  }
 //  high->next = NULL;
 // }
 // else if (low == high)
 // {
 //  low = head;
 //  while (low->next != high->next)
 //  {
 //   low = low->next;
 //   high = high->next;
 //  }
 //  high->next = NULL;
 // }
 // return;
}
```

```
// Find first node of loop in a linked list
Node * detectAndRemoveLoop(Node *head){
 if (head == NULL || head->next == NULL)
  return NULL;
 Node *slow = head, *fast = head;
 slow = slow->next;
 fast = fast->next->next;
 while (fast && fast->next)
 {
  if (slow == fast)
   break;
  slow = slow->next;
  fast = fast->next->next;
 }
 if (slow != fast)
  return NULL;
 slow = head;
 while (slow != fast)
 {
  slow = slow->next;
  fast = fast->next;
 }
 return slow;
}

// Remove Duplicates in a sorted Linked List.
Node *removeDuplicates(Node *head)
{
 Node *cur = head;
 while (cur)
 {
  while (cur->next && cur->data == cur->next->data)
  {
   cur->next = cur->next->next;
  }
  cur = cur->next;
 }
 return head;
}

// Remove duplicates from an unsorted linked list
Node *removeDuplicates(Node *head)
{
 Node *prev = NULL;
 unordered_set<int> s;
 for (Node *curr = head; curr != NULL;)
 {
  if (s.find(curr->data) != s.end())
  {
   Node *temp = curr;
   prev->next = temp->next;
   curr = curr->next;
   delete temp;
  }
  else
```

```
  {
   s.insert(curr->data);
   prev = curr;
   curr = curr->next;
  }
 }
 return head;
}
```

```
// Move the last element to Front in a Linked List.
void rearrange(struct Node **head)
{
 if (!*head || !(*head)->next)
  return;
 struct Node *ptr = *head;
 while (ptr->next->next)
 {
  ptr = ptr->next;
 }
 ptr->next->next = *head;
 *head = ptr->next;
 ptr->next = NULL;
}
```

```
// Add “1” to a number represented as a Linked List.
Node *reverse(Node *&head)
{
 if (head->next == NULL)
  return head;
 Node *temp = reverse(head->next);
 head->next->next = head;
 head->next = NULL;
 return temp;
}
Node *addOne(Node *head)
{
 head = reverse(head);

 int sum = 0;
 int carry = 0;

 sum = head->data + 1;
 head->data = sum % 10;
 carry = sum / 10;

 Node *temp = head->next;
 Node *prev = head;

 while (temp != NULL)
 {

  sum = (temp->data + carry);
  temp->data = sum % 10;
  carry = sum / 10;
```

```
  prev = prev->next;
  temp = temp->next;
 }

 if (carry)
 {
  Node *newNode = new Node(carry);
  prev->next = newNode;
 }
 prev = head;

 head = reverse(prev);
 return head;
}

// Add two numbers represented by linked lists.
struct Node *addTwoLists(struct Node *first, struct Node *second)
{
 // code here
 Node *firstr = reverse(first);
 Node *secondr = reverse(second);
 Node *ans = new Node(0);
 Node *ansh = ans;
 int sum = 0, carry = 0;
 while (firstr || secondr)
 {
  sum = (firstr ? firstr->data : 0) + (secondr ? secondr->data : 0) + carry;
  carry = sum >= 10 ? 1 : 0;
  int add = sum % 10;
  ans->next = new Node(add);

  if (firstr)
  {
   firstr = firstr->next;
  }

  if (secondr)
  {
   secondr = secondr->next;
  }

  ans = ans->next;
 }
 if (carry > 0)
  ans->next = new Node(carry);

 return reverse(ansh->next);
}

// Intersection Point of two Linked Lists.
int intersectPoint(Node *head1, Node *head2)
{
 int m = 0, n = 0;
 Node *t1 = head1, *t2 = head2;
 while (t1 != NULL){
```

```
  t1 = t1->next;
  ++m;
 }
 while (t2 != NULL){
  t2 = t2->next;
  ++n;
 }
 if (m > n){
  int t = m - n;
  while (t--)
   head1 = head1->next;
 }
 else{
  int t = n - m;
  while (t--)
   head2 = head2->next;
 }
 while (head1 != NULL && head2 != NULL)
 {
  if (head1 == head2)
   return head1->data;
  head1 = head1->next;
  head2 = head2->next;
 }
 return -1;
}

int intersectPoint(Node *head1, Node *head2)
{
 Node *ptr1 = head1;
 Node *ptr2 = head2;

 if (ptr1 == NULL || ptr2 == NULL)
  return -1;

 while (ptr1 != ptr2)
 {

  ptr1 = ptr1->next;
  ptr2 = ptr2->next;

  if (ptr1 == ptr2)
   return ptr1->data;

  if (ptr1 == NULL)
   ptr1 = head2;

  if (ptr2 == NULL)
   ptr2 = head1;
 }
 return -1;
}

// Intersection of two Sorted Linked List.
Node *findIntersection(Node *head1, Node *head2)
```

```
{
 Node *head3 = new Node(-1);
 Node *p3 = head3;

 while (head1 != NULL && head2 != NULL)
 {
  if (head1->data == head2->data)
  {
   p3->next = head1;
   head1 = head1->next;
   head2 = head2->next;
   p3 = p3->next;
  }
  else if (head1->data < head2->data)
  {
   head1 = head1->next;
  }
  else if (head1->data > head2->data)
  {
   head2 = head2->next;
  }
 }

 return head3->next;
}

// Find the middle Element of a linked list.
ListNode *middleNode(ListNode *head)
{
 ListNode *slow = head;
 ListNode *fast = head;
 while (fast != NULL && fast->next != NULL)
 {
  slow = slow->next;
  fast = fast->next->next;
 }
 return slow;
}

// Check if a linked list is a circular linked list.
bool isCircular(Node *head)
{
 Node *temp = head;
 while (temp)
 {
  temp = temp->next;
  if (head == temp)
  {
   return 1;
  }
 }
 return 0;
}

// Split a Circular linked list into two halves.
```

```
void splitList(Node *head, Node **head1_ref, Node **head2_ref)
{
 Node *prev = head, *after = head->next;

 while (after != head and after->next != head)
 {

  prev = prev->next;
  after = after->next->next;
 }

 *head1_ref = head;
 *head2_ref = prev->next;
 prev->next = *head1_ref;

 Node *curr = *head2_ref;

 while (curr->next != head)
 {
  curr = curr->next;
 }

 curr->next = *head2_ref;
}

// Merge K sorted Lists
ListNode *merge2Lists(ListNode *l1, ListNode *l2)
{
 if (!l1)
  return l2;
 if (!l2)
  return l1;
 ListNode *head = l1->val <= l2->val ? l1 : l2;
 head->next = l1->val <= l2->val ? merge2Lists(l1->next, l2) : merge2Lists(l1, l2->next);
 return head;
}
ListNode *mergeKLists(vector<ListNode *> &lists)
{
 if (lists.size() == 0)
  return NULL;
 ListNode *head = lists[0];
 for (int i = 1; i < lists.size(); i++)
  head = merge2Lists(head, lists[i]);
 return head;
}

// Merge K sorted Arrays
vector<int> mergeKArrays(vector<vector<int>> arr, int K)
{
 multiset<int> m;
 for (int i = 0; i < K; i++)
 {
  for (int j = 0; j < K; j++)
  {
   m.insert(arr[i][j]);
```

```
  }
 }
 vector<int> v;
 for (auto it = m.begin(); it != m.end(); it++)
 {
  v.push_back(*it);
 }
 return v;
}

// Check if Linked List is Palindrome
struct Node *reverseList(struct Node *head)
{
 Node *cur = head;
 Node *prv = NULL;
 Node *nextptr;
 while (cur != NULL)
 {
  nextptr = cur->next;
  cur->next = prv;
  prv = cur;
  cur = nextptr;
 }
 return prv;
}
struct Node *middle(struct Node *head)
{
 Node *slow = head;
 Node *fast = head;
 while (fast != NULL && fast->next != NULL)
 {
  slow = slow->next;
  fast = fast->next->next;
 }
 return slow;
}
bool isPalindrome(Node *head)
{
 if (head == NULL)
 {
  return true;
 }
 Node *mid = middle(head);
 Node *last = reverseList(mid);
 Node *curr = head;
 while (last != NULL)
 {
  if (last->data != curr->data)
  {
   return false;
  }
  last = last->next;
  curr = curr->next;
 }
 return true;
```

```
}

// Deletion from a Circular Linked List.

// Reverse a Doubly Linked list.
void reverse(Node **head_ref)
{
 Node *temp = NULL;
 Node *current = *head_ref;
 while (current != NULL)
 {
  temp = current->prev;
  current->prev = current->next;
  current->next = temp;
  current = current->prev;
 }
 if (temp != NULL)
  *head_ref = temp->prev;
}

// Flatten a Linked List
Node *sort(Node *l1, Node *l2)
{
 if (l1 == NULL)
  return l2;
 else if (l2 == NULL)
  return l1;
 Node *head = l1->data <= l2->data ? l1 : l2;
 head->bottom = l1->data <= l2->data ? sort(l1->bottom, l2) : sort(l1, l2->bottom);
 return head;
}
Node *flatten(Node *root)
{
 Node *start = root;
 Node *head1 = root->next;
 while (head1 != NULL)
 {
  start = sort(start, head1);
  head1 = head1->next;
 }
 return start;
}

// Clone a linked list with next and random pointer
unordered_map<Node *, Node *> m;
Node *copyRandomList(Node *head)
{
 if (!head) return NULL;
 if (m[head]) return m[head];
 m[head] = new Node(head->val);
 m[head]->next = copyRandomList(head->next);
 m[head]->random = copyRandomList(head->random);
 return m[head];
}
```

```
// Delete nodes which have a greater value on right side
Node *compute(Node *head)
{
 if (!head || !head->next)
  return head;
 auto recurAns = compute(head->next);
 if (head->data < recurAns->data){
  head = recurAns;
  return head;
 }
 head->next = recurAns;
 return head;
}

// Rotate Linked list by N nodes.
ListNode *rotateRight(ListNode *head, int k)
{
 if (!head || !head->next || k == 0)
  return head;
 ListNode *cur = head;
 int len = 1;
 while (cur->next && ++len)
  cur = cur->next;
 cur->next = head;
 k = k % len;
 k = len - k;
 while (k--)
  cur = cur->next;
 head = cur->next;
 cur->next = NULL;
 return head;
}

// Odd Even Linked List
ListNode *oddEvenList(ListNode *head)
{
 if (!head)
  return nullptr;
 ListNode *odd = head, *even = head->next, *evenHead = even;
 while (odd->next && even->next)
 {
  odd->next = even->next;
  odd = odd->next;
  even->next = odd->next;
  even = even->next;
 }
 odd->next = evenHead;
 return head;
}

// Nth node from end of linked list
int lengthLinkedList(Node *head)
{
 Node *temp = head;
 int count = 0;
```

```
 while (temp != NULL)
 {
  count += 1;
  temp = temp->next;
 }
 return count;
}
int getNthFromLast(Node *head, int n)
{
 int l = lengthLinkedList(head);
 int k = l - n;
 Node *temp = head;
 if (k < 0) return -1;
 else if (k == 0) return head->data;
 else
  while (k--){
   temp = temp->next;
  }
 return temp->data;
}

// Level Order Traversal
vector<int> levelOrder(Node *node)
{
 vector<int> ans;
 queue<Node *> q;
 q.push(node);
 while (!q.empty())
 {
  int s = q.size();
  Node *head = q.front();
  q.pop();
  ans.push_back(head->data);
  if (head->left)
   q.push(head->left);
  if (head->right)
   q.push(head->right);
 }
 return ans;
}
// Reverse Level Order Traversal
vector<int> reverseLevelOrder(Node *root)
{
 vector<int> v;
 queue<Node *> q;
 q.push(root);
 while (!q.empty())
 {
  int k = q.size();
  vector<int> temp;
  while (k--)
  {
   Node *t = q.front();
   q.pop();
   temp.push_back(t->data);
```

```
   if (t->left)
    q.push(t->left);
   if (t->right)
    q.push(t->right);
  }
  v.insert(v.begin(), temp.begin(), temp.end());
 }
 return v;
}

// Height of Binary Tree
int height(struct Node *node)
{
 return node ? 1 + max(height(node->left), height(node->right)) : 0;
}

// Diameter of Tree
int diameter(struct node *tree)
{
 if (tree == NULL)
  return 0;
 int lheight = height(tree->left);
 int rheight = height(tree->right);
 int ldiameter = diameter(tree->left);
 int rdiameter = diameter(tree->right);
 return max(lheight + rheight + 1,
      max(ldiameter, rdiameter));
}

int height(struct node *node)
{
 if (node == NULL)
  return 0;
 return 1 + max(height(node->left), height(node->right));
}

int ma;
int func(Node *root)
{
 if (!root)
  return 0;
 int x = func(root->left);
 int y = func(root->right);
 ma = max(ma, x + y + 1);
 return max(x, y) + 1;
}
int diameter(Node *root)
{
 ma = INT_MIN;
 int x = func(root);
 return ma;
}

// Check if 2 trees are mirror or not
bool areMirror(Node *a, Node *b)
```

```
{
 if (a == NULL && b == NULL)
  return true;
 if (a == NULL || b == NULL)
  return false;
 return a->data == b->data &&
     areMirror(a->left, b->right) &&
     areMirror(a->right, b->left);
}

// Invert a Binary tree
void help(TreeNode *&root)
{
 if (!root)
  return;
 help(root->left);
 help(root->right);
 swap(root->left, root->right);
}
TreeNode *invertTree(TreeNode *root)
{
 help(root);
 return root;
}

// Left View of a tree
// Right View of a tree
void help(vector<int> &ans, Node *root, int i)
{
 if (root == NULL) return;
 if (i == ans.size()) ans.push_back(root->data);
 help(ans, root->left, i + 1);
 help(ans, root->right, i + 1);
}
vector<int> leftView(Node *root)
{
 vector<int> ans;
 int i = 0;
 help(ans, root, i);
 return ans;
}

vector<int> leftView(Node *root)
{
 vector<int> ans;
 if (!root)
  return ans;
 queue<Node *> q;
 q.push(root);
 while (!q.empty())
 {
  int n = q.size();
  for (int i = 1; i <= n; i++)
  {
   Node *temp = q.front();
```

```
   q.pop();
   if (i == 1)
    ans.push_back(temp->data);
   if (temp->left != NULL)
    q.push(temp->left);
   if (temp->right != NULL)
    q.push(temp->right);
  }
 }
 return ans;
}

// Top View of a tree
// Bottom View of a tree
void Topview(Node *head, int dis, int level, auto &mp)
{
 if (head == NULL)
  return;
 if (mp.find(dis) == mp.end() || level < mp[dis].second)
 {
  mp[dis] = {head->data, level};
 }
 Topview(head->left, dis - 1, level + 1, mp);
 Topview(head->right, dis + 1, level + 1, mp);
}
vector<int> topView(Node *head)
{
 map<int, pair<int, int>> mp;
 Topview(head, 0, 0, mp);
 vector<int> ans;

 for (auto it : mp)
 {
  ans.push_back(it.second.first);
 }
 return ans;
}
vector<int> topView(Node *root)
{
 map<int, int> mp;
 queue<pair<Node *, int>> q;
 q.push({root, 0});
 while (!q.empty())
 {
  auto p = q.front();
  q.pop();
  Node *node = p.first;
  int line = p.second;

  if (mp.find(line) == mp.end())
   mp[line] = node->data;

  if (node->left)
   q.push({node->left, line - 1});
  if (node->right)
```

```
  q.push({node->right, line + 1});
 }
 vector<int> ans;
 for (auto it : mp)
  ans.push_back(it.second);
 return ans;
}

// Zig-Zag traversal of a binary tree
vector<int> zigZagTraversal(Node *root)
{
 vector<int> v;
 bool flag = 0;
 queue<Node *> q;
 q.push(root);
 while (!q.empty())
 {
  int k = q.size();
  vector<int> temp;
  while (k--)
  {
   Node *t = q.front();
   q.pop();
   temp.push_back(t->data);
   if (t->left)
    q.push(t->left);
   if (t->right)
    q.push(t->right);
  }
  if (!flag)
  {
   v.insert(v.end(), temp.begin(), temp.end());
  }
  else
  {
   reverse(temp.begin(), temp.end());
   v.insert(v.end(), temp.begin(), temp.end());
  }
  flag = !flag;
 }
 return v;
}

// Diagonal Traversal of Binary Tree
void diagonalPrintUtil(Node *root, int d, map<int, vector<int>> &diagonalPrint)
{
 if (!root)return;
 diagonalPrint[d].push_back(root->data);
 diagonalPrintUtil(root->left, d + 1, diagonalPrint);
 diagonalPrintUtil(root->right, d, diagonalPrint);
}
void diagonalPrint(Node *root)
{
 map<int, vector<int>> diagonalPrint;
 diagonalPrintUtil(root, 0, diagonalPrint);
```

```
 cout << "Diagonal Traversal of binary tree : \n";
 for (auto it : diagonalPrint)
 {
  vector<int> v = it.second;
  for (auto it : v)
   cout << it << " ";
  cout << endl;
 }
}

// BOUNDARY TRAVERSAL
void printLeaves(Node *root)
{
 if (!root) return;
 printLeaves(root->left);
 if (!(root->left) && !(root->right))
  cout << root->data << " ";
 printLeaves(root->right);
}
void printBoundaryLeft(Node *root)
{
 if (!root) return;
 cout << root->data << " ";
 if(root->left) printBoundaryLeft(root->left);
 else if(root->right) printBoundaryLeft(root->right);
}
void printBoundaryRight(Node *root)
{
 if (!root) return;
 if (root->right) printBoundaryRight(root->right);
 else if (root->left) printBoundaryRight(root->left);
 cout << root->data << " ";
}
void printBoundary(Node *root)
{
 if (root == nullptr) return;
 cout << root->data << " ";
 printBoundaryLeft(root->left);
 printLeaves(root->left);
 printLeaves(root->right);
 printBoundaryRight(root->right);
}

// Check if a tree is balanced or not
int height(Node *root)
{
 if (!root) return 0;
 return 1 + max(height(root->left), height(root->right));
}
bool isBalanced(Node *root)
{
 if (!root) return 1;
 int diff = abs(height(root->left) - height(root->right));
 if (diff > 1) return 0;
 return (isBalanced(root->left) &&
```

```
   isBalanced(root->right));
}

int dfsHeight(Node *root)
{
 if (root == NULL) return 0;
 int leftHeight = dfsHeight(root->left);
 if (leftHeight == -1) return -1;
 int rightHeight = dfsHeight(root->right);
 if (rightHeight == -1) return -1;
 if (abs(leftHeight - rightHeight) > 1) return -1;
 return max(leftHeight, rightHeight) + 1;
}
bool isBalanced(Node *root){
 return dfsHeight(root) != -1;
}

// Construct Binary Tree from String with Bracket Representation
string tree2str(TreeNode *t)
{
 if (t == NULL)
  return "";
 string s = to_string(t->val);
 if (t->left)
  s += '(' + tree2str(t->left) + ')';
 else if (t->right)
  s += "()";
 if (t->right)
  s += '(' + tree2str(t->right) + ')';
 return s;
}

// Flatten Binary Tree to Linked List
void flatten(TreeNode *root)
{
 if (root) {
  TreeNode *temp = root->right;
  root->right = root->left;
  root->left = nullptr;
  TreeNode *node = root;

  while (node->right)
   node = node->right;

  node->right = temp;
  flatten(root->right);
 }
 return;
}

// Populating Next Right Pointers in Each Node
Node *connect(Node *root)
{
 if (!root)
  return root;
```

```
 queue<Node *> q;
 q.push(root);
 while (!q.empty())
 {
  int n = q.size();
  for (int i = 0; i < n; i++)
  {
   Node *x = q.front();
   q.pop();
   if(i != n - 1)
    x->next = q.front();
   if(x->left) q.push(x->left);
   if(x->right) q.push(x->right);
  }
 }
 return root;
}
// recursive approach!!!!
void solve(Node *l, Node *r)
{
 if (l == NULL || r == NULL)
  return;

 l->next = r;
 r->next = NULL;

 solve(l->left, l->right);
 solve(l->right, r->left);
 solve(r->left, r->right);
}
Node *connect(Node *root)
{
 if (root == NULL)
  return NULL;
 if (root->left == NULL)
  return root;

 solve(root->left, root->right);
 return root;
}

// Convert Binary tree into Sum tree
int solve(Node *root)
{
 if (!root) return 0;
 int x = root->data;
 int l = solve(root->left);
 int r = solve(root->right);
 root->data = l + r;
 return l + r + x;
}
void toSumTree(Node *node)
{
 int x = solve(node);
}
```

```
// Check if Binary tree is Sum tree or not
int f(Node *root)
{
 if (!root) return 0;
 if (!root->left && !root->right)
  return root->data;
 int l = f(root->left);
 if (l == -1) return -1;
 int r = f(root->right);
 if (r == -1) return -1;
 if (root->data != l + r)
  return -1;
 else
  return root->data + l + r;
}
bool isSumTree(Node *root)
{
 f(root);
 if (f(root) == -1)
  return 0;
 return 1;
}

// Find minimum swaps required to convert a Binary tree into BST

// Check if all leaf nodes are at same level or not
int level = -1, flag = 1;
void f(Node *root, int l)
{
 if (!root)
  return;
 if (!root->left && !root->right)
 {
  if (level == -1)
   level = l;
  else if (level != l)
  {
   flag = 0;
  }
 }
 f(root->left, l + 1), f(root->right, l + 1);
}
bool check(Node *root)
{
 f(root, 0);
 return flag;
}

// Duplicate subtree in Binary Tree
map<string, int> m;
string formSubtree(Node *root)
{
 if (root == NULL) return "$";
 string s = "";
```

```
 if (root->right == NULL && root->left == NULL){
  s = to_string(root->data);
  return s;
 }
 s = s + to_string(root->data);
 s = s + formSubtree(root->left);
 s = s + formSubtree(root->right);
 m[s]++;
 return s;
}
int dupSub(Node *root)
{
 formSubtree(root);
 for (auto x : m)
  if (x.second >= 2)
   return true;
 return false;
}

// Sum of Nodes on the Longest path from root to leaf node
vector<int> solve(Node *root)
{
 if (!root)
  return {0, 0};
 vector<int> a = solve(root->left);
 vector<int> b = solve(root->right);
 if (a[0] > b[0])
  return {a[0] + 1, a[1] + root->data};
 if (a[0] < b[0])
  return {b[0] + 1, b[1] + root->data};
 else
  return {a[0] + 1, max(a[1], b[1]) + root->data};
}
int sumOfLongRootToLeafPath(Node *root)
{
 vector<inr> ans = solve(root);
 return ans[1];
}

// Largset Subtree in a Sum
int ma = 0;
int func(Node *root)
{
 if (!root)
  return 0;
 int l = func(root->left);
 int r = func(root->right);
 ma = max(ma, l + r + root->data);
 return l + r + root->data;
}

// LCA in Binary tree
Node *lca(Node *root, int n1, int n2)
{
 if (!root) return NULL;
```

```
 if (root->data == n1 || root->data == n2)
  return root;
 Node *l1 = lca(root->left, n1, n2);
 Node *l2 = lca(root->right, n1, n2);
 if (l1 && l2) return root;
 if (l1) return l1;
 else return l2;
}

// Identical Trees
bool identicalTrees(BinaryTreeNode<int> *root1, BinaryTreeNode<int> *root2)
{
 if (root1 == NULL && root2 == NULL)
  return true;
 if (root1 == NULL || root2 == NULL)
  return false;
 return ((root1->data == root2->data) && identicalTrees(root1->left, root2->left) && identicalTrees(root1->right, ro
ot2->right));
}

//    Tree Isomorphism Problem
bool isIsomorphic(Node *root1, Node *root2)
{
 if(!root1 and !root2) return 1;
 if(!root1 or !root2) return 0;
 if(root1->data!=root2->data) return 0;
 bool a = isIsomorphic(root1->left, root2->left) and
    isIsomorphic(root1->right, root2->right);
 bool b = isIsomorphic(root1->right, root2->left) and
    isIsomorphic(root1->left, root2->right);
 return a || b;
}

// Find distance between 2 nodes in a Binary tree
int solve(Node *root, int val)
{
 if(!root) return 0;
 if(root->data == val) return 1;
 int a = solve(root->left, val);
 int b = solve(root->right, val);
 if (!a and !b) return 0;
 else return a + b + 1;
}
int findDist(Node *root, int a, int b)
{
 Node *LCA = lca(root, a, b);
 int x = solve(LCA, a);
 int y = solve(LCA, b);
 return x + y - 2;
}

// Print all "K" Sum paths in a Binary tree
void func(Node *root, vector<int> &path, int k)
{
 if (!root) return;
```

```
 path.push_back(root->data);
 func(rooot->left, path, k);
 func(root->right, path, k);
 int f = 0;
 for (int j = path.size() - 1; j >= 0; j--)
 {
  f += path[j];
  if (f == k){
   for (int m = j; m < path.size(); m++)
    cout << path[m] << " ";
   cout << endl;
  }
 }
 path.pop_back();
}

// Kth Ancestor of node in a Binary tree
Node *kthAncestorDFS(Node *root, int node, int &k)
{
 if (!root)
  return NULL;
 if (root->data == node || kthAncestorDFS(root->left, node, k) || kthAncestorDFS(root->right, node, k)){
  if (k > 0) k--;
  else if (k == 0) {
   cout << "Kth ancestor is: " << root->data;
   return NULL;
  }
  return root;
 }
}

// Fina a value in a BST
struct node *search(struct node *root, int key)
{
 if (root == NULL || root->key == key)
  return root;
 if (root->key < key)
  return search(root->right, key);
 return search(root->left, key);
}

// INSERT NODE IN BST
Node *insert(Node *root, int value)
{
 if (!root)
 {
  Node *x = new Node(value);
  return x;
 }
 if (value > root->data)
 {
  root->right = insert(root->right, value);
 }
 else
 {
```

```
  root->left = insert(root->left, value);
 }
 return root;
}

// DELETE NODE IN BST
int findminfromright(TreeNode *root)
{
 while (root->left != nullptr)
  root = root->left;
 return root->val;
}
TreeNode *deleteNode(TreeNode *root, int key)
{
 if (root == nullptr)
  return root;
 else if (root->val > key)
  root->left = deleteNode(root->left, key);
 else if (root->val < key)
  root->right = deleteNode(root->right, key);
 else
 {
  if (root->right == nullptr)
  {
   return root->left;
  }
  else if (root->left == nullptr)
  {
   return root->right;
  }
  else
  {
   root->val = findminfromright(root->right);
   root->right = deleteNode(root->right, root->val);
  }
 }
 return root;
}

// Find min and max value in a BST
int minValue(Node *root)
{
 if (root == NULL)
  return -1;
 Node *cur = root;
 while (cur->left != NULL)
  cur = cur->left;
 return cur->data;
}

// Find inorder successor and inorder predecessor in a BST
Node *inpre(Node *root)
{
 Node *p = root->left;
 while (p->right)
```

```
  p = p->right;
 return p;
}
Node *insuc(Node *root)
{
 Node *p = root->right;
 while (p->left)
  p = p->left;
 return p;
}
void findPreSuc(Node *root, Node *&pre, Node *&suc, int key)
{
 if (!root) return;
 if (root->key == key) {
  if (root->left)
   pre = inpre(root);
  if (root->right)
   suc = insuc(root);
  return;
 }
 if (key > root->key)
 {
  pre = root;
  findPreSuc(root->right, pre, suc, key);
 }
 else if (key < root->key)
 {
  suc = root;
  findPreSuc(root->left, pre, suc, key);
 }
}

// Check if tree is BST or Not
void func(Node *root, Node *&prev, int &f){
 if (!root) return;
 func(root->left, prev, f);
 if (prev != NULL and root->data <= prev->data)
 {
  f = 0;
  return;
 }
 prev = root;
 func(root->right, prev, f);
}
bool isBST(Node *root)
{
 int f = 1;
 Node *prev = NULL;
 func(root, prev, f);
 return f;
}

bool isValidBSTHelper(TreeNode *root, long min, long max)
{
 if (root == NULL) return true;
```

```
 if (root->val > min && root->val < max)
 {
  return isValidBSTHelper(root->left, min, root->val) &&
      isValidBSTHelper(root->right, root->val, max);
 }
 return false;
}
bool isValidBST(TreeNode *root)
{
 return isValidBSTHelper(root, LONG_MIN, LONG_MAX);
}

// Check if given graph is tree or not. Graph Valid Tree
bool dfs(vector<vector<int>> &graph, int v, int par, vector<bool> &vis)
{
 vis[v] = 1;
 for (auto it : graph[v])
  if (!vis[it])
   if (dfs(graph, it, v, vis))
    return true;
   else if (it != par)
    return true;
 return false;
}
bool checkgraph(vector<vector<int>> edges, int n, int m)
{
 vector<vector<int>> graph(n);
 for (int i = 0; i < m; i++)
 {
  graph[edges[i][0]].push_back(edges[i][1]);
  graph[edges[i][1]].push_back(edges[i][0]);
 }
 vector<bool> vis(n, 0);
 int numberOfComponents = 1;
 for (int i = 0; i < n; i++)
 {
  if (!vis[i])
  {
   if (dfs(graph, i, -1, vis))
    return false;
   numberOfComponents++;
  }
 }
 if (numberOfComponents >= 2)
  return false;
 return true;
}

// Check if all levels of two trees are anagrams or not.
bool areAnagrams(Node *root1, Node *root2)
{
 if (root1 == NULL && root2 == NULL)
  return true;
 if (root1 == NULL || root2 == NULL)
  return false;
```

```
 queue<Node *> q1, q2;
 q1.push(root1);
 q2.push(root2);
 while (1)
 {
  int n1 = q1.size(), n2 = q2.size();
  if (n1 != n2)
   return false;
  if (n1 == 0) break;
  vector<int> curr_level1, curr_level2;
  while (n1 > 0)
  {
   Node *node1 = q1.front();
   q1.pop();
   if (node1->left != NULL)
    q1.push(node1->left);
   if (node1->right != NULL)
    q1.push(node1->right);
   n1--;

   Node *node2 = q2.front();
   q2.pop();
   if (node2->left != NULL)
    q2.push(node2->left);
   if (node2->right != NULL)
    q2.push(node2->right);

   curr_level1.push_back(node1->data);
   curr_level2.push_back(node2->data);
  }
  sort(curr_level1.begin(), curr_level1.end());
  sort(curr_level2.begin(), curr_level2.end());
  if (curr_level1 != curr_level2)
   return false;
 }
 return true;
}

// Populate Inorder successor of all nodes
void solve(Node 8root, Node *&prev)
{
 if (!root) return;
 solve(root->left, prev);
 if (prev != NULL)
 {
  prev->next = root;
 }
 prev = root;
 solve(root->right, prev);
}
void populateNext(Node *root)
{
 Node *prev = NULL;
 solve(root, prev);
}
```

```
// LCA of BST
Node *LCA(Node *root, int n1, int n2)
{
 if (!root) return NULL;
 if (n1 < root->data and n2 < root->data)
  return LCA(root->left, n1, n2);
 else if (n1 > root->data and n2 > root->data)
  return LCA(root->right, n1, n2);
 else return root;
}

// Sorted Array to BST
TNode *sortedArrayToBST(int arr[], int start, int end)
{
 if (start > end)
  return NULL;
 int mid = (start + end) / 2;
 TNode *root = newNode(arr[mid]);
 root->left = sortedArrayToBST(arr, start, mid - 1);
 root->right = sortedArrayToBST(arr, mid + 1, end);
 return root;
}


// Convert Binary tree into BST
void helper(Node *root, vector<int> &v)
{
 if (!root) return;
 v.push_back(root->data);
 helper(root->left, v);
 helper(root->right, v);
}
void h(Node *&root, vector<int> &v, int &i)
{
 if (!root) return;
 h(root->left, v, i);
 root->data = v[i++];
 h(root->right, v, i);
}
Node *binaryTreeToBST(Node *root)
{
 vector<int> v;
 helper(root, v);
 sort(v.begin(), v.end());

 int i = 0;
 h(root, v, i);
 return root;
}
// Convert a normal BST into a Balanced BST

// Merge two BST
void inorder(Node *root, vector<int> &a)
{
```

```
 if (!root) return;
 if (root->left) inorder(root->left, a);
 a.push_back(root->data);
 if (root->right) inorder(root->right, a);
}
vector<int> merge(Node *root1, Node *root2)
{
 vector<int> a, b, ans;
 inorder(root1, a), inorder(root2, b);
 // two-ptr or gap method
 int i = 0, j = 0;
 while (i < a.size() || j < b.size())
 {
  if (i == a.size())
  {
   ans.push_back(b[j]);
   j++;
   continue;
  }
  if (j == b.size())
  {
   ans.push_back(a[i]);
   i++;
   continue;
  }
  if (a[i] < b[j])
  {
   ans.push_back(a[i]);
   i++;
   continue;
  }
  ans.push_back(b[j]);
  j++;
  continue;
 }
 return ans;
}

// Find Kth largest, Smallest element in a BST
void helper(Node *root, int &k, int &item)
{
 if (!root) return;
 helper(root->right, k, item);
 if (k == 1) item = root->data;
 k--;
 helper(root->left, k, item);
}
int kthLargest(Node *root, int K)
{
 int klarge = -1;
 helper(root, K, klarge);
 return klarge;
}

// Serialize and deserialize Binary Tree
```

```
string serialize(TreeNode *root)
{
 if (!root) return "NULL,";
 return to_string(root->val) + "," + serialize(root->left) + serialize(root->right);
}
TreeNode *deserialize(string data){
 queue<string> q;
 string s;
 for (int i = 0; i < data.size(); i++){
  if (data[i] == ','){
   q.push(s);
   s = "";
   continue;
  }
  s += data[i];
 }
 if (s.size() != 0) q.push(s);
 return deserialize_helper(q);
}
TreeNode *deserialize_helper(queue<string> &q)
{
 string s = q.front();
 q.pop();
 if (s == "NULL") return NULL;
 TreeNode *root = new TreeNode(stoi(s));
 root->left = deserialize_helper(q);
 root->right = deserialize_helper(q);
 return root;
}

// Ceil and Floor of BST
vector<int> searchRange(vector<int> &nums, int target)
{
 vector<int> res{};
 res.push_back(floor(nums, 0, nums.size() - 1, target));
 res.push_back(ceil(nums, 0, nums.size() - 1, target));
 return res;
}
int floor(vector<int> &nums, int left, int right, int target)
{
 if (left > right) return -1;
 else if (left == right) return nums[left] == target ? left : -1;
 else
 {
  int mid = left + (right - left) / 2;
  if (target == nums[mid])
  {
   int tmp = floor(nums, left, mid - 1, target);
   return tmp == -1 ? mid : tmp;
  }
  else if (target < nums[mid])
   return floor(nums, left, mid - 1, target);
  else
   return floor(nums, mid + 1, right, target);
 }
```

```
}
int ceil(vector<int> &nums, int left, int right, int target)
{
 if (left > right)
  return -1;
 else if (left == right)
  return nums[left] == target ? left : -1;
 else
 {
  int mid = left + (right - left) / 2;
  if (target == nums[mid])
  {
   int tmp = ceil(nums, mid + 1, right, target);
   return tmp == -1 ? mid : tmp;
  }
  else if (target < nums[mid])
   return ceil(nums, left, mid - 1, target);
  else
   return ceil(nums, mid + 1, right, target);
 }
}

// BST Iterator
vector<int> nodes;
int curr = 0;
BSTIterator(TreeNode *root){
 inOrder(root);
}

int next()
{
 curr++;
 return nodes[curr - 1];
}

bool hasNext()
{
 return curr != nodes.size();
}
void inOrder(TreeNode *root)
{
 if (!root)
  return;
 inOrder(root->left);
 nodes.push_back(root->val);
 inOrder(root->right);
}

// Count pairs from 2 BST whose sum is equal to given value "X"
void inorder(Node *root, unordered_map<int, int> &m)
{
 if (!root)
  return;
 inorder(root->left, m);
 m[root->data]++;
```

```
 inorder(root->right, m);
}
int countPairs(Node *root1, Node *root2, int x)
{
 unordered_map<int, int> m1, m2;
 inorder(root1, m1);
 inorder(root2, m2);
 int ans = 0;
 for (auto it : m1)
  if (m2[x - it.first])
   ans++;
 return ans;
}

// Count BST ndoes that lie in a given range
void inorder(Node *root, int l, int h, vector<int> &v)
{
 if (root == NULL)
  return;
 inorder(root->left, l, h, v);
 if (root->data >= l && root->data <= h)
  v.push_back(root->data);
 inorder(root->right, l, h, v);
}
int getCount(Node *root, int l, int h)
{
 vector<int> v;
 inorder(root, l, h, v);
 return v.size();
}

// Print Ancestors of a given node in Binary Tree
bool printAncestors(struct node *root, int target)
{
 if(root == NULL) return false;
 if(root->data==target) return true;
 if (printAncestors(root->left, target) || printAncestors(root->right, target)){
  cout << root->data << " ";
  return true;
 }
 return false;
}
// Print Ancestors of a given node in Binary Tree
int solve(Node *root, int K, vector<int> &v)
{
 if (root == NULL) return 0;
 if (root->data == K) return 1;
 int lh = solve(root->left, K, v);
 int rh = solve(root->right, K, v);
 if (lh == 1 or rh == 1){
  v.push_back(root->data);
  return 1;
 }
 return 0;
}
```

```
vector<int> Ancestors(struct Node *root, int target)
{
 vector<int> v;
 if (root == NULL) return v;
 solve(root, target, v);
 return v;
}

// Print all "K" Sum paths in a Binary tree
map<int, int> mp;
int ans = 0;
void dfs(Node *root, int k, int curr)
{
 if (!root) return;
 if (curr + root->data == k) ans++;
 ans += mp[curr + root->data - k];

 mp[curr + root->data]++;
 dfs(root->left, k, curr + root->data);
 dfs(root->right, k, curr + root->data);
 mp[curr + root->data]--;

 return;
}
int sumK(Node *root, int k)
{
 dfs(root, k, 0);
 return ans;
}

// Check whether BST contains Dead End
bool find_ans(Node *curr, int start, int end)
{
 if (curr == NULL) return false;
 if (start == end) return true;

 bool l = find_ans(curr->left, start, curr->data - 1);
 bool r = find_ans(curr->right, curr->data + 1, end);

 return (l || r);
}
bool isDeadEnd(Node *root)
{
 int start = 1, end = INT_MAX;
 return find_ans(root, start, end);
}

// Activity Selection Problem
int maxMeetings(int start[], int end[], int n)
{
 int answer = 1;
 vector<pair<int, int>> v;
 for (int i = 0; i < n; i++)
  v.push_back({end[i], start[i]});
```

```
 sort(v.begin(), v.end());
 int en = v[0].first;

 for (int i = 1; i < n; i++)
 {
  if (v[i].second > en)
  {
   answer += 1;
   en = v[i].first;
  }
 }
 return answer;
}

// Minimum Platforms Problem
int findPlatforms(int arr[], int dep[], int n){
 sort(arr, arr + n);
 sort(dep, dep + n);
 int pt_no = 1, j = 0, i = 1;
 while (i < n){
  if (arr[i] <= dep[j]) pt_no++;
  else j++;
  i++;
 }
 return pt_no;
}

// Job Sequencing Problem
bool comp(Job a, Job b){
 return a.profit > b.profit;
}
vector<int> JobScheduling(Job arr[], int n)
{
 sort(arr, arr + n, comp);
 bool done[n] = {0}; // represent days
 int day = 0, profit = 0;
 for (int i = 0; i < n; i++)
 {
  for (int j = min(n, arr[i].dead) - 1; j >= 0; j--)
  {
   if (done[j] == false)
   {
    day += 1;
    profit += arr[i].profit;
    done[j] = true;
    break;
   }
  }
 }
 return {day, profit};
}

// Fractional Knapsack Problem
double fractionalKnapsack(int W, Item arr[], int n)
{
```

```
 sort(arr, arr + n, [](auto &a, auto &b)
 { return a.value * b.weight > b.value * a.weight; });
 double value = 0;
 for (int i = 0; i < n; i++)
 {
  auto &e = arr[i];
  if (e.weight < W)
  {
   value += e.value;
   W -= e.weight;
  }
  else
  {
   value += double(e.value) * W / e.weight;
   break;
  }
 }
 return value;
}

// Greedy Algorithm to find Minimum number of Coins
vector<int> v;
void minmum(int n, vector<int> currency)
{
 if (n == 0) return;
 for (int i = 9; i >= 0; i--){
  if (n >= currency[i]){
   v.push_back(currency[i]);
   minmum(n - currency[i], currency);
  }
 }
}
vector<int> minPartition(int N)
{
 vector<int> currency = {1, 2, 5, 10, 20, 50, 100, 200, 500, 2000};
 minmum(N, currency);
 return v;
}

// Buy Maximum Stocks if i stocks can be bought on i-th day
int buyMaximumProducts(int n, int k, int price[]){
 vector<pair<int, int>> v;
 for (int i = 0; i < n; ++i)
  v.push_back({price[i], i + 1});
 sort(v.begin(), v.end());
 int ans = 0;
 for (int i = 0; i < n; ++i){
  ans += min(v[i].second, k / v[i].first);
  k -= v[i].first * min(v[i].second, (k / v[i].first));
 }
 return ans;
}

// Find the minimum and maximum amount to buy all N candies
```

```
// Minimize Cash Flow among a given set of friends who have borrowed money from each other
void minCashFlowRec(int amount[])
{
 int mxCredit = getMax(amount), mxDebit = getMin(amount);
 if (amount[mxCredit] == 0 && amount[mxDebit] == 0) return;

 int min = minOf2(-amount[mxDebit], amount[mxCredit]);
 amount[mxCredit] -= min; amount[mxDebit] += min;
 cout <<"Person "<<mxDebit<< " pays " << min << " to "<< "Person " << mxCredit << endl;
 minCashFlowRec(amount);
}

// Minimum Cost to cut a board into squares
void solve()
{
 int t;
 cin >> t;
 while (t--)
 {
  int m, n;
  cin >> m >> n;
  m--, n--;
  int x[m], y[n];
  for (int i = 0; i < m; i++) cin >> x[i];
  for (int j = 0; j < n; j++) cin >> y[j];
  sort(x, x + m, greater<int>());
  sort(y, y + n, greater<int>());
  int horizontal_count = 1, vertical_count = 1;
  int answer = 0, i = 0, j = 0;
  while (i < m and j < n){
   if (x[i] > y[j]){
    answer += x[i] * vertical_count;
    horizontal_count++;
    i++;
   }
   else{
    answer += y[j] * horizontal_count;
    vertical_count++;
    j++;
   }
  }
  while (i < m){
   answer += x[i] * vertical_count;
   i++;
  }
  while (j < n){
   answer += y[j] * horizontal_count;
   j++;
  }
  cout << answer;
 }
}

// Check if it is possible to survive on Island
// Find maximum meetings in one room
```

```
// Maximum product subset of an array
long long maxProduct(vector<int> a, int n)
{
 if (!a.size())
  return 0;
 long long ans = a[0], mxP = a[0], mnP = a[0];
 for (int i = 1; i < n; i++)
 {
  if (a[i] < 0)
   swap(mxP, mnP);
  mxP = max(mxP * a[i], (long long)(a[i]));
  mnP = min(mnP * a[i], (long long)(a[i]));
  ans = max(mxP, ans);
 }
 return ans;
}

// Maximize array sum after K negations
long long int maximizeSum(long long int a[], int n, int k)
{
 sort(a, a + n);
 int i = 0;
 while (k > 0)
 {
  if (a[i] < 0)
  {
   a[i] = a[i] * (-1);
   i++;
  }
  else
  {
   break;
  }
  k--;
 }
 sort(a, a + n);
 while (k > 0)
 {
  a[0] = a[0] * (-1);
  k--;
 }
 int sum = 0;
 sum = accumulate(a, a + n, sum);
 return sum;
}

// Maximize the sum of arr[i]*i
int Maximize(int a[], int n)
{
 sort(a, a + n);
 long long sum = 0;
 long long int mod = 1e9 + 7;
 for (long long i = 0; i < n; i++)
  sum += (a[i] * i);
```

```
 int ans = (int)(sum % mod);
 return ans;
}

// Maximum sum of absolute difference of an array
int MaxSumDifference(int a[], int n)
{
 vector<int> f;
 sort(a, a + n);
 for (int i = 0; i < n / 2; ++i){
  f.push_back(a[i]);
  f.push_back(a[n - i - 1]);
 }
 if (n % 2 != 0)
  f.push_back(a[n / 2]);
 int s = 0;
 for (int i = 0; i < n - 1; ++i)
  s += abs(f[i] - f[i + 1]);
 s += abs(f[n - 1] - f[0]);
 return s;
}

// Minimum sum of absolute difference of pairs of two arrays
long long int findMinSum(long long int a[], long long int b[], int n)
{
 sort(a, a + n);
 sort(b, b + n);
 long long int sum = 0;
 for (int i = 0; i < n; i++)
  sum += abs(a[i] - b[i]);
 return sum;
}

// Smallest subarray with sum greater than x
int smallestSubWithSum(int a[], int n, int x)
{
 int l = 0, r = 0, sum = 0, mn = INT_MAX;
 while (r < n)
 {
  sum += a[r++];
  while (l <= r && sum > x)
  {
   sum -= a[l++];
   mn = min(mn, r - l + 1);
  }
 }
 return mn;
}

// Chocolate Distribution Problem
long long findMinDiff(vector<long long> a, long long n, long long m)
{
 sort(a.begin(), a.end());
 long long ans = LONG_MAX;
 for (long long j = m - 1; j < n; j++)
```

```
  ans = min(ans, a[j] - a[j - m + 1]);
 return ans;
}

// Minimum Cost of ropes
long long minCost(long long arr[], long long n)
{
 if (n == 1)
  return 0;
 priority_queue<long long, vector<long long>, greater<long long>> pq;
 long long cost = 0;

 for (long long i = 0; i < n; i++)
  pq.push(arr[i]);

 while (pq.size() > 1)
 {
  long long pop1 = pq.top(); pq.pop();
  long long pop2 = pq.top(); pq.pop();
  long long sum = pop1 + pop2;
  pq.push(sum);
  cost += sum;
 }
 return cost;
}

// Find Maximum Equal sum of Three Stacks
int maxEqualSum(int N1, int N2, int N3, vector<int> &S1, vector<int> &S2, vector<int> &S3)
{
 int s1 = accumulate(S1.begin(), S1.end(), 0);
 int s2 = accumulate(S2.begin(), S2.end(), 0);
 int s3 = accumulate(S3.begin(), S3.end(), 0);
 int i = 0, j = 0, k = 0;
 while (i < N1 && j < N2 && k < N3)
 {
  if (s1 == s2 && s2 == s3)
   return s1;
  else if (s1 >= s2 && s1 >= s3)
   s1 -= S1[i++];
  else if (s2 >= s1 && s2 >= s3)
   s2 -= S2[j++];
  else if (s3 >= s1 && s3 >= s2)
   s3 -= S3[k++];
 }
 return 0;
}

// Rat in a Maze Problem
void solve(int i, int j, string move, vector<string> &res, vector<vector<int>> &m, vector<vector<int>> &visited, int
 n)
{
 if (i == n - 1 && j == n - 1) {
  res.push_back(move);
  return;
 }
```

```
 if (i < 0 || j < 0 || i > n - 1 || j > n - 1 || visited[i][j] == 1 || m[i][j] == 0) return;

 visited[i][j] = 1;
 solve(i + 1, j, move + "D", res, m, visited, n);
 solve(i, j - 1, move + "L", res, m, visited, n);
 solve(i, j + 1, move + "R", res, m, visited, n);
 solve(i - 1, j, move + "U", res, m, visited, n);
 visited[i][j] = 0;
}
vector<string> findPath(vector<vector<int>> &m, int n)
{
 vector<vector<int>> visited(n, vector<int>(n, 0));
 vector<string> res;
 if (m[0][0] == 0 || m[n - 1][n - 1] == 0)
 {
  return res;
 }
 solve(0, 0, "", res, m, visited, n);
 return res;
}

// N-Queens
vector<vector<string>> ret;
bool is_valid(vector<string> &board, int row, int col)
{
 for (int i = row; i >= 0; --i)
  if (board[i][col] == 'Q')
   return false;
 for (int i = row, j = col; i >= 0 && j >= 0; --i, --j)
  if (board[i][j] == 'Q')
   return false;
 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)
{
 if (row == board.size()){
  ret.push_back(board);
  return;
 }
 for (int i = 0; i < board.size(); ++i){
  if (is_valid(board, row, i)){
   board[row][i] = 'Q';
   dfs(board, row + 1);
   board[row][i] = '.';
  }
 }
}
vector<vector<string>> solveNQueens(int n)
{
 if (n <= 0) return {{}};
 vector<string> board(n, string(n, '.'));
 dfs(board, 0);
```

```
 return ret;
}
```

```
// Remove Invalid Parentheses
vector<string> ans;
unordered_set<string> uset;
int countRemoval(string s)
{
 stack<char> st;
 for (int i = 0; i < s.size(); i++)
 {
  if (s[i] == '(')
  {
   st.push('(');
  }
  else if (s[i] == ')')
  {
   if (st.size() == 0)
   {
    st.push(')');
   }
   else if (st.top() == ')')
   {
    st.push(')');
   }
   else if (st.top() == '(')
   {
    st.pop();
   }
  }
 }
 int invalid = st.size(); // minimum removals
 return invalid;
}
void helper(int invalid, string s)
{
 if (invalid < 0)
  return;
 if (invalid == 0)
 {
  int invalidNow = countRemoval(s);
  if (invalidNow == 0)
  {
   ans.push_back(s);
  }
  return;
 }
 for (int i = 0; i < s.size(); i++)
 {
  if (s[i] != ')' && s[i] != '(')
   continue;
```

```
  string left = s.substr(0, i);
  string right = s.substr(i + 1);
  string temp = left + right;
  if (uset.find(temp) == uset.end())
  {
   uset.insert(temp);
   helper(invalid - 1, temp);
  }
 }
}
vector<string> removeInvalidParentheses(string s)
{
 int invalid = countRemoval(s);
 helper(invalid, s);
 return ans;
}

// Sudoku Solver
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 (isValid(board, i, j, c))
     {
      board[i][j] = c;
      if (solve(board) == true)
       return true;
      else
       board[i][j] = '.';
     }
    }
    return false;
   }
  }
 }
 return true;
}
bool isValid(vector<vector<char>> &board, int row, int col,
    char c)
{
 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
const int N = 20;
int color[N];
bool check(int u, int n, int c, bool graph[101][101])
{
 for (int v = 0; v < n; v++)
 {
  if (u != v && graph[u][v] && color[v] == c)
   return true;
 }
 return false;
}
bool help(int u, int n, int m, bool graph[101][101])
{
 if (u == n) return true;
 for (int c = 0; c < m; c++){
  if (check(u, n, c, graph))
   continue;
  color[u] = c;
  if (help(u + 1, n, m, graph))
   return true;
  color[u] = -1;
 }
 return false;
}
bool graphColoring(bool graph[101][101], int m, int n)
{
 memset(color, -1, sizeof(color));
 return help(0, n, m, graph);
}

// Print all palindromic partitions of a string
vector<vector<string>> ans;
vector<string> subAns;
bool isPalindrome(string s)
{
 for (int i = 0; i < s.size() / 2; i++)
 {
  if (s[i] != s[s.size() - i - 1])
   return 0;
 }
 return 1;
}
void f(string s){
 if (!s.size()){
  ans.push_back(subAns);
  return;
 }
 for (int i = 0; i < s.size(); i++)
```

```
 {
  string l = s.substr(0, i + 1);
  string r = s.substr(i + 1);

  if (isPalindrome(l))
  {
   subAns.push_back(l);
   f(r);
   subAns.pop_back();
  }
 }
}
vector<vector<string>> allPalindromicPerms(string s){
 f(s);
 return ans;
}

// Knight Walk
bool isValid(int x, int y, int n)
{
 return (x >= 1 && x <= n && y >= 1 && y <= n);
}
int minStepToReachTarget(vector<int> &k, vector<int> &t, int N)
{
 vector<vector<int>> v(N + 1, vector<int>(N + 1, 0));
 queue<pair<int, int>> q;
 q.push({k[0], k[1]});
 v[k[0]][k[1]] = 1;

 int dx[8] = {-1, -2, -2, -1, 1, 2, 2, 1};
 int dy[8] = {2, 1, -1, -2, 2, 1, -1, -2};

 int moves = 0;
 bool found = false;
 if (k[0] == t[0] && k[1] == t[1])
 {
  return moves;
 }

 while (!q.empty())
 {
  int n = q.size();
  while (n--)
  {
   auto f = q.front(); q.pop();
   for (int i = 0; i < 8; i++){
    int x = f.first + dx[i];
    int y = f.second + dy[i];
    if (x == t[0] && y == t[1]){
     found = true;
     break;
    }
    if (isValid(x, y, N) && v[x][y] != 1){
     q.push({x, y});
     v[x][y] = 1;
```

```
    }
   }
  }
  moves++;
  if (found)
  {
   break;
  }
 }
 if (found == true)
 {
  return moves;
 }
 else
 {
  return -1;
 }
}

// Print Knight’s Tour
void printKnightsTour(vector<vector<int>> &chess, int r, int c, int move)
{
 if (r < 0 || c < 0 || r == chess.size() || c == chess.size())
  return;
 else if (move == chess.size() * chess.size()){
  chess[r][c] = move;
  displayBoard(chess);
  chess[r][c] = 0;
  return;
 }
 int xMove[8] = {2, 1, -1, -2, -2, -1, 1, 2};
 int yMove[8] = {1, 2, 2, 1, -1, -2, -2, -1};
 chess[r][c] = move;
 for (int i = 0; i < 8; i++){
  printKnightsTour(chess, r + xMove[i], c + yMove[i], move + 1);
 }
 chess[r][c] = 0;
}

// Combination Sum
// Combination Sum II
// copied subset free by second method
vector<vector<int>> ans;
void help(int i, vector<int> &C, int t, vector<int> &sol)
{
 if (t == 0)
 {
  ans.push_back(sol);
  return;
 }
 if (t < 0 || i == C.size())
 {
  return;
 }
 sol.push_back(C[i]);
```

```
 help(i, C, t - C[i], sol);
 sol.pop_back();

 help(i + 1, C, t, sol);
}
vector<vector<int>> combinationSum(vector<int> &C, int t)
{
 vector<int> sol;
 help(0, C, t, sol);
 return ans;
}

void findCombinations(int ind, int target, vector<int> &arr, vector<vector<int>> &ans, vector<int> &ds)
{
 if (target == 0)
 {
  ans.push_back(ds);
  return;
 }
 for (int i = ind; i < arr.size(); i++)
 {
  if (i > ind && arr[i] == arr[i - 1])
   continue;
  if (arr[i] > target)
   break;
  ds.push_back(arr[i]);
  findCombinations(i, target - arr[i], arr, ans, ds);
  ds.pop_back();
 }
}
vector<vector<int>> combinationSum(vector<int> &candidates, int target)
{
 sort(candidates.begin(), candidates.end());
 vector<vector<int>> ans;
 vector<int> ds;
 findCombinations(0, target, candidates, ans, ds);
 return ans;
}

// Reverse a string using Stack
char *reverse(char *S, int len)
{
 stack<int> st;
 for (int i = 0; i < len; i++) st.push(S[i]);

 int j = 0;
 while (!st.empty()) {
  char x = st.top();
  S[j] = x;
  j++;
  st.pop();
 }
 return S;
}
```

```
// Delete middle element of a stack
void deleteelement(stack<int> &s, int mid)
{
 if (mid == 0) {
  s.pop();
  return;
 }
 int temp = s.top();
 s.pop();
 deleteelement(s, mid - 1);
 s.push(temp);
}
void deleteMid(stack<int> &s, int sizeOfStack){
 int mid = (sizeOfStack / 2);
 deleteelement(s, mid);
}

// The Celebrity Problem
int celebrity(vector<vector<int>> &M, int n)
{
 int c = 0;
 for (int i = 1; i < n; i++)
  if (M[c][i] == 1)
   c = i;
 for (int i = 0; i < n; i++)
  if (i != c)
   if (M[c][i] == 1 or M[i][c] == 0)
    return -1;
 return c;
}

// Next Greater Element, Next Smaller Element
vector<long long> nextLargerElement(vector<long long> a, int n)
{
 stack<long long> s;
 for (int i = 0; i < n; i++)
 {
  while (s.size() && a[s.top()] < a[i])
  {
   a[s.top()] = a[i];
   s.pop();
  }
  s.push(i);
 }
 while (s.size())
 {
  a[s.top()] = -1;
  s.pop();
 }
 return a;
}

// Minimum element stack
stack<int> st;
void push(stack<int> &s, int a)
```

```
{

 if (st.empty() || st.top() >= a)
 {
  st.push(a);
 }
 s.push(a);
}

bool isFull(stack<int> &s, int n)
{
 // Your code goes here
 if (s.size() == n)
 {
  return true;
 }
 return false;
}

bool isEmpty(stack<int> &s)
{
 // Your code goes here
 if (s.empty())
 {
  return true;
 }
 return false;
}

int pop(stack<int> &s)
{
 // Your code goes here
 if (!s.empty())
 {
  int a = s.top();
  s.pop();
  if (a == st.top())
  {
   st.pop();
  }
  return a;
 }
 return -1;
}

int getMin(stack<int> &s)
{
 if (st.empty())
  return -1;
 return st.top();
}

// Reverse a stack using recursion
stack<int> st;
void insert_at_bottom(char x)
```

```
{
 if (st.size() == 0)
  st.push(x);
 else
 {
  char a = st.top();
  st.pop();
  insert_at_bottom(x);
  st.push(a);
 }
}
void reverse()
{
 if (st.size() > 0)
 {
  char x = st.top();
  st.pop();
  reverse();
  insert_at_bottom(x);
 }
}

// Reverse a Queue using recursion
void reverseQueue(queue<long long int> &q)
{
 if (q.empty())
  return;
 long long int data = q.front();
 q.pop();
 reverseQueue(q);
 q.push(data);
}

// Sort a Stack using recursion
void insertAtBottom(stack<int> &st, int x)
{
 if (!st.size() || st.top() <= x)
 {
  st.push(x);
  return;
 }
 int temp = st.top();
 st.pop();

 insertAtBottom(st, x);
 st.push(temp);
}
void sorting(stack<int> &st)
{
 if (st.empty())
  return;
 int x = st.top();
 st.pop();

 sorting(st);
```

```
 if (st.empty() || x >= st.top())
  st.push(x);
 else
  insertAtBottom(st, x);
}

// Merge Intervals
vector<vector<int>> merge(vector<vector<int>> &intervals)
{
 sort(intervals.begin(), intervals.end());
 vector<vector<int>> ans;
 for (auto interval : intervals)
 {
  if (ans.size() == 0 || ans.back()[1] < interval[0])
   ans.push_back(interval);
  else
   ans.back()[1] = max(ans.back()[1], interval[1]);
 }
 return ans;
}

// Length of the longest valid parentheses substring
int findMaxLen(string s)
{
 stack<int> stck;
 stck.push(-1);
 int res = 0;
 for (int i = 0; i < s.size(); i++)
 {
  if (s[i] == '(')
   stck.push(i);
  else {
   if (!stck.empty()){
    stck.pop();
    res = max(res, i - stck.top());
   }
   else
    stck.push(i);
  }
 }
 return res;
}

// Expression contains redundant bracket or not
bool checkRedundancy(string &str)
{
 stack<char> st;
 for (auto &ch : str)
 {
  if (ch == ')')
  {
   char top = st.top();
   st.pop();
   bool flag = true;
   while (!st.empty() and top != '(')
```

```
    {
    if (top == '+' || top == '-' ||
     top == '*' || top == '/')
     flag = false;
    top = st.top();
    st.pop();
   }
   if (flag == true)
    return true;
  }
  else
   st.push(ch);
 }
 return false;
}

// Implement Stack using Queue
// pop expensive
class MyStack
{
public:
 queue<int> q1;
 MyStack() {}
 void push(int x)
 {
  q1.push(x);
 }
 int pop()
 {
  int len = q1.size();
  for (int i = 0; i < len - 1; i++)
  {
   int tmp = q1.front();
   q1.pop();
   q1.push(tmp);
  }
  int res = q1.front();
  q1.pop();
  return res;
 }
 int top()
 {
  return q1.back();
 }
 bool empty()
 {
  return q1.empty();
 }
};
// push expensive

// Implement Queue using Stack
class MyQueue
{
public:
```

```
 stack<int> s1, s2;
 MyQueue()
 {
  // MyQueue myQueue = new MyQueue();
 }
 void push(int x)
 {
  while (s1.size())
  {
   s2.push(s1.top());
   s1.pop();
  }
  s1.push(x);
  while (s2.size())
  {
   s1.push(s2.top());
   s2.pop();
  }
 }
 int pop()
 {
  int temp = s1.top();
  s1.pop();
  return temp;
 }
 int peek()
 {
  return s1.top();
 }
 bool empty()
 {
  return s1.empty();
 }
};

// Count Derangements (Permutation such that no element appears in its original position)
int countDer(int n)
{
  if (n == 1) return 0;
  if (n == 2) return 1;
  return (n - 1) * (countDer(n - 1) + countDer(n - 2));
}

// Coin game winner where every player has three choices
bool findWinner(int x, int y, int n)
{
    int dp[n + 1];
    dp[0] = false; dp[1] = true;

    for (int i = 2; i <= n; i++) {
        if (i - 1 >= 0 and !dp[i - 1])
            dp[i] = true;
        else if (i - x >= 0 and !dp[i - x])
            dp[i] = true;
        else if (i - y >= 0 and !dp[i - y])
```

```
        dp[i] = true;
      else
        dp[i] = false;
    }
    return dp[n];
}
```

```
// Optimal Strategy For A Game
   long long func(int arr[],int s,int l,int n)
   {
      if(dp[s][l]!=-1) return dp[s][l];
      if(n==2) return dp[s][l]=max(arr[s],arr[l]);

      return dp[s][l]=max(min(func(arr,s+2,l,n-2),func(arr,s+1,l-1,n-2))+arr[s],
                    min(func(arr,s+1,l-1,n-2),func(arr,s,l-2,n-2))+arr[l]);
      }
    long long maximumAmount(int arr[], int n){
      memset(dp,-1,sizeof(dp));
      return  func(arr,0,n-1,n);
    }
```

```
// Stack Permutations
int isStackPermutation(int N, vector<int> &a, vector<int> &b)
{
 stack<int> temp;
 int j = 0;
 for (int i = 0; i < a.size(); i++){
  temp.push(a[i]);
  while (temp.size() && temp.top() == b[j]){
   temp.pop();
   j++;
  }
 }
 return j == b.size();
}
```

```
// Interleave the first half of the queue with second half
// Find the first circular tour that visits all Petrol Pumps- Gas Station
int canCompleteCircuit(vector<int> &gas, vector<int> &cost){
 int gas_tank = 0, start_index = 0, n = gas.size(), sum = 0;
 for (int i = 0; i < n; i++)
 {
  sum += gas[i] - cost[i];
  gas_tank += gas[i] - cost[i];
  if (gas_tank < 0)
  {
   start_index = i + 1;
   gas_tank = 0;
  }
 }
 return sum < 0 ? -1 : start_index;
}
```

```
// Reverse first k elements of a queue.
```

```
queue<int> modifyQueue(queue<int> q, int k)
{
 vector<int> v;
 queue<int> Q;
 while (k--){
  v.push_back(q.front());
  q.pop();
 }
 for (int i = v.size() - 1; i > -1; i--)
 {
  Q.push(v[i]);
 }
 while (!q.empty())
 {
  Q.push(q.front());
  q.pop();
 }
 return Q;
}

// Min Stack
vector<pair<int, int>> s;
void push(int val)
{
 if (s.empty())
  s.push_back({val, val});
 else
  s.push_back({val, min(s.back().second, val)});
}
void pop() { s.pop_back(); }
int top() { return s.back().first; }
int getMin() { return s.back().second; }

// First negative integer in every window of size k
vector<long long> printFirstNegativeInteger(long long int A[], long long int N, long long int K)
{
 vector<long long> ans;
 queue<long long> q;
 for (int i = 0; i < K - 1; i++)
  if (A[i] < 0)
   q.push(i);
 for (int i = K - 1; i < N; i++)
 {
  if (A[i] < 0)
   q.push(i);
  if (!q.empty())
   if (q.front() < i - K + 1)
    q.pop();
  if (!q.empty())
   ans.push_back(A[q.front()]);
  else
   ans.push_back(0);
 }
 return ans;
}
```

```
// Maximum of all subarrays of size k
// Sliding Window Maximum
vector<int> max_of_subarrays(int *arr, int n, int k)
{
 deque<int> d;
 vector<int> v;
 for (int i = 0; i < k; i++)
 {
  while (!d.empty() && arr[i] >= arr[d.back()])
   d.pop_back();
  d.push_back(i);
 }
 for (int i = k; i < n; i++)
 {
  v.push_back(arr[d.front()]);
  while (!d.empty() && d.front() <= i - k)
   d.pop_front();
  while (!d.empty() && arr[i] >= arr[d.back()])
   d.pop_back();
  d.push_back(i);
 }
 v.push_back(arr[d.front()]);
 return v;
}

//Sliding Window Median

// Maximum sum of a subarray of size K.
long maximumSumSubarray(int K, vector<int> &Arr, int N)
{
 long long int sum = 0, mx = INT_MIN;
 int i = 0, j = 0;
 while (j < N)
 {
  sum += Arr[j];
  if (j - i + 1 < K)
   j++;
  else if (j - i + 1 == K)
  {
   mx = max(mx, sum);
   sum -= Arr[i];
   i++;
   j++;
  }
 }
 return mx;
}

// Distance of nearest cell having 1
vector<vector<int>> nearest(vector<vector<int>> grid)
{
 int dx[4] = {-1, 1, 0, 0};
 int dy[4] = {0, 0, 1, -1};
 int n = grid.size(), m = grid[0].size();
```

```
 vector<vector<int>> vis(n, vector<int>(m, 0));
 queue<pair<int, int>> q;

 for (int i = 0; i < n; i++)
  for (int j = 0; j < m; j++)
   if (grid[i][j] == 1)
   {
    grid[i][j] = 0;
    q.push({i, j});
    vis[i][j] = 1;
   }

 while (!q.empty())
 {
  int s = q.size();
  while (s--)
  {
   auto pr = q.front();
   q.pop();
   int x = pr.first, y = pr.second;
   for (int i = 0; i < 4; i++)
   {
    if (x + dx[i] >= 0 && x + dx[i] < n && y + dy[i] >= 0 && y + dy[i] < m && vis[x + dx[i]][y + dy[i]] != 1)
    {
     grid[x + dx[i]][y + dy[i]] = 1 + grid[x][y];
     vis[x + dx[i]][y + dy[i]] = 1;
     q.push({x + dx[i], y + dy[i]});
    }
   }
  }
 }
 return grid;
}

// GRAPH
int graph()
{
 int n, m;
 cin >> n >> m;
 vector<int> adj[n + 1];
 for (int i = 0; i < m; i++)
 {
  int u, v;
  cin >> u >> v;
  adj[u].push_back(v);
  adj[v].push_back(u);
 }
 return 0;
}

// no of connected components//bfs
vector<int> bfsOfgraph(int V, vector<int> adj[])
{
 vector<int> bfs;
 vector<int> vis(V + 1, 0);
```

```
 int connectedComponenets = 0;
 for (int i = 1; i <= V; i++){
  if (!vis[i]){
   queue<int> q;
   q.push(i);
   vis[i] = 1;
   while (!q.empty()){
    int node = q.front();
    q.pop();
    bfs.push_back(node);
    for (auto it : adj[node]){
     if (!vis[it]){
      q.push(it);
      vis[it] = 1;
     }
    }
   }
  }
  connectedComponenets++;
 }
 return bfs;
}
// dfs
void dfs(int node, vector<int> &vis, vector<int> adj[], vector<int> &storeDfs){
 storeDfs.push_back(node);
 vis[node] = 1;
 for (auto it : adj[node])
  if (!vis[it])
      dfs(it, vis, adj, storeDfs);
}
vector<int> dfsOfGraph(int V, vector<int> adj[])
{
 vector<int> storeDfs;
 vector<int> vis(V + 1, 0);
 int connectedComponents = 0;
 for (int i = 1; i <= V; i++)
 {
  if (!vis[i])
  {
   dfs(i, vis, adj, storeDfs);
  }
 }
 return storeDfs;
}


// Detect a cycle in undirected graph using BFS
bool checkForCycle(int s, int V, vector<int> adj[],vector<int> &visited){
 queue<pair<int, int>> q;
 visited[s] = 1;
 q.push({s, -1});
 while (!q.empty()){
  int node = q.front().first;
  int par = q.front().second;
  q.pop();
```

```
  for (auto it : adj[node])
  {
   if (!visited[it])
   {
    visited[it] = 1;
    q.push({it, node});
   }
   else if (par != it)
   {
    return 1;
   }
  }
 }
 return 0;
}
bool isCycle(int V, vector<int> adj[])
{
 vector<int> vis(V + 1, 0);
 for (int i = 1; i <= V; i++)
  if (!vis[i])
   if (checkForCycle(i, V, adj, vis))
    return 1;
 return 0;
}

// Detect a cycle in undirected graph using DFS
bool checkForCycle(int node, int par, vector<int> &vis, vector<int> adj[])
{
 vis[node] = 1;
 for (auto it : adj[node])
 {
  if (!vis[it])
  {
   if (checkForCycle(it, node, vis, adj))
    return 1;
  }
  else if (it != par)
   return 1;
 }
 return 0;
}
bool isCycle(int V, vector<int> adj[])
{
 vector<int> vis(V + 1, 0);
 for (int i = 1; i <= V; i++)
 {
  if (!vis[i])
  {
   if (checkForCycle(i, -1, adj, vis))
    return 1;
  }
 }
 return 0;
}
```

```
// Detect a Cycle in Directed Graph - DFS
bool checkCycle(int node, vector<int> adj[], int vis[], int dfsVis[])
{
 vis[node] = 1;
 dfsVis[node] = 1;
 for (auto it : adj[node])
  {
  if (!vis[it])
   if(checkCycle(it, adj, vis, dfsVis))
    return 1;
  else if(dfsVis[it])
   return 1;
 }
 dfsVis[node] = 0;
 return 0;
}
bool isCyclic(int N, vector<int> adj[])
{
 int vis[N], dfsVis[N];
 memset(vis, 0, sizeof vis);
 memset(dfsVis, 0, sizeof dfsVis);
 for (int i = 0; i < N; i++)
  if (!vis[i])
   if (checkCycle(i, adj, vis, dfsVis))
    return 1;
 return 0;
}

// Detect a Cycle in Directed Graph - BFS
// Kahn's Algorithm
bool isCyclic(int N, vector<int> adj[])
{
 queue<int> q;
 vector<int> indegree(N, 0);
 for (int i = 0; i < N; i++)
  for (auto it : adj[i])
   indegree[it]++;
 for (int i = 0; i < N; i++)
  if (indegree[i] == 0)
   q.push(i);
 int cnt = 0;
 while (!q.empty()){
  int node = q.front();
  q.pop();
  cnt++;
  for (auto it : adj[node]){
   indegree[it]--;
   if (indegree[it] == 0)
    q.push(it);
  }
 }
 if (cnt == N) return 0; // no cycle
 return 1;   // cycle found
}
```

```
// Bipartite Graph - BFS
bool bipartiteBfs(int src, vector<int> adj[], int color[]){
 queue<int> q;
 q.push(src); color[src] = 1;
 while (!q.empty()){
  int node=q.front(); q.pop();
  for (auto it : adj[node]){
   if (color[it] == -1){
    color[it] = !color[node]; // 1->0, 0->1
    q.push(it);
   }
   else if (color[it] == color[node])
    return 0;
  }
 }
 return 1;
}
bool checkBipartite(vector<int> adj[], int n)
{
 int color[n];
 memset(color, -1, sizeof color);
 for (int i = 0; i < n; i++)
  if (color[i] == -1)
   if (!bipartiteBfs(i, adj, color))
    return 0;
 return 1;
}

// Bipartite Graph - DFS
bool bipartiteDfs(int node, vector<int> adj[], int color[])
{
 if (color[node] == -1)
  color[node] = 1;
 for (auto it : adj[node])
 {
  if (color[it] == -1)
  {
   color[it] = 1 - color[node];
   // color[it] = !color[node]; //1->0, 0->1
   if (!bipartiteDfs(it, adj, color))
    return 0;
  }
  else if (color[it] == color[node])
   return 0;
 }
 return 1;
}
bool checkBipartite(vector<int> adj[], int n)
{
 int color[n];
 memset(color, -1, sizeof color);
 for (int i = 0; i < n; i++)
 {
  if (color[i] == -1)
  {
```

```
   if (!bipartiteDfs(i, adj, color))
    return 0;
  }
 }
 return 1;
}

// Topological Sorting - DFS
// linear ordereing of vertices
void findTopoSort(int node, vector<int> &vis, stack<int> &st,
      vector<int> adj[])
{
 vis[node] = 1;
 for (auto it : adj[node])
  if(!vis[it])
   findTopoSort(it, vis, st, adj);
 st.push(node);
}
vector<int> topoSort(int N, vector<int> adj[])
{
 stack<int> st;
 vector<int> vis(N, 0);
 for (int i = 0; i < N; i++)
  if (!vis[i])
   findTopoSort(i, vis, st, adj);

 vector<int> topo;
 while (!st.empty()){
  topo.push_back(st.top());
  st.pop();
 }
 return topo;
}

// Topological Sorting - BFS
// Kahn's Algorithm
vector<int> topoSort(int N, vector<int> adj[])
{
 queue<int> q;
 vector<int> indegree(N, 0);
 for (int i = 0; i < N; i++)
  for (auto it : adj[i])
   indegree[it]++;
 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);
```

```
   }
  }
 }
 return topo;
}

// Shortest Path in Undirected Graph with Unit Weight
void BFS(vector<int> adj[], int N, int src){
 int dist[N]; memset(dist, INT_MAX, sizeof dist);
 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++)
  cout << dist[i] << " ";
}

// Shortest Path in Directed Acyclic Graph (DAG)
void shortestPath(int src, int N, vector<pair<int, int>> adj[]){
 vector<int> vis(N, 0);
 stack<int> st;
 for (int i = 0; i < N; i++)
  if (!vis[i])
   findTopoSort(i, vis, st, adj);
 int dist[N]; memset(dist, INT_MAX, sizeof dist);
 dist[src] = 0;
 while (!st.empty())
 {
  int node = st.top();
  st.pop();
  if (dist[node] != INT_MAX)
  {
   for (auto it : adj[node])
   {
    if (dist[node] + it.second < dist[it.first])
    {
     dist[it.first] = dist[node] + it.second;
    }
   }
  }
 }
 for (int i = 0; i < N; i++)
 {
  (dist[i] == INT_MAX) ? cout << "INF" : cout << dist;
  cout << " ";
 }
}
```

```
// Flood Fill
vector<vector<int>> floodFill(vector<vector<int>> &image, int sr, int sc, int newColor){
 if(newColor!=image[sr][sc]) fill(image, sr, sc, newColor, image[sr][sc]);
 return image;
}
void fill(vector<vector<int>> &image, int r, int c, int newColor, int oldColor){
 if (r<0 || r>=image.size() || c<0 || c>=image[0].size() || image[r][c]!=oldColor) return;
 image[r][c] = newColor;
 fill(image, r - 1, c, newColor, oldColor);
 fill(image, r + 1, c, newColor, oldColor);
 fill(image, r, c - 1, newColor, oldColor);
 fill(image, r, c + 1, newColor, oldColor);
}
vector<vector<int>> floodFill(vector<vector<int>> &image, int sr, int sc, int newColor)
{
 vector<vector<int>> visited(image.size(), vector<int>(image[0].size(), 0));

 queue<pair<int, int>> pq;
 pq.push({sr, sc});
 visited[sr][sc] = 1;

 while (pq.size() > 0)
 {
  int i = pq.front().first;
  int j = pq.front().second;

  pq.pop();

  if (i - 1 >= 0 && image[i - 1][j] == image[i][j] && visited[i - 1][j] == 0)
  {
   visited[i][j] = 1;
   pq.push({i - 1, j});
  }
  if (j - 1 >= 0 && image[i][j - 1] == image[i][j] && visited[i][j - 1] == 0)
  {
   visited[i][j - 1] = 1;
   pq.push({i, j - 1});
  }
  if (i + 1 < image.size() && image[i + 1][j] == image[i][j] && visited[i + 1][j] == 0)
  {
   visited[i + 1][j] = 1;
   pq.push({i + 1, j});
  }
  if (j + 1 < image[0].size() && image[i][j + 1] == image[i][j] && visited[i][j + 1] == 0)
  {
   visited[i][j + 1] = 1;
   pq.push({i, j + 1});
  }
  image[i][j] = newColor;
 }
 return image;
}

// Clone Graph
unordered_map<Node *, Node *> copies;
```

```
Node *cloneGraph(Node *node)
{
 if (!node) return NULL;
 Node *copy = new Node(node->val, {});
 copies[node] = copy;
 queue<Node *> todo;
 todo.push(node);
 while (!todo.empty()){
  Node *cur = todo.front(); todo.pop();
  for (Node *neighbor : cur->neighbors)
  {
   if (copies.find(neighbor) == copies.end())
   {
    copies[neighbor] = new Node(neighbor->val, {});
    todo.push(neighbor);
   }
   copies[cur]->neighbors.push_back(copies[neighbor]);
  }
 }
 return copy;
}

// Making wired Connections

// Word Ladder
int ladderLength(string beginWord, string endWord, vector<string> &wordList)
{
 if (find(wordList.begin(), wordList.end(), endWord) == wordList.end())
  return 0;
 set<string> s;
 for (auto i : wordList)
  s.insert(i);
 queue<string> q;
 q.push(beginWord);
 int d = 0;
 while (!q.empty())
 {
  d++;
  int n = q.size();
  while (n--)
  {
   string curr = q.front();
   q.pop();
   for (int i = 0; i < curr.length(); i++)
   {
    string tmp = curr;
    for (char c = 'a'; c <= 'z'; c++)
    {
     tmp[i] = c;
     if (tmp == curr)
      continue;
     if (tmp == endWord)
      return d + 1;
     if (s.find(tmp) != s.end())
     {
```

```
      q.push(tmp);
      s.erase(tmp);
     }
    }
   }
  }
 }
 return 0;
}

// Dijkstra Algorithm
vector<int> dijkstra(int V, vector<vector<int>> adj[], int S)
{
 priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
 vector<int> d(V, INT_MAX);

 d[S] = 0;
 pq.push({0, S});

 while (!pq.empty())
 {
  pair<int, int> pr = pq.top();
  pq.pop();
  for (auto j : adj[pr.second])
  {
   if (pr.first + j[1] < d[j[0]])
   {
    d[j[0]] = pr.first + j[1];
    pq.push({d[j[0]], j[0]});
   }
  }
 }
 return d;
}

// Minimum time taken by each job to be completed given by a Directed Acyclic Graph
vector<int> minimumTime(int n, vector<vector<int>> &edges, int m)
{
 vector<int> indegree(n + 1, 0);
 vector<int> adj[n + 1];
 for (auto i : edges){
  indegree[i[1]]++;
  adj[i[0]].push_back(i[1]);
 }
 vector<int> ans(n);
 queue<int> q;
 for(int i = 1; i <= n; i++)
  if(indegree[i] == 0)
   q.push(i);

 int count = 0;
 while (q.empty() == false){
  int x = q.size();
  count++;
  while (x){
```

```
   int node = q.front();
   ans[node - 1] = count;
   q.pop();
   for (int i : adj[node])
   {
    indegree[i]--;
    if (indegree[i] == 0)
     q.push(i);
   }
   x--;
  }
 }
 return ans;
}

// Prerequisite Tasks
bool dfs(int node, vector<int> adj[], vector<bool> &vis, vector<bool> &recs)
{
 vis[node] = true;
 recs[node] = true;
 for (auto i : adj[node])
 {
  if (!vis[i])
   if (dfs(i, adj, vis, recs))
    return true;
  if (recs[i])
   return true;
 }
 recs[node] = false;
 return false;
}
bool isPossible(int N, vector<pair<int, int>> &prerequisites)
{
 vector<int> adj[N];
 for (auto it : prerequisites)
  adj[it.first].push_back(it.second);
 vector<bool> vis(N, false);
 vector<bool> recs(N, false);
 for (int i = 0; i < N; i++)
 {
  if (!vis[i])
  {
   if (dfs(i, adj, vis, recs))
    return false;
  }
 }
 return true;
}

// Number of Islands
vector<pair<int, int>> direction{{1, 0}, {-1, 0}, {0, 1}, {0, -1}, {1, 1}, {-1, 1}, {1, -1}, {-1, -1}};
bool check(int x, int y, int n, int m)
{
 if (x >= 0 && y >= 0 && x < n && y < m)
  return true;
```

```
 else
  return false;
}
void dfs(vector<vector<char>> &grid, int x, int y)
{
 int n = grid.size();
 int m = grid[0].size();
 grid[x][y] = 0;
 for (auto i : direction)
 {
  int nx = x + i.first;
  int ny = y + i.second;
  if (check(nx, ny, n, m) && grid[nx][ny] == '1')
   dfs(grid, nx, ny);
 }
}
int numIslands(vector<vector<char>> &grid)
{
 int n = grid.size(), m = grid[0].size();
 int ans = 0;
 for (int i = 0; i < n; i++)
 {
  for (int j = 0; j < m; j++)
  {
   if (grid[i][j] == '1')
   {
    dfs(grid, i, j);
    ans++;
   }
  }
 }
 return ans;
}

// Alien Dictionary
string findOrder(string dict[], int N, int K)
{
 vector<int> g[K];
 vector<int> indegree(K, 0);
 for (int i = 0; i < N - 1; i++)
 {
  string a = dict[i], b = dict[i + 1];
  for (int j = 0; j < min(a.size(), b.size()); j++)
  {
   if (a[j] != b[j])
   {
    g[a[j] - 'a'].push_back(b[j] - 'a');
    indegree[b[j] - 'a']++;
    break;
   }
  }
 }
 queue<int> q;
 for (int i = 0; i < K; i++)
 {
```

```
 if (indegree[i] == 0)
  q.push(i);
 }
 vector<int> ans;
 while (q.size())
 {
  int f = q.front();
  q.pop();
  ans.push_back(f);
  for (auto v : g[f])
  {
   indegree[v]--;
   if (indegree[v] == 0)
    q.push(v);
  }
 }
 string s;
 for (auto v : ans)
  s += (v + 'a');
 return s;
}

// Bellman Ford ALgorithm
void BellmanFord(struct Graph *graph, int src)
{
 int V = graph->V;
 int E = graph->E;
 int dist[V];

 for (int i = 0; i < V; i++)
  dist[i] = INT_MAX;
 dist[src] = 0;

 for (int i = 1; i <= V - 1; i++)
 {
  for (int j = 0; j < E; j++)
  {
   int u = graph->edge[j].src;
   int v = graph->edge[j].dest;
   int weight = graph->edge[j].weight;
   if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
    dist[v] = dist[u] + weight;
  }
 }
 for (int i = 0; i < E; i++)
 {
  int u = graph->edge[i].src;
  int v = graph->edge[i].dest;
  int weight = graph->edge[i].weight;
  if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
  {
   printf("Graph contains negative weight cycle");
   return;
  }
 }
```

```
 printArr(dist, V);
 return;
}

// Implement Floyd warshall Algorithm
void floydWarshall(int graph[][V])
{
 int dist[V][V], i, j, k;
 for (i = 0; i < V; i++)
  for (j = 0; j < V; j++)
   dist[i][j] = graph[i][j];

 for (k = 0; k < V; k++)
 {
  for (i = 0; i < V; i++)
  {
   for (j = 0; j < V; j++)
   {
    if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF && dist[i][k] != INF))
     dist[i][j] = dist[i][k] + dist[k][j];
   }
  }
 }
 printSolution(dist);
}

// Snake and Ladders Problem
int snakesAndLadders(vector<vector<int>> &board)
{
 int n = board.size();
 vector<vector<bool>> visited(n, vector<bool>(n, false));

 queue<int> q; q.push(1);
 visited[n - 1][0] = true;
 int steps = 0;
 while (!q.empty()){
  int size = q.size();
  while (size--){
   int currpos = q.front();
   if (currpos == n * n)
    return steps;
   q.pop();
   for (int i = 1; i <= 6; i++)
   {
    int nextpos = currpos + i;
    if (nextpos > n * n) break;
    int r = n - (nextpos - 1) / n - 1;
    int c = (nextpos - 1) % n;
    if (r % 2 == n % 2)
     c = n - c - 1;

    if (!visited[r][c])
    {
     visited[r][c] = true;
     if (board[r][c] != -1)
```

```
     q.push(board[r][c]);
    else
     q.push(nextpos);
   }
  }
 }
 steps++;
}
return -1;
}
```

```
// Minimise the cashflow among a given set of friends who have borrowed money from each other
void minCashFlowRec(int amount[])
{
 int mxCredit = getMax(amount), mxDebit = getMin(amount);
 if (amount[mxCredit] == 0 && amount[mxDebit] == 0)
  return;
 int min = minOf2(-amount[mxDebit], amount[mxCredit]);
 amount[mxCredit] -= min;
 amount[mxDebit] += min;
 cout << "Person " << mxDebit << " pays " << min
   << " to "
   << "Person " << mxCredit << endl;
 minCashFlowRec(amount);
}
```

```
// Longest path in a Directed Acyclic Graph

// Cheapest Flights Within K Stops
int findCheapestPrice(int n, vector<vector<int>> &flights, int src, int dst, int k)
{
 vector<pair<int, int>> adj[n];
 int m = flights.size();
 for (int i = 0; i < m; i++){
  int p = flights[i][0];
  int q = flights[i][1];
  adj[p].push_back({q, flights[i][2]});
 }
 queue<pair<int, int>> q;
 vector<bool> vis(n, 0);
 q.push({0, src});
 vector<int> dist(n, INT_MAX);
 dist[src] = 0;
 k += 1;
 while (!q.empty())
 {
  int size = q.size();
  k--;
  if (k < 0) break;
  for (int i = 0; i < size; i++)
  {
   auto p = q.front();
   int node = p.second;
   int wt = p.first;
   q.pop();
```

```
   for (auto x : adj[node])
   {
    if (x.second + wt < dist[x.first])
    {
     dist[x.first] = x.second + wt;
     q.push({dist[x.first], x.first});
    }
   }
  }
 }
 return dist[dst] == INT_MAX ? -1 : dist[dst];
}

// Minimum edges to reverse or make path from source to destination

// Water Jug BFS
void BFS(int a, int b, int target)
{
 map<pii, int> m;
 bool isSolvable = false;
 vector<tuple<int, int, int>> path;
 map<pii, pii> mp;
 queue<pii> q;
 q.push(make_pair(0, 0));
 while (!q.empty())
 {
  auto u = q.front();
  // cout<<u.first<<" "<<u.second<<endl;
  q.pop();
  if (m[u] == 1)
   continue;
  if ((u.first > a || u.second > b || u.first < 0 || u.second < 0))
   continue;
  // cout<<u.first<<" "<<u.second<<endl;
  m[{u.first, u.second}] = 1;
  if (u.first == target || u.second == target)
  {
   isSolvable = true;
   printpath(mp, u);
   if (u.first == target)
   {
    if (u.second != 0)
     cout << u.first << " " << 0 << endl;
   }
   else
   {
    if (u.first != 0)
     cout << 0 << " " << u.second << endl;
   }
   return;
  }
  // completely fill the jug 2
  if (m[{u.first, b}] != 1)
  {
   q.push({u.first, b});
```

```
 mp[{u.first, b}] = u;
}
// completely fill the jug 1
if (m[{a, u.second}] != 1)
{
 q.push({a, u.second});
 mp[{a, u.second}] = u;
}
// transfer jug 1 -> jug 2
int d = b - u.second;
if (u.first >= d)
{
 int c = u.first - d;
 if (m[{c, b}] != 1)
  {
  q.push({c, b});
  mp[{c, b}] = u;
  }
}
else
{
 int c = u.first + u.second;
 if (m[{0, c}] != 1)
  {
  q.push({0, c});
  mp[{0, c}] = u;
  }
}
// transfer jug 2 -> jug 1
d = a - u.first;
if (u.second >= d)
{
 int c = u.second - d;
 if (m[{a, c}] != 1)
  {
  q.push({a, c});
  mp[{a, c}] = u;
  }
}
else
{
 int c = u.first + u.second;
 if (m[{c, 0}] != 1)
  {
  q.push({c, 0});
  mp[{c, 0}] = u;
  }
}
// empty the jug 2
if (m[{u.first, 0}] != 1)
{
 q.push({u.first, 0});
 mp[{u.first, 0}] = u;
}
// empty the jug 1
```

```
  if (m[{0, u.second}] != 1)
  {
   q.push({0, u.second});
   mp[{0, u.second}] = u;
  }
 }
 if (!isSolvable)
  cout << "No solution";
}

// Count set bits in an integer
int setBits(int N)
{
 int count = 0;
 while (N != 0)
 {
  count += (N % 2) & 1;
  N >>= 1;
 }
 return count;
}

// Non Repeating Numbers - 2 distinct
vector<int> singleNumber(vector<int> nums)
{
 int xorr = 0;
 for (auto &v : nums)
  xorr ^= v;
 vector<int> ans(2, 0);
 int rmb = xorr & (-xorr);
 for (auto &v : nums)
 {
  if (v & rmb)
   ans[0] ^= v;
  else
   ans[1] ^= v;
 }
 sort(ans.begin(), ans.end());
 return ans;
}

// Bit Difference
// Program to find whether a no is power of two
// Find position of the only set bit
// lmb = log2(n);
// rmb = n&(n);

// Repeat and Missing Number
void printTwoElements(int arr[], int size)
{
 int i;
 cout << " The repeating element is ";
 for (i = 0; i < size; i++)
 {
  if (arr[abs(arr[i]) - 1] > 0)
```

```
   arr[abs(arr[i]) - 1] = -arr[abs(arr[i]) - 1];
  else
   cout << abs(arr[i]) << "\n";
 }
 cout << "and the missing element is ";
 for (i = 0; i < size; i++)
 {
  if (arr[i] > 0)
   cout << (i + 1);
 }
}

// Set Matrix Zeros
void setZeroes(vector<vector<int>> &matrix)
{
 int m = matrix.size(), n = matrix[0].size();
 vector<int> row(m, 1), col(n, 1);
 for (int i = 0; i < m; i++)
  for (int j = 0; j < n; j++)
   if (matrix[i][j] == 0)
    row[i] = 0, col[j] = 0;
 for (int i = 0; i < m; i++)
  for (int j = 0; j < n; j++)
   if (row[i] == 0 || col[j] == 0)
    matrix[i][j] = 0;
}

// Pascal's Triangle
vector<vector<int>> generate(int numRows)
{
 vector<vector<int>> r(numRows);
 for (int i = 0; i < numRows; i++)
 {
  r[i].resize(i + 1);
  r[i][0] = r[i][i] = 1;

  for (int j = 1; j < i; j++)
   r[i][j] = r[i - 1][j - 1] + r[i - 1][j];
 }
 return r;
}

// Rotate Image
void rotate(vector<vector<int>> &matrix)
{
 int n = matrix.size();
 // transpose of given matrix
 for (int i = 0; i < n; i++)
  for (int j = 0; j < i; j++)
   swap(matrix[i][j], matrix[j][i]);
 // reverse of transpose matric
 for (int i = 0; i < n; i++)
  reverse(matrix[i].begin(), matrix[i].end());
}
```

```
// Pow(X,n)
double myPow(double x, int n)
{
 if (n == 0)
  return 1;
 double t = myPow(x, n / 2);
 if (n % 2)
  return n < 0 ? 1 / x * t * t : x * t * t;
 else
  return t * t;
}

// Majority Element (>N/3 times)
vector<int> majorityElement(vector<int> &a)
{
 int y(-1), z(-1), cy(0), cz(0);

 for (const auto &x : a)
 {
  if (x == y)
   cy++;
  else if (x == z)
   cz++;
  else if (!cy)
   y = x, cy = 1;
  else if (!cz)
   z = x, cz = 1;
  else
   cy--, cz--;
 }

 cy = cz = 0;
 for (const auto &x : a)
  if (x == y)
   cy++;
  else if (x == z)
   cz++;

 vector<int> r;
 if (cy > size(a) / 3)
  r.push_back(y);
 if (cz > size(a) / 3)
  r.push_back(z);
 return r;
}

// Unique Paths
int solve(int i, int j, int m, int n, vector<vector<int>> &dp)
{
 if (i >= m || j >= n)
  return 0;
 if (i == m - 1 && j == n - 1)
  return 1;
 if (dp[i][j] != -1)
  return dp[i][j];
```

```
 return dp[i][j] = solve(i + 1, j, m, n, dp) + solve(i, j + 1, m, n, dp);
}
int uniquePaths(int m, int n)
{
 vector<vector<int>> dp(m, vector<int>(n, -1));
 return solve(0, 0, m, n, dp);
}
int uniquePaths(int n, int m)
{
 vector<vector<int>> dp(m, vector<int>(n, 1));
 for (int i = 1; i < m; i++)
 {
  for (int j = 1; j < n; j++)
  {
   dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
  }
 }
 return dp[m - 1][n - 1];
}

// Unique Paths-2
int solveMemo(int i, int j, vector<vector<int>> &grid, vector<vector<int>> &dp)
{
 if (i < 0 || j < 0 || grid[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] = solveMemo(i - 1, j, grid, dp) + solveMemo(i, j - 1, grid, dp);
}
int solveTabu(int m, int n, vector<vector<int>> &grid)
{
 vector<vector<int>> dp(m, vector<int>(n, 0));
 for (int i = 0; i < m; i++)
 {
  for (int j = 0; j < n; j++)
  {
   if (grid[i][j] == 1)
    continue;
   if (i == 0 && j == 0)
    dp[i][j] = 1;
   else
   {
    int up = 0, 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];
```

```
}


// Longest Substring Without Repeating Characters
int lengthOfLongestSubstring(string s)
{
 vector<int> mpp(256, -1);
 int left = 0, right = 0;
 int n = s.size();
 int len = 0;
 while (right < n)
 {
  if (mpp[s[right]] != -1)
   left = max(mpp[s[right]] + 1, left);

  mpp[s[right]] = right;

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

// Count number of subarrays with given XOR
int subarraysXor(vector<int> &A, int B)
{
 map<int, int> m;
 int xr = 0, c = 0;
 m[0]++;
 for (int i = 0; i < A.size(); i++)
 {
  xr ^= A[i];
  c += m[xr ^ B];
  m[xr]++;
 }
 return c;
}

// Remove Duplicate from Sorted array
int removeDuplicates(vector<int> &nums)
{
 if (!nums.size())
  return 0;
 int i = 0;
 for (int j = 1; j < nums.size(); j++)
  if (nums[j] != nums[i])
   nums[i++] = nums[j];
 return i + 1;
}

// Max consecutive one
int findMaxConsecutiveOnes(vector<int> &nums)
{
 int n = nums.size();
 int count = 0, maxsm = 0;
```

```
 for (int i = 0; i < n; i++)
 {
  if (nums[i] == 1)
   count++;
  else
   count = 0;
  maxsm = max(maxsm, count);
 }
 return maxsm;
}

// Job Sequencing Problem
bool comp(Job a, Job b){
 return a.profit > b.profit;
}
vector<int> JobScheduling(Job arr[], int n)
{
 sort(arr, arr + n, comp);
 bool done[n] = {0}; // represent days
 int day = 0, profit = 0;
 for (int i = 0; i < n; i++)
 {
  for (int j = min(n, arr[i].dead) - 1; j >= 0; j--)
  {
   if (done[j] == false)
   {
    day += 1;
    profit += arr[i].profit;
    done[j] = true;
    break;
   }
  }
 }
 return {day, profit};
}

// Subset Sum Problem
bool isSubsetSum(vector<int> arr, int sum)
{
 bool dp[arr.size() + 1][sum + 1];
 for (int i = 0; i <= sum; i++)
  dp[0][i] = false;
 for (int i = 0; i <= arr.size(); i++)
  dp[i][0] = true;

 for (int i = 1; i <= arr.size(); i++)
 {
  for (int j = 1; j <= sum; j++)
  {
   if (arr[i - 1] <= j)
   {
    dp[i][j] = dp[i - 1][j - arr[i - 1]] || dp[i - 1][j];
   }
   else
   {
```

```
    dp[i][j] = dp[i - 1][j];
   }
  }
 }
 return dp[arr.size()][sum];
}
int knapsack(vector<int> &wt, int W, int n, vector<vector<int>> &dp)
{
 if (W == 0)
  return 1;
 if (n <= 0)
  return 0;
 if (dp[n][W] != -1)
  return dp[n][W];
 if (W - wt[n - 1] >= 0)
  return dp[n][W] = knapsack(wt, W - wt[n - 1], n - 1, dp) + knapsack(wt, W, n - 1, dp);
 else
  return dp[n][W] = knapsack(wt, W, n - 1, dp);
}
bool isSubsetSum(vector<int> arr, int sum)
{
 int n = arr.size();
 vector<vector<int>> dp(n + 1, vector<int>(sum + 1, -1));
 return knapsack(arr, sum, n, dp);
}

// Equal sum subset
bool canPartition(vector<int> &nums, int i = 0, int sum1 = 0, int sum2 = 0)
{
 if (i >= size(nums))
  return sum1 == sum2;            // check if both subset have equal sum
 return canPartition(nums, i + 1, sum1 + nums[i], sum2)    // try including into subset-1
     || canPartition(nums, i + 1, sum1, sum2 + nums[i]); // try including into subset-2
}
bool canPartition(vector<int> &nums)
{
 int sum = 0;
 for (auto a : nums)
  sum += a;
 if (sum % 2)
  return false;
 sum /= 2;
 vector<bool> dp(sum + 1, false);
 dp[0] = true;
 for (auto a : nums)
 {
  for (int i = sum; i >= a; i--)
  {
   dp[i] = dp[i] || dp[i - a];
  }
 }
 return dp[sum];
}
bool canPartition(vector<int> &nums)
{
```

```
 int totalSum = accumulate(begin(nums), end(nums), 0);
 if (totalSum & 1)
  return false;
 return subsetSum(nums, totalSum / 2);
}
bool subsetSum(vector<int> &nums, int sum, int i = 0)
{
 if (sum == 0)
  return true;
 if (i >= size(nums) || sum < 0)
  return false;
 if (dp[i][sum] != -1)
  return dp[i][sum];
 return dp[i][sum] = subsetSum(nums, sum - nums[i], i + 1) || subsetSum(nums, sum, i + 1);
}
bool subsetSum(vector<int> &nums, int sum, int i = 0)
{
 if (sum == 0)
  return true;
 if (i >= size(nums) || sum < 0)
  return false;
 if (dp[sum] != -1)
  return dp[sum];
 return dp[sum] = subsetSum(nums, sum - nums[i], i + 1) || subsetSum(nums, sum, i + 1);
}

// K Partition sum
bool canPartitionKSubsets(vector<int> &nums, int k)
{
 int sum = 0;
 sum = accumulate(nums.begin(), nums.end(), sum);
 if (nums.size() < k || sum % k)
  return false;

 vector<int> visited(nums.size(), false);
 return backtrack(nums, visited, sum / k, 0, 0, k);
}

bool backtrack(vector<int> &nums, vector<int> &visited, int target, int curr_sum, int i, int k)
{
 if (k == 1) return true;
 if (curr_sum == target) return backtrack(nums, visited, target, 0, 0, k - 1);
 for (int j = i; j < nums.size(); j++){
  if (visited[j] || curr_sum + nums[j] > target)
   continue;

  visited[j] = true;
  if (backtrack(nums, visited, target, curr_sum + nums[j], j + 1, k))
   return true;
  visited[j] = false;
 }
 return false;
}

// K-th permutation Sequence
```

```
string getPermutation(int N, int K)
{
 int n = N - 1, k = K - 1, nt, kt;
 findfact(N);
 vector<int> num(N);
 for (int i = 0; i < N; i++)
  num[i] = i + 1;
 vector<int>::iterator it;
 string ans = "";
 while (n >= 0){
  nt = k / fact[n];
  kt = k % fact[n];

  ans += (num[nt] + '0');
  it = num.begin();
  num.erase(it + nt);
  n--;
  k = kt;
 }
 return ans;
}

// N-th root of an integer
void getNthRoot(int n, int m)
{
 double low = 1;
 double high = m;
 double eps = 1e-6;

 while ((high - low) > eps)
 {
  double mid = (low + high) / 2.0;
  if (multiply(mid, n) < m)
  {
   low = mid;
  }
  else
  {
   high = mid;
  }
 }
 cout << n << "th root of " << m << " is " << low << endl;
}

// Find square of a number without using multiplication or division operators.
int square(int n) // 1
{
 if (n < 0)
  n = -n;
 int res = n;
 for (int i = 1; i < n; i++)
  res += n;
 return res;
}
int square(int n) // 2
```

```
{
 if (n == 0)
  return 0;
 if (n < 0)
  n = -n;
 if (n & 1)
  return 4 * square(n / 2) + 4 * x + 1;
 else
  return 4 * square(n / 2);
}

// Divide Integers without / operator
int divide(int dividend, int divisor)
{
 int sign = ((dividend < 0) ^ (divisor < 0)) ? -1 : 1;
 dividend = abs(dividend);
 divisor = abs(divisor);
 int quotient = 0;
 while (dividend >= divisor)
 {
  dividend -= divisor;
  ++quotient;
 }
 // long long quotient = 0, temp = 0;
 // for (int i = 31; i >= 0; --i) {
 //  if (temp + (divisor << i) <= dividend) {
 //   temp += divisor << i;
 //   quotient |= 1LL << i;
 //  }
 // }
 return quotient * sign;
}
int divide(int dividend, int divisor)
{
    int lo = 0,  hi = abs(dividend);
    if (dividend == INT_MIN)
    {
        if (divisor == -1)
        {
            return INT_MAX;
        }
        else if (divisor == 1)
        {
            return INT_MIN;
        }
    }

    int quotient = 0;
    while (lo <= hi)
    {
        int mid = (lo + hi) / 2;
        if (abs(divisor) * mid <= abs(dividend))
        {
            quotient = mid;
            lo = mid + 1;
```

```
        }
        else
        {
            hi = mid - 1;
        }
    }
    return ((dividend < 0 && divisor < 0) || (dividend >= 0 && divisor >= 0) ? quotient : -quotient);
}

// LRU cache
class LRUCache
{
    public:
        list<pair<int,int>> l;
        unordered_map<int,list<pair<int, int>>::iterator> m;
        int size;
        LRUCache(int capacity)
        {
            size=capacity;
        }
        int get(int key)
        {
            if(m.find(key)==m.end())
                return -1;
            l.splice(l.begin(),l,m[key]);
            return m[key]->second;
        }
        void put(int key, int value)
        {
            if(m.find(key)!=m.end())
            {
                l.splice(l.begin(),l,m[key]);
                m[key]->second=value;
                return;
            }
            if(l.size()==size)
            {
                auto d_key=l.back().first;
                l.pop_back();
                m.erase(d_key);
            }
            l.push_front({key,value});
            m[key]=l.begin();
        }
};
// LFU Cache
class LFUCache {
public:
    struct Node {
        int key; // key of the element.
        int val; // value of the ement.
        int fre; // usage frequency
        int timeStamp; // the latest time stamp when this element is accessed.
        Node(): key(-1), val(-1), timeStamp(-1), fre(0) {}
        Node(int k, int v, int ts): key(k), val(v), timeStamp(ts), fre(1) {}
```

```
    };

    LFUCache(int capacity) {
        Cap = capacity;
        Node* dummy = new Node();
        pq.push_back(dummy); // The pq start from pq[1].
        ts = 0;
    }

    int get(int key) {
        if(!mp.count(key)) return -1;
        int index = mp[key];
        int val = pq[index]->val;
 pq[index]->fre++;
 pq[index]->timeStamp = ++ts;
        sink(index);
        return val;
    }

    void set(int key, int value) {
        if(Cap <= 0) return;
 if(mp.count(key)) {
    int index = mp[key];
    pq[index]->val = value;
    get(key);
 }
 else {
     if(pq.size() - 1 == Cap) {
         int oldKey = pq[1]->key;
  mp.erase(oldKey);
  Node* newnode = new Node(key, value, ++ts);
  pq[1] = newnode;
  mp[key] = 1;
  sink(1);
     }
     else {
         Node* newnode = new Node(key, value, ++ts);
  pq.push_back(newnode);
  mp[key] = pq.size() - 1;
  swim(pq.size() - 1);
     }
 }
    }

private:
 vector<Node*> pq; // A priority queue, with the least usage frequency and least recently used element at the top.
 unordered_map<int, int> mp; // A mapping from the key of the element to its index in the priority queue.
 int Cap; // Capcity of the cache
 int ts; // time-stamp: indicate the time stamp of the latest operation of an element. According to the requirement of L
FU cache, when we need to evict an element from the cache, but there are multiple elements with the same minimum
 frequency, then the least recently used element should be evicted.

    /*
     * Recursively sink a node in priority queue. A node will be sinked, when its frequency is larger than any of its
     * children nodes, or the node has the same frequency with a child, but it is recently updated.
```

```
    */
 void sink(int index) {
    int left = 2 * index, right = 2 * index + 1, target = index;
    if(left < pq.size() && pq[left]->fre <= pq[target]->fre) // If the left child has the same frequency, we probably nee
d to swap the parent node and the child node, because the parent node is recently accessed, and the left child node w
as accessed at an older time stamp.
            target = left;
          if(right < pq.size()) {
              if(pq[right]->fre < pq[target]->fre || (pq[right]->fre == pq[target]->fre && pq[right]->timeStamp < pq[targ
et]->timeStamp)) // If right child has the same frequency and an older time stamp, we must swap it.
                  target = right;
  }
  if(target != index) {
     myswap(target, index);
           sink(target);
  }
 }

   /*a
    * Recursively swim a node in priority queue. A node will be swimmed, when its frequency is less than its
    * parent node. If the node has the same frequency with its parent, it is not needed to be swimmed, because
    * it is recently accessed.
    */
 void swim(int index) {
    int par = index / 2;
    while(par > 0 && pq[par]->fre > pq[index]->fre) {
        myswap(par, index);
  index = par;
  par /= 2;
    }
 }

 void myswap(int id1, int id2) {
    swap(pq[id1], pq[id2]);
    mp[pq[id1]->key] = id1;
    mp[pq[id2]->key] = id2;
 }
};

// Travelling Salesman TSP
void tsp(vector<vector<int>>&cost,int s,int mask,int fare,int &ans)
{
 int n = cost.size();
 if(mask==((1<<n)-1))
 {
  ans = min(ans,fare+cost[s][0]);
  return;
 }

 for(int i=0;i<n;i++)
  if(!(mask&(1<<i)))
   tsp(cost,i,mask|(1<<i),fare+cost[s][i],ans);

}
int total_cost(vector<vector<int>>cost){
```

```
 int mask = 1,ans=INT_MAX , n = cost.size();
 if(n<=1) return 0;
    tsp(cost,0,mask,0,ans);
    return ans;
}

// Construct Binary Tree from Preorder and Inorder Traversal
int m_curr = 0; vector<int> m_preorder, m_inorder;
TreeNode* rec(int l, int r) {
 if (l > r) return NULL;
 int i = 0;
 while (m_inorder[i] != m_preorder[m_curr]) i++;

 m_curr++;
 TreeNode* node = new TreeNode(m_inorder[i]);
 node->left = rec(l, i-1);
 node->right = rec(i+1, r);
 return node;

}
TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
 m_preorder = preorder, m_inorder = inorder;
 return rec(0, preorder.size()-1);
}
// Construct Binary Tree from Inorder and Postorder Traversal
int m_curr; vector<int> m_postorder, m_inorder;
TreeNode* rec(int l, int r) {
 if (l > r) return NULL;

 int i = 0;
 while (m_inorder[i] != m_postorder[m_curr]) i++;

 m_curr--;
 TreeNode* node = new TreeNode(m_inorder[i]);
 node->right = rec(i+1, r);
 node->left = rec(l, i-1);
 return node;
}
TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
 m_postorder = postorder, m_inorder = inorder, m_curr = inorder.size()-1;
 return rec(0, postorder.size()-1);
}

// Construct Binary Tree from Preorder and Postorder Traversal
int preIndex = 0, posIndex = 0;
TreeNode* constructFromPrePost(vector<int>& pre, vector<int>& post) {
 TreeNode* root = new TreeNode(pre[preIndex++]);
 if (root->val != post[posIndex])
  root->left = constructFromPrePost(pre, post);
 if (root->val != post[posIndex])
  root->right = constructFromPrePost(pre, post);
 posIndex++;
 return root;
}
```

```
// Rotting Oranges
int orangesRotting(vector<vector<int>> &grid)
{
 vector<int> dir = {-1, 0, 1, 0, -1};
 int m = grid.size(), n = grid[0].size();
 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++;

 int ans = -1;
 while (!q.empty())
 {
  int sz = q.size();
  while (sz--)
  {
   pair<int, int> p = q.front();
   q.pop();
   for (int i = 0; i < 4; i++)
   {
    int r = p.first + dir[i];
    int c = p.second + dir[i + 1];
    if (r >= 0 && r < m && c >= 0 && c < n && grid[r][c] == 1)
    {
     grid[r][c] = 2;
     q.push({r, c});
     fresh--;
    }
   }
  }
  ans++;
 }

 if (fresh > 0)
  return -1;
 if (ans == -1)
  return 0;
 return ans;
}

// Stock span problem
vector<int> calculateSpan(int price[], int n)
{
 vector<int> res;
 stack<int> st;
 for (int i = 0; i < n; i++)
 {
```

```
  int curr = price[i];
  while (st.empty() == false && price[st.top()] <= curr)
  {
   st.pop();
  }
  if (st.empty())
   res.push_back(i + 1);
  else
   res.push_back(i - st.top());
  st.push(i);
 }
 return res;
}

// Reverse Words in a String
string reverseWords(string s)
{
 vector<string> vec;
 stringstream str(s);
 string word;
 while (str >> word)
  vec.push_back(word);

 reverse(vec.begin(), vec.end());
 string res;
 for (const auto &it : vec)
  res += " " + it;
 res.erase(0, 1);
 return res;
}

// Binary Tree Maximum Path Sum
int mx = INT_MIN;
int f(TreeNode *root)
{
 if (!root)
  return 0;
 int l = max(f(root->left), 0);
 int r = max(f(root->right), 0);
 mx = max(mx, root->val + l + r);
 return root->val + max(l, r);
}
int maxPathSum(TreeNode *root)
{
 f(root);
 return mx;
}



// Common elements in all rows of a given matrix
vector<int> findCommonElements(vector<vector<int>> &m)
{

 map<int, int> mp;
```

```
 for (int i = 0; i < m.size(); i++)
 {
  map<int, int> mp1;
  for (int j = 0; j < m[0].size(); j++)
  {
   mp1[m[i][j]]++;
  }
  for (auto v : mp1)
   mp[v.first]++;
 }
 vector<int> ans;
 for (auto v : mp)
  if (v.second == m.size())
   ans.push_back(v.first);
 return ans;
}

// Policy Based Data Structure
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
{
 ordered_set o;
 o.insert(1);
 o.insert(2);
 o.insert(3);
 // find_by_order= nth element
 cout << *(o.find_by_order(1)) << endl;
 // order_of_key less than x
 cout << o.order_of_key(4) << endl;
}

// Replace Elements with Greatest Element on Right Side
vector replaceElements(vector &arr)
{
 int n = arr.size(), temp, mx = -1;
 for (int i = n - 1; i >= 0; i--)
 {
  temp = arr[i];
  arr[i] = mx;
  mx = max(mx, temp);
 }
 return arr;
}

// Median of BST
void Inorder(Node *root, vector<int> &ans)
{
 if (!root) return;
 Inorder(root->left, ans);
 ans.push_back(root->data);
 Inorder(root->right, ans);
}
```

```
float findMedian(struct Node *root)
{
 vector<int> v;
 Inorder(root, v);
 if (v.size() % 2 == 0)
 { // even case
  return (double)(v[v.size() / 2] + v[v.size() / 2 - 1]) / 2;
 }
 return v[v.size() / 2];
}

// Largest BST
class info{
 public:
  int maxi;
  int mini;
  bool isbst;
  int size;
};
info solve(Node *root, int &ans)
{
 if (root == NULL)
  return {INT_MIN, INT_MAX, true, 0};

 info left = solve(root->left, ans);
 info right = solve(root->right, ans);

 info curnode;
 curnode.maxi = max(root->data, right.maxi);
 curnode.mini = min(root->data, left.mini);
 if (left.isbst && right.isbst && (root->data > left.maxi && root->data < right.mini))
  curnode.isbst = true;
 else
  curnode.isbst = false;
 curnode.size = left.size + right.size + 1;

 if (curnode.isbst)
  ans = max(ans, curnode.size);
 return curnode;
}
int largestBst(Node *root)
{
 int ans = 0;
 solve(root, ans);
 return ans;
}

// Pre Order Traverse
List<Integer> preorderTraversal(TreeNode root)
{
 List<Integer> result = new ArrayList<>();
 Deque<TreeNode> stack = new ArrayDeque<>();
 TreeNode p = root;
 while (!stack.isEmpty() || p != null)
 {
```

```
  if (p != null)
  {
   stack.push(p);
   result.add(p.val); // Add before going to children
   p = p.left;
  }
  else
  {
   TreeNode node = stack.pop();
   p = node.right;
  }
 }
 return result;
}
// In Order Traverse
List<Integer> inorderTraversal(TreeNode root)
{
 List<Integer> result = new ArrayList<>();
 Deque<TreeNode> stack = new ArrayDeque<>();
 TreeNode p = root;
 while (!stack.isEmpty() || p != null)
 {
  if (p != null)
  {
   stack.push(p);
   p = p.left;
  }
  else
  {
   TreeNode node = stack.pop();
   result.add(node.val); // Add after all left children
   p = node.right;
  }
 }
 return result;
}
// Post Order Traverse
List<Integer> postorderTraversal(TreeNode root)
{
 LinkedList<Integer> result = new LinkedList<>();
 Deque<TreeNode> stack = new ArrayDeque<>();
 TreeNode p = root;
 while (!stack.isEmpty() || p != null)
 {
  if (p != null)
  {
   stack.push(p);
   result.addFirst(p.val); // Reverse the process of preorder
   p = p.right;   // Reverse the process of preorder
  }
  else
  {
   TreeNode node = stack.pop();
   p = node.left; // Reverse the process of preorder
  }
```

```
 }
 return result;
}
```

```
// Construct BST from inorder traversal

// Unique rows in boolean matrix
vector<vector<int>> uniqueRow(int M[MAX][MAX], int row, int col)
{
 vector<vector<int>> ans;
 unordered_map<int, int> u1;
 set<int> u2;
 int sum = 0;
 for (int i = 0; i < row; i++)
 {
  sum = 0;
  for (int j = 0; j < col; j++)
  {
   sum = sum + M[i][j] * pow(2, j);
  }
  auto it = u1.find(sum);
  if (it != u1.end())
  {
   continue;
  }
  else
  {
   u1.insert({sum, i});
  }
 }
 for (auto it = u1.begin(); it != u1.end(); it++)
 {
  u2.insert(it->second);
 }
 for (auto it = u2.begin(); it != u2.end(); it++)
 {
  vector<int> row1;
  for (int j = 0; j < col; j++)
  {
   row1.push_back(M[(*it)][j]);
  }
  ans.push_back(row1);
 }
 return ans;
}

// K-th Largest Sum Contiguous Subarray
int kthLargestSum(int arr[], int n, int k)
{
 int sum[n + 1];
 sum[0] = 0;
 sum[1] = arr[0];
 for (int i = 2; i <= n; i++)
  sum[i] = sum[i - 1] + arr[i - 1];
 priority_queue<int, vector<int>, greater<int>> Q;
```

```
 for (int i = 1; i <= n; i++)
 {
  for (int j = i; j <= n; j++)
  {
   int x = sum[j] - sum[i - 1];
   if (Q.size() < k)
    Q.push(x);
   else
   {
    if (Q.top() < x)
    {
     Q.pop();
     Q.push(x);
    }
   }
  }
 }
 return Q.top();
}
```

// Minimum sum of squares of character counts in a given string after removing “k” characters.

```
// Queue based approach or first non-repeating character in a stream.
string FirstNonRepeating(string A)
{
 vector<int> vis(26, 0);
 string ans = "";
 vector<char> v;
 int n = A.size();

 for (int i = 0; i < n; i++)
 {
  if (!vis[A[i] - 'a'])
   v.push_back(A[i]);
  vis[A[i] - 'a']++;
  int f = 0, m = v.size();

  for (int j = 0; j < m; j++)
  {
   if (vis[v[j] - 'a'] == 1)
   {
    ans.push_back(v[j]);
    f = 1;
    break;
   }
  }
  if (!f)
  {
   ans.push_back('#');
  }
 }
 return ans;
}
```

```
// Find median in a stream of running integers.
priority_queue<int> q1;
priority_queue<int, vector<int>, greater<int>> q2;
void balance()
{
 while (q1.size() < q2.size())
 {
  q1.push(q2.top());
  q2.pop();
 }
 if (q1.size() - q2.size() > 1)
 {
  q2.push(q1.top());
  q1.pop();
 }
}
MedianFinder()
{
}
void addNum(int num)
{
 if (q1.empty() || num < q1.top())
  q1.push(num);
 else
  q2.push(num);
 balance();
}
double findMedian()
{
 if (q1.size() > q2.size())
  return q1.top();
 else
  return ((float)q1.top() + q2.top()) / 2;
}

// Subarrays with K Different Integers
int subarraysWithKDistinct(vector<int> &A, int K)
{
 return atMostK(A, K) - atMostK(A, K - 1);
}
int atMostK(vector<int> &A, int K)
{
 int i = 0, res = 0;
 unordered_map<int, int> count;
 for (int j = 0; j < A.size(); ++j)
 {
  if (!count[A[j]]++)
   K--;
  while (K < 0)
  {
   if (!--count[A[i]])
    K++;
   i++;
  }
  res += j - i + 1;
```

```
 }
 return res;
}

// Smallest range in K lists
vector<int> smallestRange(vector<vector<int>> &nums)
{
 vector<int> v = {-100000, 100000};
 priority_queue<vector<int>, vector<vector<int>>, greater<vector<int>>> pq;
 int mx = INT_MIN, n = nums.size(), d = INT_MAX;
 for (int i = 0; i < n; i++)
 {
  if (nums[i][0] > mx)
   mx = nums[i][0];
  pq.push({nums[i][0], i, 0});
 }
 while (!pq.empty())
 {
  vector<int> mn = pq.top();
  pq.pop();
  int td = mx - mn[0];
  if (td < d)
  {
   d = td;
   v = {mn[0], mx};
  }
  if (mn[2] + 1 < nums[mn[1]].size())
  {
   if (nums[mn[1]][mn[2] + 1] > mx)
    mx = nums[mn[1]][mn[2] + 1];
   pq.push({nums[mn[1]][mn[2] + 1], mn[1], mn[2] + 1});
  }
  else
   break;
 }
 return v;
}

// Minimum sum of two numbers formed from digits of an array
string largeStringAdd(string &a, string &b){
   int n = a.size();
   int m = b.size();
   int nm = max(n, m);
   string ans(nm, '0');

   int i = n-1;
   int j = m-1;
   int k = nm-1;
   int c = 0;

   while(i >= 0 || j >= 0){
       int n1 = i < 0 ? 0 : a[i--]-'0';
       int n2 = j < 0 ? 0 : b[j--]-'0';
       int sum = n1 + n2 + c;
       c = sum / 10;
```

```
        sum = sum % 10;
        ans[k--] = sum+'0';
    }
    if(c == 1)
        ans = "1"+ans;
    return ans;
}
string solve(int arr[], int n) {
    sort(arr, arr+n);
    string a,b;
    for(int i=0; i<n; i++){
        if(arr[i] == 0)  continue;
        if(i % 2)  a.push_back(arr[i]+'0');
        else  b.push_back(arr[i]+'0');
    }
    return largeStringAdd(a, b);
}



// Optimal binary search tree- MCM Variation
int dp[101][101];
int sum(int i, int j, int frq[])
{
 int s = 0;
 for (int k = i; k <= j; k++)
 {
  s += frq[k];
 }
 return s;
}
int mcm(int i, int j, int freq[])
{

 if (j < i)
  return 0;
 if (i == j)
  return freq[i];
 if (dp[i][j] != -1)
  return dp[i][j];
 int sum1 = sum(i, j, freq);
 int ans = INT_MAX;
 for (int k = i; k <= j; k++)
 {
  ans = min(ans, mcm(i, k - 1, freq) + mcm(k + 1, j, freq));
 }
 return dp[i][j] = ans + sum1;
}
int optimalSearchTree(int keys[], int freq[], int n)
{

 memset(dp, -1, sizeof dp);
 return mcm(0, n - 1, freq);
}
```

```
// Evaluation of Postfix expression
int evalRPN(vector<string> &tokens)
{
 stack<int> s;
 for (auto &t : tokens)
  if (t == "+" || t == "-" || t == "*" || t == "/")
  {
   int op1 = s.top();
   s.pop();
   int op2 = s.top();
   s.pop();
   if (t == "+")
    op1 = op2 + op1;
   if (t == "-")
    op1 = op2 - op1;
   if (t == "/")
    op1 = op2 / op1;
   if (t == "*")
    op1 = op2 * op1;
   s.push(op1);
  }
  else
   s.push(stoi(t)); // stoi - converts from
 string to int return s.top();
}

// Prim's Algorithm - BruteForce n2
void prim()
{
 int N, m;
 cin >> N >> m;
 vector<pair<int, int>> adj[N + 1];
 int a, b, wt;
 for (int i = 0; i < m; i++)
 {
  cin >> a >> b >> wt;
  adj[a].push_back({b, wt});
  adj[b].push_back({a, wt});
 }
 int parent[N] = {-1};
 int key[N] = {INT_MAX};
 int mstSet[N] = {0};
 key[0] = 0;
 parent[0] = -1;
 // n-1 edges
 for (int count = 0; count < N - 1; count++)
 {
  int mini = INT_MAX, u;
  for (int v = 0; v < N; v++)
   if (mstSet[v] == false && key[v] < mini)
    mini = key[v], u = v;
  mstSet[u] = 1;
  for (auto it : adj[u])
  {
   int v = it.first;
```

```
   int weight = it.second;
   if (mstSet[v] == false && weight < key[v])
    parent[v] = u, key[v] = weight;
  }
 }
}
// Prim's Algorithm - PriorityQueue nLogn
void prim()
{
 int N, m;
 cin >> N >> m;
 vector<pair<int, int>> adj[N + 1];
 int a, b, wt;
 for (int i = 0; i < m; i++)
 {
  cin >> a >> b >> wt;
  g[a].push_back({b, wt});
  g[b].push_back({a, wt});
 }
 int parent[N] = {-1};
 int key[N] = {INT_MAX};
 int mstSet[N] = {0};
 key[0] = 0;
 parent[0] = -1;
 priority_queue<pair<int, int>, vector<pair<int, int>>,
       greater<pair<int, int>>>
  pq;
 pq.push({0, 0});
 for (int count = 0; count < N - 1; count++)
 {
  int u = pq.top().second;
  pq.pop();
  mstSet[u] = true;
  for (auto it : adj[u])
  {
   int v = it.first;
   int weight = it.second;
   if (mstSet[v] == false && weight < key[v])
   {
    parent[v] = u;
    pq.push({key[v], v});
    key[v] = weight;
   }
  }
 }
}


// Rabin Karp
void search(char pat[], char txt[], int q)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int i, j;
    int p = 0;
```

```
    int t = 0;
    int h = 1;

    for (i = 0; i < M - 1; i++) h = (h * d) % q;
    for (i = 0; i < M; i++){
        p = (d * p + pat[i]) % q;
        t = (d * t + txt[i]) % q;
    }
    for (i = 0; i <= N - M; i++){
        if ( p == t ){
            for (j = 0; j < M; j++)
                if (txt[i+j] != pat[j])
                  break;
            if (j == M)
                cout<<"Pattern found at index "<< i<<endl;
        }
        if ( i < N-M ){
            t = (d*(t - txt[i]*h) + txt[i+M])%q;
            if (t < 0) t = (t + q);
        }
    }
}

// kmp
void KMPSearch(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);
    int lps[M];

    computeLPSArray(pat, M, lps);

    int i = 0;
    int j = 0;
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }

        if (j == M) {
            printf("Found pattern at index %d ", i - j);
            j = lps[j - 1];
        }

        else if (i < N && pat[j] != txt[i]) {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}
void computeLPSArray(char* pat, int M, int* lps)
{
```

```
    int len = 0;
    lps[0] = 0;
    int i = 1;
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            if (len != 0) {
                len = lps[len - 1];
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}
// lps[i] = the longest proper prefix of pat[0..i] which is also a suffix of pat[0..i].
int longestPrefixSuffix(string s)
{
    int n = s.length();

    int lps[n];
    lps[0] = 0;
    int len = 0;

    int i = 1;
    while (i < n)
    {
        if (s[i] == s[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            if (len != 0)
            {
                len = lps[len-1];
            }
            else // if (len == 0)
            {
                lps[i] = 0;
                i++;
            }
        }
    }

    int res = lps[n-1];
```

```
    // Since we are looking for non overlapping parts.
    return (res > n/2)? res/2 : res;
}

 // Booyer Moore
 void badCharHeuristic( string str, int size, int badchar[NO_OF_CHARS])
{
    int i;
    for (i = 0; i < NO_OF_CHARS; i++)
        badchar[i] = -1;
    for (i = 0; i < size; i++)
        badchar[(int) str[i]] = i;
}
void search( string txt, string pat)
{
    int m = pat.size();
    int n = txt.size();

    int badchar[NO_OF_CHARS];
    badCharHeuristic(pat, m, badchar);

    int s = 0;
    while(s <= (n - m))
    {
        int j = m - 1;
        while(j >= 0 && pat[j] == txt[s + j])
            j--;
        if (j < 0){
            cout << "pattern occurs at shift = " <<  s << endl;
            s += (s + m < n)? m-badchar[txt[s + m]] : 1;

        }
        else  s += max(1, j - badchar[txt[s + j]]);
    }
}



// Check preorder is valid or not
    Node *build_tree(int pre[],int n,int &i,int mini,int maxi){
        if(i>=n)return NULL;
        if(pre[i]<mini||pre[i]>maxi)return NULL;
        Node *root=(Node *)malloc(sizeof(Node));

        root->data=pre[i++];

        root->left=build_tree(pre,n,i,mini,root->data);
        root->right=build_tree(pre,n,i,root->data,maxi);
        return root;
    }

    Node* post_order(int pre[], int size)
    {
        int i=0,mini=INT_MIN,maxi=INT_MAX;
        Node* root= build_tree(pre,size,i,mini,maxi);
```

```
        return root;
    }


// Compare Version Numbers
    int compareVersion(string version1, string version2) {
        int i = 0;
        int j = 0;
        int n1 = version1.size();
        int n2 = version2.size();

        int num1 = 0;
        int num2 = 0;
        while(i<n1 || j<n2)
        {
            while(i<n1 && version1[i]!='.'){
                num1 = num1*10+(version1[i]-'0');
                i++;
            }

            while(j<n2 && version2[j]!='.'){
                num2 = num2*10+(version2[j]-'0');;
                j++;
            }

            if(num1>num2) return 1;
            else if(num1 < num2) return -1;

            num1 = 0;
            num2 = 0;
            i++;
            j++;
        }
        return 0;
    }



// Trie

int ALPHABET_SIZE = 26;
struct TrieNode{
    struct TrieNode *children[ALPHABET_SIZE];
    bool isEndOfWord;
};
struct TrieNode *getNode(void){
    struct TrieNode *pNode =  new TrieNode;
    pNode->isEndOfWord = false;
    for (int i = 0; i < ALPHABET_SIZE; i++)
        pNode->children[i] = NULL;
    return pNode;
}
void insert(struct TrieNode *root, string key){
    struct TrieNode *pCrawl = root;
    for (int i = 0; i < key.length(); i++){
```

```
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();
        pCrawl = pCrawl->children[index];
    }
    pCrawl->isEndOfWord = true;
}
bool search(struct TrieNode *root, string key){
    struct TrieNode *pCrawl = root;
    for (int i = 0; i < key.length(); i++){
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            return false;
        pCrawl = pCrawl->children[index];
    }
    return (pCrawl->isEndOfWord);
}

// Find shortest unique prefix for every word in a given list
#define MAX 256
#define MAX_WORD_LEN 500
struct trieNode{
    struct trieNode *child[MAX];
    int freq;
};
struct trieNode *newTrieNode(void)
{
    struct trieNode *newNode = new trieNode;
    newNode->freq   = 1;
    for (int i = 0; i<MAX; i++)
        newNode->child[i] = NULL;
    return newNode;
}
void insert(struct trieNode *root, string str){
    int len = str.length();
    struct trieNode *pCrawl = root;

    for (int level = 0; level<len; level++){
        int index = str[level];
        if (!pCrawl->child[index])
            pCrawl->child[index] = newTrieNode();
        else
           (pCrawl->child[index]->freq)++;
        pCrawl = pCrawl->child[index];
    }
}
void findPrefixesUtil(struct trieNode *root, char prefix[], int ind){
    if (root == NULL) return;
    if (root->freq == 1){
       prefix[ind] = '\0';
       cout << prefix << " ";
       return;
    }
    for (int i=0; i<MAX; i++){
       if (root->child[i] != NULL){
```

```
            prefix[ind] = i;
            findPrefixesUtil(root->child[i], prefix, ind+1);
        }
    }
}
void findPrefixes(string arr[], int n){
    struct trieNode *root = newTrieNode();
    root->freq = 0;
    for (int i = 0; i<n; i++)
        insert(root, arr[i]);

    char prefix[MAX_WORD_LEN];
    findPrefixesUtil(root, prefix, 0);
}



//Word Break
const int ALPHABET_SIZE = 26;
struct TrieNode {
    struct TrieNode* children[ALPHABET_SIZE];
    bool isEndOfWord;
};
struct TrieNode* getNode(void)
{
    struct TrieNode* pNode = new TrieNode;
    pNode->isEndOfWord = false;
    for (int i = 0; i < ALPHABET_SIZE; i++)
        pNode->children[i] = NULL;
    return pNode;
}
void insert(struct TrieNode* root, string key)
{
    struct TrieNode* pCrawl = root;
    for (int i = 0; i < key.length(); i++) {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();
        pCrawl = pCrawl->children[index];
    }
    pCrawl->isEndOfWord = true;
}
bool search(struct TrieNode* root, string key)
{
    struct TrieNode* pCrawl = root;
    for (int i = 0; i < key.length(); i++) {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            return false;
        pCrawl = pCrawl->children[index];
    }
    return (pCrawl != NULL && pCrawl->isEndOfWord);
}
bool wordBreak(string str, TrieNode* root)
{
```

```
    int size = str.size();
    if (size == 0) return true;
    for (int i = 1; i <= size; i++) {
        if (search(root, str.substr(0, i))
            && wordBreak(str.substr(i, size - i), root))
            return true;
    }
    return false;
}
```

```
// print all anagrams together
 #define NO_OF_CHARS 26
struct IndexNode{
    int index;
    struct IndexNode* next;
};
struct TrieNode
{
    bool isEnd;  // indicates end of word
    struct TrieNode* child[NO_OF_CHARS]; // 26 slots each for 'a' to 'z'
    struct IndexNode* head; // head of the index list
};
struct TrieNode* newTrieNode()
{
    struct TrieNode* temp = new TrieNode;
    temp->isEnd = 0;
    temp->head = NULL;
    for (int i = 0; i < NO_OF_CHARS; ++i)
        temp->child[i] = NULL;
    return temp;
}
int compare(const void* a, const void* b)
{  return *(char*)a - *(char*)b; }
struct IndexNode* newIndexNode(int index)
{
    struct IndexNode* temp = new IndexNode;
    temp->index = index;
    temp->next = NULL;
    return temp;
}
void insert(struct TrieNode** root, char* word, int index)
{
    if (*root == NULL)
        *root = newTrieNode();
    if (*word != '\0')
        insert( &( (*root)->child[tolower(*word) - 'a'] ), word+1, index );
    else  // If end of the word reached
    {
        if ((*root)->isEnd)
        {
            IndexNode* pCrawl = (*root)->head;
            while( pCrawl->next )
                pCrawl = pCrawl->next;
```

```
            pCrawl->next = newIndexNode(index);
        }
        else
        {
            (*root)->isEnd = 1;
            (*root)->head = newIndexNode(index);
        }
    }
}
void printAnagramsUtil(struct TrieNode* root, char *wordArr[])
{
    if (root == NULL) return;
    if (root->isEnd)
    {
        IndexNode* pCrawl = root->head;
        while (pCrawl != NULL)
        {
            printf( "%s ", wordArr[ pCrawl->index ] );
            pCrawl = pCrawl->next;
        }
    }
    for (int i = 0; i < NO_OF_CHARS; ++i)
        printAnagramsUtil(root->child[i], wordArr);
}
void printAnagramsTogether(char* wordArr[], int size)
{
    struct TrieNode* root = NULL;
    for (int i = 0; i < size; ++i){
        int len = strlen(wordArr[i]);
        char *buffer = new char[len+1];
        strcpy(buffer, wordArr[i]);
        qsort( (void*)buffer, strlen(buffer), sizeof(char), compare );
        insert(&root,  buffer, i);
    }
    printAnagramsUtil(root, wordArr);
}

// Phone Directory
 struct Node{
      Node* vec[26] = {NULL};
      vector<string> list;
   };

   struct trie{
      Node* root = new Node;
      void insert(string contact[], int n){
          unordered_map<string,bool>mp;
          for(int i=0; i<n; i++){
              Node* temp = root;
              string str = contact[i];
              if(mp.find(str) == mp.end()){
                  mp[str] = true;
                  for(auto x:str){
                      if(!temp->vec[x-'a']) temp->vec[x-'a'] = new Node;
                      temp = temp->vec[x-'a'];
```

```
                    temp->list.push_back(str);
                }
            }
        }
    }

    vector<vector<string> > search(string str){
        vector<vector<string> > ans;
        Node* temp = root;
        for(auto x:str){
            if(temp == NULL){
                ans.push_back({"0"});
            }
            else if(!temp->vec[x-'a']){
                ans.push_back({"0"});
                temp = temp->vec[x-'a'];
            }
            else{
                temp = temp->vec[x-'a'];
                ans.push_back(temp->list);
            }
        }
        return ans;
    }
};

vector<vector<string>> displayContacts(int n, string contact[], string s)
{
    trie t;
    sort(contact, contact + n);
    t.insert(contact,n);
    return t.search(s);
}
```

```
// Kernighan's bit count algorithm
int getBits(int x) {
 int count = 0;
 while (x) {
  x &= x - 1;
  count++;
 }
 return count;
}
vector<int> countBits(int n) {
 vector<int> res(n + 1, 0);
 for (int x = 1; x <= n; x++)
  res[x] = res[x / 2] + (x % 2);
 return res;
}
// Count total set bits in all numbers from 1 to n
int countSetBits(int n)
{
 if(!n) return 0;
```

```
 int bits = log2(n+1), x=1<<bits;
 bits*= (x>>1);
 if(n<x) return bits;
 return bits+ (n-x+1) + countSetBits(n-x);
}
```

```
// Copy Set Bits In A Range
  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();
    int left = scn.nextInt();
    int right = scn.nextInt();

    int mask = (1 << (right - left + 1)) - 1;
    mask = ((mask << (left - 1)) & a);
    b |= mask;
    System.out.println(b);

  }
```

```
//Interleaved strings
         int dp[200][200];
       int f(string s1, string s2, string s3, int ind1, int ind2){
       if(ind1==-1 && ind2==-1) return 1;
       int cur;
       if(ind1<0 && ind2>=0){
           if(s2.substr(0,ind2+1)==s3.substr(0,ind2+1)) return 1;
           else return 0;
       }

       if(ind2<0 && ind1>=0){
           if(s1.substr(0,ind1+1)==s3.substr(0,ind1+1)) return 1;
           else return 0;
       }

       if(dp[ind1][ind2]!=-1) return dp[ind1][ind2];
       int op1=0,op2=0;

       cur=ind1+ind2+1;
       if(s3[cur]==s1[ind1]) op1=f(s1, s2, s3, ind1-1, ind2);
       if(s3[cur]==s2[ind2]) op2=f(s1, s2, s3, ind1, ind2-1);

       return dp[ind1][ind2]= (op1||op2);
    }
    bool isInterleave(string A, string B, string C)
    {
       //Your code heref
       int n=A.size(), m=B.size();
       memset(dp, -1, sizeof(dp));
       return f(A, B, C, n-1, m-1);
    }
```

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
int main()
{
    return 0;
}

// Coin Change - 2
long long int count(int S[], int m, int n)
{
    vector<vector<long long int>> dp((m + 1), vector<long long int>(n + 1, 0));
    for (int i = 0; i <= m; i++)
        dp[i][0] = 1;
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (S[i - 1] > j)
                dp[i][j] = dp[i - 1][j];
            else
                dp[i][j] = dp[i - 1][j] + dp[i][(j - S[i - 1])];
        }
    }
    return dp[m][n];
}
long long int count(int s[], int m, int n)
{
    ll dp[n + 1] = {0};
    dp[0] = 1;
    for (int i = 0; i < m; i++)
    {
        for (int j = s[i]; j <= n; j++)
        {
            if (j - s[i] >= 0)
            {
                dp[j] += dp[j - s[i]];
            }
        }
    }
    return dp[n];
}

// 0 - 1 Knapsack Problem
// Unbounded Knapsack (Repetition of items allowed)
int memoization(vector<vector<int>> &dp, int W, int wt[], int val[], int n)
{
    if (n == 0 || W == 0)
        return 0;
    if (dp[n][W] != -1)
        return dp[n][W];
    if (wt[n - 1] <= W)
    {
        return dp[n][W] = max(memoization(dp, W, wt, val, n - 1), val[n - 1] + memoization(dp, W - wt[n - 1], wt, val,
n - 1));
```

```
    }
    else
        return dp[n][W] = memoization(dp, W, wt, val, n - 1);
}
int knapSack(int W, int wt[], int val[], int n)
{
    vector<vector<int>> dp(n + 1, vector<int>(W + 1, -1));
    return memoization(dp, W, wt, val, n);
}
int knapSack(int w, int wt[], int val[], int n)
{
    int t[n + 1][w + 1];
    for (int i = 0; i < n + 1; i++)
        for (int j = 0; j < w + 1; j++)
            if (i == 0 || j == 0)
                t[i][j] = 0;
            else if (wt[i - 1] <= j)
                t[i][j] = max(val[i - 1] + t[i - 1][j - wt[i - 1]], t[i - 1][j]);
            else
                t[i][j] = t[i - 1][j];
    return t[n][w];
}

// Binomial Coefficient
int nCr(int n, int k)
{
    int M = 1e9 + 7;
    if (n < k)
        return 0;
    long long int C[n + 1][k + 1];
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= min(i, k); j++)
        {
            if (j == 0 || j == i)
                C[i][j] = 1;
            else
                C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % M;
        }
    }
    return (int)C[n][k];
}
int binomialCoeffUtil(int n, int k, int **dp)
{
    if (dp[n][k] != -1)
        return dp[n][k];
    if (k == 0 || k == n)
    {
        dp[n][k] = 1;
        return dp[n][k];
    }
    dp[n][k] = binomialCoeffUtil(n - 1, k - 1, dp) +
            binomialCoeffUtil(n - 1, k, dp);
    return dp[n][k];
}
```

```
// Permutation Coefficient
int permutationCoeff(int n, int k)
{
    int P[n + 1][k + 1];
    memset(P, 0, sizeof(P));
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= min(i, k); j++)
        {
            if (j == 0)
                P[i][j] = 1;
            else
                P[i][j] = P[i - 1][j] +
                          (j * P[i - 1][j - 1]);
        }
    }
    return P[n][k];
}
// catalan no
// recursive
unsigned long int catalan(unsigned int n)
{
    if (n <= 1)
        return 1;
    unsigned long int res = 0;
    for (int i = 0; i < n; i++)
        res += (catalan(i) * catalan(n - i - 1));
    return res;
}
// dp
unsigned long int catalanDP(unsigned int n)
{
    unsigned long int catalan[n + 1];
    catalan[0] = catalan[1] = 1;
    for (int i = 2; i <= n; i++)
    {
        catalan[i] = 0;
        for (int j = 0; j < i; j++)
            catalan[i] += catalan[j] * catalan[i - j - 1];
    }
    return catalan[n];
}

int f(int i, int j, int arr[], vector<vector<int>> &dp)
{
    if (i == j)
        return 0;
    if (dp[i][j] != -1)
        return dp[i][j];
    int mini = 1e9;
    for (int k = i; k < j; k++)
    {
        int steps = arr[i - 1] * arr[k] * arr[j] + f(i, k, arr, dp) + f(k + 1, j, arr, dp);
        mini = min(mini, steps);
```

```
    }
    return dp[i][j] = mini;
}

// Matrix Chain Multiplication
int matrixMultiplication(int N, int arr[])
{
    vector<vector<int>> dp(N, vector<int>(N, -1));
    return f(1, N - 1, arr, dp);
}
int matrixMultiplication(int n, int arr[])
{
    int dp[n - 1][n - 1];
    for (int g = 0; g < n - 1; g++)
    {
        for (int i = 0, j = g; j < n - 1; i++, j++)
        {
            if (g == 0)
                dp[i][j] = 0;
            else if (g == 1)
                dp[i][j] = arr[i] * arr[j] * arr[j + 1];
            else
            {
                int mincost = INT_MAX;
                for (int k = i; k < j; k++)
                {
                    int lc = dp[i][k];
                    int rc = dp[k + 1][j];
                    int mc = arr[i] * arr[k + 1] * arr[j + 1];
                    int tc = lc + rc + mc;
                    mincost = min(tc, mincost);
                }
                dp[i][j] = mincost;
            }
        }
    }
    return dp[0][n - 2];
}

// Edit Distance
int f(int i, int j, string s1, string s2, vector<vector<int>> &dp)
{
    if (i < 0 && j < 0)
    {
        return 0;
    }
    if (i < 0)
        return j + 1;
    if (j < 0)
        return i + 1;
    if (dp[i][j] != -1)
        return dp[i][j];
    if (s1[i] == s2[j])
        return dp[i][j] = f(i - 1, j - 1, s1, s2, dp);
    return dp[i][j] = 1 + min({f(i - 1, j, s1, s2, dp),
```

```
                    f(i, j - 1, s1, s2, dp),
                    f(i - 1, j - 1, s1, s2, dp)});
}

int editDistance(string s, string t)
{
    vector<vector<int>> dp(s.size(), vector<int>(t.size(), -1));
    return f(s.size() - 1, t.size() - 1, s, t, dp);
}

int editDistance(string s, string t)
{
    int n = s.length();
    int m = t.length();
    int dp[n + 1][m + 1];
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= m; j++)
        {
            if (i == 0)
                dp[i][j] = j;
            else if (j == 0)
                dp[i][j] = i;
            else if (s[i - 1] == t[j - 1])
                dp[i][j] = dp[i - 1][j - 1];
            else
                dp[i][j] = 1 + min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]});
        }
    }
    return dp[n][m];
}

// Partition Equal Subset Sum
bool f(int arr[], int idx, int sum, vector<vector<int>> &dp)
{
    if (sum < 0 || idx < 0)
        return 0;
    if (sum == 0)
        return 1;
    if (dp[idx][sum] != -1)
        return dp[idx][sum];
    if (arr[idx] > sum)
        return dp[idx][sum] = f(arr, idx - 1, sum, dp);
    return dp[idx][sum] = f(arr, idx - 1, sum - arr[idx], dp) || f(arr, idx - 1, sum, dp);
}
int equalPartition(int N, int arr[])
{
    // code here
    int sum = 0;
    for (int i = 0; i < N; i++)
        sum += arr[i];
    if (sum & 1)
        return 0;
    vector<vector<int>> dp(N + 1, vector<int>(sum, -1));
    return f(arr, N - 1, sum >> 1, dp);
```

```
}

int equalPartition(int n, int arr[])
{
    int s = 0;
    for (int i = 0; i < n; i++)
        s += arr[i];
    if (s & 1)
        return 0;
    vector<vector<bool>> dp(n + 1, vector<bool>(s / 2, false));
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= s / 2; j++)
        {
            if (j < arr[i - 1])
                dp[i][j] = dp[i - 1][j];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - arr[i - 1]]);
        }
    }
    return dp[n][s / 2];
}

// Friends Pairing Problem
long long dp[10001];
long long fun(int n)
{
    const int M = 1000000007;
    if (n <= 2)
        return n;
    if (dp[n] != -1)
        return dp[n];
    return dp[n] = (fun(n - 1) % M + (n - 1) * (fun(n - 2) % M)) % M;
}

int countFriendsPairings(int n)
{
    memset(dp, -1, sizeof(dp));
    return (int)fun(n);
}
int countFriendsPairings(int n)
{
    long long int dp[n + 1];
    const int M = 1000000007;
    for (int i = 0; i <= n; i++)
    {
        if (i <= 2)
            dp[i] = i;
        else
            dp[i] = (dp[i - 1] % M + ((i - 1) * dp[i - 2]) % M) % M;
    }
    return dp[n];
}
```

```
// Gold Mine Problem
int solve(int i, int j, int m, int n, vector<vector<int>> &dp, vector<vector<int>> &M)
{
    if (i < 0 or i >= n or j >= m)
    {
        return 0;
    }
    if (dp[i][j] != -1)
    {
        return dp[i][j];
    }
    int a = M[i][j] + solve(i - 1, j + 1, m, n, dp, M);
    int b = M[i][j] + solve(i, j + 1, m, n, dp, M);
    int c = M[i][j] + solve(i + 1, j + 1, m, n, dp, M);
    return dp[i][j] = max(a, max(b, c));
}
int maxGold(int n, int m, vector<vector<int>> M)
{
    // code here
    vector<vector<int>> dp(n, vector<int>(m, -1));
    int s = 0;
    for (int i = 0; i < n; i++)
    {
        s = max(s, solve(i, 0, m, n, dp, M));
    }
    return s;
}
int maxGold(int n, int m, vector<vector<int>> M)
{
    int dp[51][51] = {{0}};
    for (int i = 0; i < n; i++)
    {
        dp[i][0] = M[i][0];
    }
    for (int j = 1; j < m; j++)
    {
        for (int i = 0; i < n; i++)
        {
            dp[i][j] = M[i][j] + max((i - 1 >= 0 ? dp[i - 1][j - 1] : 0),
                                 max(dp[i][j - 1], (i + 1 < n ? dp[i + 1][j - 1] : 0)));
        }
    }
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        res = max(res, dp[i][m - 1]);
    }
    return res;
}

// Assembly Line Scheduling
int carAssembly(vector<vector<int>> &a, vector<vector<int>> &T, vector<int> &e, vector<int> &x)
{
    int upper = e[0] + a[0][0];
    int lower = e[1] + a[1][0];
```

```
    for (int i = 1; i < a[0].size(); i++)
    {
        int tmp = upper;
        upper = a[0][i] + min(upper, lower + T[1][i]);
        lower = a[1][i] + min(lower, tmp + T[0][i]);
    }
    return min(upper + x[0], lower + x[1]);
}

// Painting the Fenceproblem
long long countWays(int n, int k)
{
    // n wall, k color, no more than two consecutive
    //      total[i] = same[i] + diff[i]
    //      same[i] = diff[i-1]
    //      diff[i] = (diff[i-1] + diff[i-2]) * (k-1) = total[i-1] * (k-1)
    long long dp[n + 1];
    memset(dp, 0, sizeof(dp));
    long long mod = 1000000007;
    dp[1] = k;
    dp[2] = k * k;
    for (int i = 3; i <= n; i++)
    {
        dp[i] = ((k - 1) * (dp[i - 1] + dp[i - 2])) % mod;
    }
    return dp[n];
}



// Minimum removals from array to make max – min <= K
int countRemovals(int a[], int i, int j, int k)
{
    if (i >= j) return 0;
    else if ((a[j] - a[i]) <= k) return 0;
    else if (dp[i][j] != -1) return dp[i][j];
    else if ((a[j] - a[i]) > k) {
        dp[i][j] = 1 + min(countRemovals(a, i + 1, j, k),countRemovals(a, i, j - 1, k));
    }
    return dp[i][j];
}

// Longest Common Subsequence
int fun(int x, int y, string s1, string s2, vector<vector<int>> &v)
{
    if (x == 0 || y == 0)
        return 0;
    if (v[x][y] != -1)
        return v[x][y];
    if (s1[x - 1] == s2[y - 1])
        return v[x][y] = 1 + fun(x - 1, y - 1, s1, s2, v);
    return v[x][y] = max(fun(x - 1, y, s1, s2, v), fun(x, y - 1, s1, s2, v));
}
int lcs(int x, int y, string s1, string s2)
{
```

```
    vector<vector<int>> v(x + 1, vector<int>(y + 1, -1));
    return fun(x, y, s1, s2, v);
}
int lcs(int x, int y, string s1, string s2)
{
    int dp[x + 1][y + 1];
    for (int i = 0; i <= x; i++)
        dp[i][0] = 0;
    for (int j = 0; j <= y; j++)
        dp[0][j] = 0;
    for (int i = 1; i <= x; i++)
        for (int j = 1; j <= y; j++)
            if (s1[i - 1] == s2[j - 1])
                dp[i][j] = 1 + dp[i - 1][j - 1];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
    return dp[x][y];
}

// Longest Repeated Subsequence
int LongestRepeatingSubsequence(string s)
{
    int n = s.size();
    vector<vector<int>> dp(n + 1, vector<int>(n + 1, 0));
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (s[i - 1] == s[j - 1] && i != j)
                dp[i][j] = 1 + dp[i - 1][j - 1];
            else
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
    return dp[n][n];
}

// Longest Increasing Subsequence
int longestSubsequence(int n, int a[])
{
    int dp[n] = {1};
    int mx = 0;
    for (int i = 1; i < n; i++)
        for (int j = 0; j < i; j++)
            if (a[i] > a[j])
                dp[i] = max(1 + dp[j], dp[i]);

    mx = *max_element(dp, dp + n);
    return mx;
}
int longestSubsequence(int n, int a[])
{
    vector<int> lis;
    for (int i = 0; i < n; i++)
    {
        int pos = lower_bound(lis.begin(), lis.end(), a[i]) - lis.begin();
        if (pos == lis.size())
            lis.push_back(a[i]);
        else
```

```
        lis[pos] = a[i];
    }
    return lis.size();
}

// LCS (Longest Common Subsequence) of three strings
int LCSof3(string A, string B, string C, int n1, int n2, int n3)
{
    int dp[n1 + 1][n2 + 1][n3 + 1];

    for (int i = 0; i < n1 + 1; i++)
    {
        for (int j = 0; j < n2 + 1; j++)
        {
            for (int k = 0; k < n3 + 1; k++)
            {

                if (i == 0 || j == 0 || k == 0)
                {
                    dp[i][j][k] = 0;
                }
                else if (A[i - 1] == B[j - 1] && B[j - 1] == C[k - 1])
                {
                    dp[i][j][k] = 1 + dp[i - 1][j - 1][k - 1];
                }
                else
                {

                    int a = max(dp[i - 1][j][k], dp[i][j - 1][k]);
                    int b = max(dp[i][j - 1][k], dp[i][j][k - 1]);
                    int c = max(dp[i - 1][j][k], dp[i][j][k - 1]);
                    dp[i][j][k] = max(max(a, b), c);
                }
            }
        }
    }

    return dp[n1][n2][n3];
}

// Maximum sum increasing subsequence
int maxSumIS(int arr[], int n)
{
    int i, j, max = 0;
    vector<int> msis(n, 0);
    for (i = 0; i < n; i++)
        msis[i] = arr[i];
    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
            if (arr[i] > arr[j] && msis[i] < msis[j] + arr[i])
                msis[i] = msis[j] + arr[i];
    return *max_element(msis.begin(), msis.end());
}

// Count all subarray having product less than K
```

```
int countSubArrayProductLessThanK(const vector<int> &a, int n, long long k)
{
    long long prod = 1;
    int i = 0, j = 0, ans = 0;
    while (i < n && j < n)
    {
        prod *= a[j];
        while (prod >= k && i < j)
        {
            prod /= a[i++];
        }
        if (prod < k)
            ans += (j - i + 1);
        j++;
    }
    return ans;
}

// Count all subsequences having product less than K way 2
int productSubSeqCount(vector<int> &arr, int k)
{
    int n = arr.size();
    int dp[k + 1][n + 1];
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= k; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            dp[i][j] = dp[i][j - 1];
            if (arr[j - 1] <= i)
                dp[i][j] += dp[i / arr[j - 1]][j - 1] + 1;
        }
    }
    return dp[k][n];
}

// Longest subsequence such that difference between adjacent is one
int longestSubseqWithDiffOne(int arr[], int n)
{
    int dp[n];
    for (int i = 0; i < n; i++)
        dp[i] = 1;
    for (int i = 1; i < n; i++)
    {
        for (int j = 0; j < i; j++)
        {
            if (abs(arr[i] - arr[j]) <= 1)
                dp[i] = max(dp[i], dp[j] + 1);
        }
    }
    return *max_element(dp, dp + n);
}

// Maximum subsequence sum such that no three are consecutive
int mx3(int n, int sum[], int arr[])
```

```
{
    if (sum[n] != -1)
        return sum[n];
    if (n == 0)
        return sum[n] = 0;
    if (n == 1)
        return sum[n] = arr[0];
    if (n == 2)
        return sum[n] = arr[1] + arr[0];
    return sum[n] = max({mx3(n - 1, sum, arr), mx3(n - 2, sum, arr) + arr[n],
                        arr[n] + arr[n - 1] + mx3(n - 3, sum, arr)});
}

int maxSumWO3Consec(int arr[], int n)
{
    int sum[n];
    if (n >= 1)
        sum[0] = arr[0];
    if (n >= 2)
        sum[1] = arr[0] + arr[1];
    if (n > 2)
        sum[2] = max(sum[1], max(arr[1] + arr[2], arr[0] + arr[2]));
    for (int i = 3; i < n; i++)
        sum[i] = max({sum[i - 1], sum[i - 2] + arr[i],
                     arr[i] + arr[i - 1] + sum[i - 3]});
    return sum[n - 1];
}

// Egg Dropping Problem
int solveEggDrop(int n, int k)
{
    if (memo[n][k] != -1)
        return memo[n][k];
    if (k == 1 || k == 0)
        return k;
    if (n == 1)
        return k;
    int min = INT_MAX, x, res;
    for (x = 1; x <= k; x++)
    {
        res = max(
            solveEggDrop(n - 1, x - 1),
            solveEggDrop(n, k - x));
        if (res < min)
            min = res;
    }
    memo[n][k] = min + 1;
    return min + 1;
}
int eggDrop(int n, int k)
{
    int eggFloor[n + 1][k + 1];
    int res;
    int i, j, x;
    for (i = 1; i <= n; i++)
```

```
    {
        eggFloor[i][1] = 1;
        eggFloor[i][0] = 0;
    }
    for (j = 1; j <= k; j++)
        eggFloor[1][j] = j;

    for (i = 2; i <= n; i++)
    {
        for (j = 2; j <= k; j++)
        {
            eggFloor[i][j] = INT_MAX;
            for (x = 1; x <= j; x++)
            {
                res = 1 + max(eggFloor[i - 1][x - 1], eggFloor[i][j - x]);
                if (res < eggFloor[i][j])
                    eggFloor[i][j] = res;
            }
        }
    }
    return eggFloor[n][k];
}

// Maximum Length Chain of Pairs
int maxChainLen(struct val p[], int n)
{
    sort(p, p + n, [](auto &a, auto &b)
         { return a.first < b.first; });
    vector<int> F(n, 1);
    for (int i = 1; i < n; i++)
        for (int j = 0; j < i; j++)
            if (p[j].second < p[i].first)
                F[i] = max(F[i], F[j] + 1);
    return *max_element(F.begin(), F.end());
}
// Maximum Length Chain of Pairs new way
bool cmp(vector<int> &a, vector<int> &b)
{
    if (a[1] == b[1])
        return a[0] < b[0];
    return a[1] < b[1];
}
int findLongestChain(vector<vector<int>> &pairs)
{
    sort(pairs.begin(), pairs.end(), cmp);
    int r = pairs[0][1], ans = 1;
    for (auto it : pairs)
        if (it[0] > r)
            r = it[1], ans++;
    return ans;
}
```

```
// Maximum sum of pairs with specific difference
int maxSumPairWithDifferenceLessThanK(int a[], int n, int k)
{
    sort(a, a + n);
    int sum = 0;
    for (int i = n - 1; i > 0; i--)
    {
        if (a[i] - a[i - 1] < k)
        {
            sum += (a[i] + a[i - 1]);
            i--;
        }
    }
    return sum;
}

// Maximum falling path sum in matrix
int maximumPath(int n, vector<vector<int>> v)
{
    if (n == 1)
        return v[0][0];
    int dp[n][n];
    for (int i = 0; i < n; i++)
        dp[0][i] = v[0][i];
    int ans = INT_MIN;
    for (int i = 1; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j == 0)
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j + 1]) + v[i][j];
            else if (j == n - 1)
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - 1]) + v[i][j];
            else
                dp[i][j] = v[i][j] + max(dp[i - 1][j], max(dp[i - 1][j - 1], dp[i - 1][j + 1]));
            if (i == n - 1)
                ans = max(ans, dp[i][j]);
        }
    }
    return ans;
}

int solve(vector<vector<int>> &Matrix, int N, int i, int j, vector<vector<int>> &memo)
{
    if (i < 0 or i >= N or j < 0 or j >= N)
        return 0;
    if (memo[i][j] != -1)
        return memo[i][j];

    int down = solve(Matrix, N, i + 1, j, memo);
    int right = solve(Matrix, N, i + 1, j + 1, memo);
    int left = solve(Matrix, N, i + 1, j - 1, memo);

    return memo[i][j] = Matrix[i][j] + max({down, right, left});
}
```

```
int maximumPath(int N, vector<vector<int>> Matrix)
{
   vector<vector<int>> memo(N, vector<int>(N, -1));
   int result = 0;
   for (int i = 0; i < N; i++)
      result = max(result, solve(Matrix, N, 0, i, memo));
   return result;
}

// Maximum difference of zeros and ones in binary string
int maxSubstring(string str)
{
   int ans = -1e9, sum = 0, cnt = 0;
   for (auto &it : str)
   {
      sum += it == '1' ? -1 : 1;
      ans = max(ans, sum);
      if (sum < 0)
         sum = 0;
   }
   return ans;
}

// Minimum cost to fill given weight in a bag
// 1. cost[i] = -1 means that 'i' kg packet of orange is unavailable
// 2. It may be assumed that there is infinite supply of all available packet types.
long long int helper(int cost[], int n, int w)
{
   if (w == 0)
      return 0;
   if (n == 0 || w < 0)
      return INT_MAX;
   if (dp[n][w] != -1)
      return dp[n][w];
   if (cost[n - 1] != -1)
      return dp[n][w] = min(cost[n - 1] + helper(cost, n, w - n), helper(cost, n - 1, w));
   else
      return dp[n][w] = helper(cost, n - 1, w);
}
int minimumCost(int cost[], int N, int W)
{
   // Your code goes here
   memset(dp, -1, sizeof(dp));
   return helper(cost, N, W);
}

int longestCommonSubstr(string s1, string s2, int n, int m)
{
   int n1 = n, n2 = m, mx = 0;
   int arr[n1 + 1][n2 + 1];
   for (int i = 0; i < n1 + 1; i++)
   {
      for (int j = 0; j < n2 + 1; j++)
      {
         if (i == 0 || j == 0)
```

```
            arr[i][j] = 0;
        else if (s1[i - 1] == s2[j - 1])
            arr[i][j] = arr[i - 1][j - 1] + 1;
        else
            arr[i][j] = 0;
        mx = max(mx, arr[i][j]);
      }
   }
   return mx;
}

// Count Balanced Binary Trees of Height h
long long int countBT(int h)
{
   int mod = 1e9 + 7;
   long long int height[h + 1];
   height[0] = 1;
   height[1] = 1;
   for (int i = 2; i <= h; i++)
      height[i] = ((height[i - 1] * height[i - 1]) % mod + (2 * height[i - 1] * height[i - 2]) % mod) % mod;
   return height[h];
}

// Longest Palindromic Subsequence
int fun(string &A, vector<vector<int>> &v, int i, int j)
{
   if (i == j)
      return 1;
   if (i > j)
      return 0;
   if (v[i][j] != -1)
      return v[i][j];
   if (A[i] == A[j])
      return v[i][j] = 2 + fun(A, v, i + 1, j - 1);
   else
      return v[i][j] = max(fun(A, v, i + 1, j), fun(A, v, i, j - 1));
}
int longestPalinSubseq(string A)
{
   int n = A.length();
   vector<vector<int>> v(n, vector<int>(n, -1));
   return fun(A, v, 0, n - 1);
}
int longestPalinSubseq(string A)
{
   int n = A.length();
   vector<vector<int>> v(n, vector<int>(n, 0));
   for (int i = n - 1; i >= 0; i--)
   {
      for (int j = i; j < n; j++)
      {
         if (i == j)
            v[i][j] = 1;
         else if (A[i] == A[j])
            v[i][j] = 2 + v[i + 1][j - 1];
```

```
            else
                v[i][j] = max(v[i][j - 1], v[i + 1][j]);
        }
    }
    return v[0][n - 1];
}

// Count Palindromic Subsequences
long long int f(string &s, int l, int r)
{
    const long long int M = 1e9 + 7;
    if (l > r)
        return 0;
    if (dp[l][r] != -1)
        return dp[l][r];
    if (s[l] == s[r])
        return dp[l][r] = (1 + f(s, l + 1, r, dp) + f(s, l, r - 1, dp)) % M;
    else
        return dp[l][r] = (M + f(s, l + 1, r, dp) + f(s, l, r - 1, dp) - f(s, l + 1, r - 1, dp)) % M;
}
long long int countPS(string str)
{
    int dp[1001][1001];
    memset(dp, -1, sizeof(dp));
    return f(str, 0, str.size() - 1, dp);
}

// Longest Palindromic Substring
string longestPalindrome(string s)
{
    int n = s.length();
    if (n == 1)
        return s;
    bool dp[n][n];
    int start = 0, end = 0;
    for (int g = 0; g < s.length(); g++)
    {
        for (int i = 0, j = g; j < s.length(); i++, j++)
        {
            if (g == 0)
                dp[i][j] = true;
            else if (g == 1)
            {
                if (s[i] == s[j])
                    dp[i][j] = true;
                else
                    dp[i][j] = false;
            }
            else
            {
                if (s[i] == s[j] && dp[i + 1][j - 1] == true)
                    dp[i][j] = true;
                else
                    dp[i][j] = false;
            }
```

```
            if (dp[i][j] == true)
            {
                start = i;
                end = g + 1;
            }
        }
    }
    return s.substr(start, end);
}

// Longest alternating subsequence
int AlternatingaMaxLength(vector<int> &arr)
{
    int inc = 1, dec = 1, n = arr.size();
    for (int i = 1; i < n; i++)
        if (arr[i] > arr[i - 1])
            inc = dec + 1;
        else if (arr[i] < arr[i - 1])
            dec = inc + 1;
    return max(inc, dec);
}

// Mobile Numeric Keypad Problem //bishop, knight
int solution(int n)
{
    vector<vector<int>> dp(n + 1, vector<int>(10, 0));
    vector<vector<int>> data = {
        // where i can go for each data
        {0, 8},
        {1, 2, 4},
        {1, 2, 3, 5},
        {2, 3, 6},
        {1, 4, 5, 7},
        {2, 4, 5, 6, 8},
        {3, 5, 6, 9},
        {4, 7, 8} {5, 7, 8, 9, 0}};
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= 9; j++)
        {
            if (i == 1)
                dp[i][j] = 1;
            else
            {
                for (int prev : data[j])
                {
                    dp[i][j] += dp[i - 1][prev];
                }
            }
        }
    }
    int sum = 0;
    for (int j = 0; j <= 9; j++)
        sum += dp[n][j];
    return sum;
```

```
}

// Letter Combinations of a Phone Number
vector<string> letterCombinations(string digits)
{
    vector<string> mappings = {"abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"}, ans;
    if (digits == "")
        return {};
    string combination = "";
    helper(digits, 0, combination, mappings, ans);
    return ans;
}
void helper(string &digits, int i, string &combi, vector<string> &mappings, vector<string> &ans)
{
    if (i == digits.size())
    {
        ans.push_back(combi);
        return;
    }
    for (auto &c : mappings[digits[i] - '2'])
    {
        combi.push_back(c);
        helper(digits, i + 1, combi, mappings, ans);
        combi.pop_back();
    }
}

int dp[200][200];
int f(string s1, string s2, string s3, int ind1, int ind2)
{
    if (ind1 == -1 && ind2 == -1)
        return 1;
    int cur;
    if (ind1 < 0 && ind2 >= 0)
    {
        if (s2.substr(0, ind2 + 1) == s3.substr(0, ind2 + 1))
            return 1;
        else
            return 0;
    }
    if (ind2 < 0 && ind1 >= 0)
    {
        if (s1.substr(0, ind1 + 1) == s3.substr(0, ind1 + 1))
            return 1;
        else
            return 0;
    }

    if (dp[ind1][ind2] != -1)
        return dp[ind1][ind2];
    int op1 = 0, op2 = 0;

    cur = ind1 + ind2 + 1;
    if (s3[cur] == s1[ind1])
        op1 = f(s1, s2, s3, ind1 - 1, ind2);
```

```
    if (s3[cur] == s2[ind2])
        op2 = f(s1, s2, s3, ind1, ind2 - 1);

    return dp[ind1][ind2] = (op1 || op2);
}
bool isInterleave(string A, string B, string C)
{
    int n = A.size(), m = B.size();
    memset(dp, -1, sizeof(dp));
    return f(A, B, C, n - 1, m - 1);
}

// Find shortest safe route in a path with landmines
vector<int> dir = {0, 1, 0, -1, 0};
int mn = INT_MAX;
void trv(int r, int c, int n, int m, int dist, vector<vector<int>> &grid, vector<vector<int>> &vis)
{
    if (c == m - 1)
        mn = min(mn, dist);
    if (dist > mn)
        return;
    vis[r][c] = 1;
    for (int d = 0; d < 4; d++)
    {
        int nr = r + dir[d], nc = c + dir[d + 1];
        if (nr < n && nc < m && nr > -1 && nc > -1 && vis[nr][nc] != 1 && grid[nr][nc] == 1)
            trv(nr, nc, n, m, dist + 1, grid, vis);
    }
    vis[r][c] = 0;
}
int shortestPath(vector<vector<int>> &field)
{
    mn = INT_MAX;
    int n = field.size(), m = field[0].size();
    vector<vector<int>> grid = field;
    vector<vector<int>> vis(n, vector<int>(m, 0));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (field[i][j] == 0)
            {
                for (int d = 0; d < 4; d++)
                {
                    int nr = i + dir[d], nc = j + dir[d + 1];
                    if (nr < n && nc < m && nr > -1 && nc > -1)
                        grid[nr][nc] = 0;
                }
            }

    for (int i = 0; i < n; i++)
        trv(i, 0, n, m, 0, grid, vis);

    return mn == INT_MAX ? -1 : mn;
}

// Largest number in K swaps
```

```
void findMax(string str, int k, int i, string &ans)
{
    if (i == str.size() - 1 || k == 0)
    {
        if (ans < str)
        {
            ans = str;
        }
        return;
    }
    for (int j = i + 1; j < str.size(); j++)
    {
        if (str[j] > str[i])
        {
            swap(str[i], str[j]);
            findMax(str, k - 1, i + 1, ans);
            swap(str[i], str[j]);
        }
    }
    findMax(str, k, i + 1, ans);
}
string findMaximumNum(string str, int k)
{
    string ans = str;
    findMax(str, k, 0, ans);
    return ans;
}
// Largest number in K swaps
void solve(string str, string &ans, int ind, int k, int n)
{
    if (k == 0)
        return;
    char mxchar = str[ind];
    for (int i = ind + 1; i < n; i++)
    {
        if (mxchar < str[i])
            mxchar = str[i];
    }
    if (mxchar != str[ind])
        k--;
    for (int i = n - 1; i >= ind; i--)
    {
        if (str[i] == mxchar)
        {
            swap(str[ind], str[i]);
            if (ans < str)
                ans = str;
            solve(str, ans, ind + 1, k, n);
            swap(str[ind], str[i]);
        }
    }
    return;
}
string find
```

```
Num(string str, int k)
{
    string ans = str;
    int n = str.length();
    solve(str, ans, 0, k, n);
    return ans;
}

// Find if there is a path of more than k length from a source
bool sove(int src, int k, vector<pair<int, int>> g[],
          vector<bool> &vis)
{
    vis[src] = 1;
    if (k <= 0)
        return 1;
    for (auto x : g[src])
    {
        int u = x.first;
        int w = x.second;
        if (vis[u] == 1)
            continue;
        if (w >= k)
            return 1;
        if (solve(u, k - u, g, vis))
            return 1;
    }
    vis[src] = 0;
    return 0;
}

// Longest Possible Route in a Matrix with Hurdles
Pair f(int mat[R][C], int i, int j, int x, int y,
       bool visited[R][C])
{
    if (i == x && j == y)
    {
        Pair p = {true, 0};
        return p;
    }
    if (i < 0 || i >= R || j < 0 || j >= C || mat[i][j] == 0 || visited[i][j])
    {
        Pair p = {false, INT_MAX};
        return p;
    }
    visited[i][j] = true;
    int res = INT_MIN;
    res = max({f(mat, i, j - 1, x, y, visited).value, f(mat, i, j + 1, x, y, visited).value,
             f(mat, i - 1, j, x, y, visited).value, f(mat, i + 1, j, x, y, visited).value})
    visited[i][j] = false;

    if (res != INT_MIN)
    {
        Pair p = {true, 1 + res};
        return p;
```

```
    }
    else
    {
        Pair p = {false, INT_MAX};
        return p;
    }
}
```

```
// Print all possible paths from top left to bottom right of a mXn matrix
bool issafe(int r,int c,vector<vector<int>>& visited,int n,int m) {
  return (r < n and c <m and visited[r] !=-1 );  // return true if all values satisfied else false
}


void FindPaths(vector<vector<int>> &grid,int r,int c, int n,int m,vector<int> &ans) {
    if(r == n-1 and c == m-1) {
        ans.push_back(grid[r]);
        display(ans);  // function to display the path stored in ans vector
        ans.pop_back(); // pop back because we need to backtrack to explore more path
        return ;
    }

    int ch = grid[r];
    ans.push_back(ch);

    grid[r] = -1;
    if(issafe(r+1,c,grid,n,m)) FindPaths(grid,r+1,c,n,m,ans);
    if(issafe(r,c+1,grid,n,m)) FindPaths(grid,r,c+1,n,m,ans);
    grid[r] = ch;

    ans.pop_back();
    return ;
}
```

```
// Maximum size square sub-matrix with all 1s
int maxSquare(int n, int m, vector<vector<int>> mat)
{
    for (int i = 1; i < n; i++)
    {
        for (int j = 1; j < m; j++)
            if (mat[i][j])
                mat[i][j] += min({mat[i - 1][j], mat[i][j - 1], mat[i - 1][j - 1]});
    }
    int h = 0;
    for (auto i : mat)
        for (auto j : i)
            h = max(h, j);
    return h;
}
```

```
// Maximal Rectangle
```

```
int largestRect(int arr[], int n)
{
 stack<int> s;
 int res = 0;
 for (int i = 0; i < n; i++)
 {
  while (!s.empty() && arr[s.top()] >= arr[i])
  {
   int temp = s.top();
   s.pop();
   int val = arr[temp] * (s.empty() ? i : i - s.top() - 1);
   res = max(res, val);
  }
  s.push(i);
 }
 while (!s.empty())
 {
  int temp = s.top();
  s.pop();
  int val = arr[temp] * (s.empty() ? n : n - s.top() - 1);
  res = max(res, val);
 }
 return res;
}

int maxArea(int M[MAX][MAX], int n, int m)
{
 int arr[m] = {0};
 int res = 0;
 for (int i = 0; i < n; i++)
 {
  for (int j = 0; j < m; j++)
  {
   if (M[i][j] == 1)
    arr[j] += M[i][j];
   else
    arr[j] = 0;
  }
  res = max(largestRect(arr, m), res);
 }
 return res;
}

// Maximum sum rectangle in a 2D matrix
    int maximumSumRectangle(int R, int C, vector<vector<int>> M) {
        // code here
               int ans = 0;
        int ans1 = -1000;
        for(int i = 0 ; i < R ; i ++){
            for(int j = 0 ;  j < C ; j ++){
                ans1 = max(ans1,M[i][j]);
            }
        }
        for(int i = 0 ; i < C ; i ++){
            vector< int > dp(R,0);
```

```
        for(int j = i ; j < C ; j++){
            for(int k = 0 ; k < R; k ++){
                dp[k] += M[k][j];
            }
            int res = 0;
            for(int k = 0 ; k < R ; k ++){
                res += dp[k];
                if(res < 0) res  = 0;
                ans = max(ans,res);
            }
        }
    }
    if(ans == 0) return ans1;
    return ans;
}
```

```
// 1654. Minimum Jumps to Reach Home
int minimumJumps(vector<int>& forbidden, int a, int b, int x) {
    unordered_map<int,int> v;
    queue<pair<int,int>> q;
    for(auto i:forbidden) v[i]=true;
    q.push({0,0}) ;
    int ans = 0;
    while(!q.empty()){
        int size = q.size() ;
        while(size--){
            auto curr = q.front() ;
            q.pop() ;
            int num = curr.first;
            if(num == x){
                return ans;
            }

            if(v[num] == true){
                continue;
            }
            v[num]=true;
            if(curr.second == 0 && num-b>=0) {
                q.push({(num-b),1});
            }
            if(num <= 2000+b){
                q.push({(num+a),0});
            }
        }
        ans++;
    }
    return -1;
}
```

```
// Largest rectangular sub-matrix whose sum is 0
// Largest area rectangular sub-matrix with equal number of 1's and 0's
#define MAX_ROW 10
#define MAX_COL 10
```

```
bool subArrWithSumZero(int arr[], int &start,  int &end, int n)
{
   // to store cumulative sum
   int sum[n];

   // Initialize all elements of sum[] to 0
   memset(sum, 0, sizeof(sum));

   // map to store the indexes of sum
   unordered_map<int, int> um;

   // build up the cumulative sum[] array
   sum[0] = arr[0];
   for (int i=1; i<n; i++)
      sum[i] = sum[i-1] + arr[i];

   // to store the maximum length subarray
   // with sum equal to 0
   int maxLen = 0;

   // traverse to the sum[] array
   for (int i=0; i<n; i++)
   {
      // if true, then there is a subarray
      // with sum equal to 0 from the
      // beginning up to index 'i'
      if (sum[i] == 0)
      {
         // update the required variables
         start = 0;
         end = i;
         maxLen = (i+1);
      }

      // else if true, then sum[i] has not
      // seen before in 'um'
      else if (um.find(sum[i]) == um.end())
         um[sum[i]] = i;

      // sum[i] has been seen before in the
      // unordered_map 'um'
      else
      {
         // if previous subarray length is smaller
         // than the current subarray length, then
         // update the required variables
         if (maxLen < (i-um[sum[i]]))
         {
            maxLen = (i-um[sum[i]]);
            start = um[sum[i]] + 1;
            end = i;
         }
      }
   }
```

```
    // if true, then there is no
    // subarray with sum equal to 0
    if (maxLen == 0)
        return false;

    // else return true
    return true;
}
void maxAreaRectWithSumZero(int mat[MAX_ROW][MAX_COL],
                                int row, int col)
{
    // to store intermediate values
    int temp[row], startRow, endRow;

    // to store the final outputs
    int finalLeft, finalRight, finalTop, finalBottom;
    finalLeft = finalRight = finalTop = finalBottom = -1;
    int maxArea = 0;

    // Set the left column
    for (int left = 0; left < col; left++)
    {
        // Initialize all elements of temp as 0
        memset(temp, 0, sizeof(temp));
        for (int right = left; right < col; right++)
        {
            for (int i=0; i<row; i++)
                temp[i] += mat[i][right] ? 1 : -1;
            if (subArrWithSumZero(temp, startRow, endRow, row))
            {
                int area = (right - left + 1) *
                                    (endRow - startRow + 1);

                // Compare current 'area' with previous area
                // and accordingly update final values
                if (maxArea < area)
                {
                    finalTop = startRow;
                    finalBottom = endRow;
                    finalLeft = left;
                    finalRight = right;
                    maxArea = area;
                }
            }
        }
    }

    // if true then there is no rectangular submatrix
    // with equal number of 1's and 0's
    if (maxArea == 0)
        cout << "No such rectangular submatrix exists:";
    else
    {
        cout << "(Top, Left): "
             << "(" << finalTop << ", " << finalLeft
```

```
            << ")" << endl;

        cout << "(Bottom, Right): "
            << "(" << finalBottom << ", " << finalRight
            << ")" << endl;

        cout << "Area: " << maxArea << " sq.units";
    }
}
```

```
// Boolean Parenthesization Problem
public static int solution(String str1, String str2) {
  int n = str1.length();
  int[][] t = new int[n][n];
  int[][] f = new int[n][n];
  for (int gap = 0; gap < n; gap++) {
   int si = 0, ei = gap;
   while (ei < n) {
    if (gap == 0) {
     t[si][ei] = str1.charAt(si) == 'T' ? 1 : 0;
     f[si][ei] = str1.charAt(si) == 'F' ? 1 : 0;
    } else {
     for (int cp = si; cp < ei; cp++) {
      char sign = str2.charAt(cp);
      if (sign == '&') {
       t[si][ei] += t[si][cp] * t[cp + 1][ei];
       f[si][ei] += ((t[si][cp] * f[cp + 1][ei]) + (f[si][cp] * t[cp + 1][ei])
         + (f[si][cp] * f[cp + 1][ei]));
      }
      if (sign == '|') {
       t[si][ei] += ((t[si][cp] * t[cp + 1][ei]) + (t[si][cp] * f[cp + 1][ei])
         + (f[si][cp] * t[cp + 1][ei]));
       f[si][ei] += ((f[si][cp]) * (f[cp + 1][ei]));
      }
      if (sign == '^') {
       t[si][ei] += ((t[si][cp] * f[cp + 1][ei]) + (f[si][cp] * t[cp + 1][ei]));
       f[si][ei] += ((t[si][cp] * t[cp + 1][ei]) + (f[si][cp] * f[cp + 1][ei]));
      }
     }
    }
    si++;
    ei++;
   }
  }
  return (t[0][t[0].length - 1]);
 }
```

```
// Count Bits - dp
    vector<int> countBits(int num) {
        vector<int> res(num);
        res.push_back(0);  // for num=0
        if(num==0) return res;
```

```
        for(int i=1;i<=num;i++){
            if(i%2==0){
                res[i]=res[i/2];
            } else {
                res[i]=res[i-1]+1;
            }
        }
        return res;
    }

// Count total set bits in all numbers from 1 to n
int countSetBits(int n)
{
 if(!n) return 0;
 int bits = log2(n+1), x=1<<bits;
 bits*= (x>>1);
 if(n<x) return bits;
 return bits+ (n-x+1) + countSetBits(n-x);
}


// 2. Add Two Numbers
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    int carry = 0, first, second;
    ListNode *head = new ListNode(0), *tail = head;
    while (l1 || l2 || carry) {
        if (l1) {
            first = l1->val;
            l1 = l1->next;
        }
        else
            first = 0;

        if (l2) {
            second = l2->val;
            l2 = l2->next;
        }
        else
            second = 0;

        int temp = first+second+carry;
        tail->next = new ListNode(temp%10);
        tail = tail->next;
        carry = temp/10;

    }
    return head->next;
}


// 6. Zigzag Conversion
    string convert(string s, int numRows) {
        if (numRows == 1) return s;

        vector<string> v(min(numRows, int(s.size())), "");
```

```
        int direction = -1, idx = 0;

        for (int i = 0; i < s.size(); i++) {
            v[idx] += s[i];
            idx += direction == -1 ? 1 : -1;
            if (idx == 0 || idx == numRows - 1) direction = -direction;
        }

        string res = "";
        for (auto a : v) res += a;

        return res;
    }

// 8. String to Integer (atoi)
    int myAtoi(string str) {
        int res=0;
        bool is_positive = true;
        int i=0;

        while (str[i] == ' ') // count leading spaces
            i++;

        if (str[i] == '-') // check sign
        {
            is_positive = false;
            i++;
        }
        else if (str[i] == '+')
        {
            is_positive = true;
            i++;
        }
        str.erase(0,i); // remove leading spaces and sign

        for (int i=0;i<str.size();i++)
        {
            if (isdigit(str[i]))
            {
                int value = str[i] - '0';

                if ((res > (INT_MAX - value)/10) && (is_positive))
                    return INT_MAX;
                if ((res > (INT_MAX - value)/10) && (!is_positive))
                    return INT_MIN;

                (res*=10) += value;
            }
            else // no more digits
                break;
        }

        if (!is_positive)
            return -res;
        return res;
```

```
//        int ans=0;
//        for(int i =0; i < s.size(); i++){
//            if(s[i] == 45 && i==0)
//                    ans=-1;
//            else if(s[i] == 45 && i==0)
//                ans*=-1;
//            if(int(s[i])>=48 && int(s[i])<=57 )
//             {
//                int a= s[i]-48;
//                ans+= a;
//                ans*=10;
//            }

//        }
//        return ans/10;

//        long result = 0;
//        int indicator = 1;
//        for(int i = 0; i<str.size();i++)
//        {
//            if(i==0)
//            i = str.find_first_not_of(' ');

//            if(str[i] == '-' || str[i] == '+')    //step2
//                indicator = (str[i] == '-')? -1 : 1;

//            if('0'<= str[i] && str[i] <= '9')     //step3
//            {
//                result = result*10 + (str[i]-'0');
//                if(result*indicator >= INT_MAX) return INT_MAX;
//                if(result*indicator <= INT_MIN) return INT_MIN;
//            }
//        }
//        return result*indicator;
//        return 0;
    }


// 16. 3Sum Closest
    int threeSumClosest(vector<int>& nums, int target) {
        int n = nums.size(), j, k, res = nums[0] + nums[1] + nums[2], curr;
        sort(nums.begin(), nums.end());

        for (int i = 0; i < n; i++) {
            j = i + 1, k = n - 1;

            while (j < k){
                curr = nums[i] + nums[j] + nums[k];
                if (abs(curr - target) < abs(res - target))
                    res = curr;

                if (curr < target) j++;
                else k--;
            }
```

```cpp
        }
        return res;
    }

// 22. Generate Parentheses
vector <string> valid;
void generate( string &s, int open , int close)
{
    if(open==0 && close==0){
        valid.push_back(s);
        return;
    }

    if(open > 0 ){
        s.push_back('(');
        generate(s, open-1, close);
        s.pop_back();
    }

    if(close > 0){
        if(open < close){
            s.push_back(')');
            generate(s, open, close-1);
            s.pop_back();
        }
    }
}
vector<string> generateParenthesis(int n) {
    string s;
    generate(s,n,n);
    return valid;
}


// 43. Multiply Strings
    string multiply(string num1, string num2) {
        if (num1 == "0" || num2 == "0")
            return "0";

        vector<int> res(num1.size()+num2.size(), 0);

        for (int i = num1.size()-1; i >= 0; i--){
            for (int j = num2.size()-1; j >= 0; j--){
                res[i + j + 1] += (num1[i]-'0') * (num2[j]-'0');
                res[i + j] += res[i + j + 1] / 10;
                res[i + j + 1] %= 10;
            }
        }

        int i = 0;
        string ans = "";
        while (res[i] == 0)    i++;
        while (i < res.size())
            ans += to_string(res[i++]);
        return ans;
```

```
    }


// 50. Pow(x, n)
    double myPow(double x, int n) {
        if(n==0) return 1;
        double t = myPow(x,n/2);
        if(n%2) return n<0 ? 1/x*t*t : x*t*t;
        else return t*t;
    }


// 64. Minimum Path Sum
    int minCost(vector<vector<int>> &cost,int m, int n,vector<vector<int>>& memo) {
      if (n <0 || m <0)
         return INT_MAX;
      else if(m==0 && n==0)
         return cost[m][n];
      if(memo[m][n]!=-1)
          return memo[m][n];
      return memo[m][n] = cost[m][n]+min(minCost(cost,m-1,n,memo),minCost(cost,m,n-1,memo));
    }

    int minPathSum(vector<vector<int>>& grid) {
        int m=grid.size(),n=grid[0].size();
        vector<vector<int>> memo(m,vector<int>(n, -1));
        return minCost(grid,m-1,n-1,memo);
    }
    int minPathSum(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();
        vector<vector<int> > sum(m, vector<int>(n, grid[0][0]));
        for (int i = 1; i < m; i++)
            sum[i][0] = sum[i - 1][0] + grid[i][0];
        for (int j = 1; j < n; j++)
            sum[0][j] = sum[0][j - 1] + grid[0][j];
        for (int i = 1; i < m; i++)
            for (int j = 1; j < n; j++)
                sum[i][j]  = min(sum[i - 1][j], sum[i][j - 1]) + grid[i][j];
        return sum[m - 1][n - 1];
    }

// 71. Simplify Path
    string simplifyPath(string path) {
        string res, tmp;
        vector<string> stk;
        stringstream ss(path);
        while(getline(ss,tmp,'/')) {
            if (tmp == "" or tmp == ".") continue;
            if (tmp == ".." and !stk.empty()) stk.pop_back();
            else if (tmp != "..") stk.push_back(tmp);
        }
        for(auto str : stk) res += "/"+str;
        return res.empty() ? "/" : res;
    }
```

// 89. Gray Code
```
    vector<int> grayCode(int n) {
        vector<int> v;
        long long int p=pow(2,n);
        for(int i=0; i<p; i++){
            v.push_back(i^(i/2));
        }
        return v;
    }
```

// 91. Decode Ways
```
    int numDecodings(string s) {
        int n = s.size();
        vector<int> dp(n+1);
        dp[n] = 1;
        for(int i=n-1;i>=0;i--) {
            if(s[i]=='0') dp[i]=0;
            else {
                dp[i] = dp[i+1];
                if(i<n-1 && (s[i]=='1'||s[i]=='2'&&s[i+1]<'7')) dp[i]+=dp[i+2];
            }
        }
        return s.empty()? 0 : dp[0];
    }
```

// 95. Unique Binary Search Trees II
```
    vector<TreeNode*> rec(int start, int end) {
        vector<TreeNode*> res;
        if (start > end) return {NULL};

        if (start == end) return {new TreeNode(start)};

        for (int i = start; i <= end; i++) {
            vector<TreeNode*> left = rec(start, i-1), right = rec(i+1, end);

            for (auto l : left)
                for (auto r : right)
                    res.push_back(new TreeNode(i, l, r));
        }
        return res;
    }

    vector<TreeNode*> generateTrees(int n){
        vector<TreeNode*> res = rec(1, n);
        return res;
    }
```

// 99. Recover Binary Search Tree

// 113. Path Sum II

```
    vector<vector<int>> pathSum(TreeNode* root, int sum) {
        vector<vector<int> > paths;
        vector<int> path;
        findPaths(root, sum, path, paths);
        return paths;
    }
    void findPaths(TreeNode* node, int sum, vector<int>& path, vector<vector<int> >& paths) {
        if (!node)   return;
        path.push_back(node->val);

        if (!(node -> left) && !(node -> right) && sum == node -> val)
            paths.push_back(path);

        findPaths(node->left, sum - node->val, path, paths);
        findPaths(node->right, sum - node->val, path, paths);

        path.pop_back();
    }

// 120. Triangle
    int minimumTotal(vector<vector<int>>& t) {
        for(int level = 1; level < size(t); level++)
            for(int i = 0; i <= level; i++)
                t[level][i] +=  min(t[level - 1][ min(i, level - 1)], t[level - 1][max(i - 1, 0)]);

        return *min_element(begin(t.back()), end(t.back()));
    }

// 129. Sum Root to Leaf Numbers
    int sumNumbers(TreeNode* root) {
        return dfs(root, 0);
    }

    int dfs(TreeNode* root, int cur) {
        if(!root) return 0;
        cur = cur * 10 + root -> val;
        if(!root -> left && !root -> right)
            return cur;
        return dfs(root -> left, cur) + dfs(root -> right, cur);
    }

// 150. Evaluate Reverse Polish Notation
    int evalRPN(vector<string>& tokens) {
        stack<int> s;
        for(auto& t : tokens)
            if(t == "+" || t == "-" || t == "*" || t == "/") {
                int op1 = s.top(); s.pop();
                int op2 = s.top(); s.pop();
                if(t == "+") op1 = op2 + op1;
                if(t == "-") op1 = op2 - op1;
                if(t == "/") op1 = op2 / op1;
                if(t == "*") op1 = op2 * op1;
                s.push(op1);
            }
            else
```

```
            s.push(stoi(t));    // stoi - converts from string to int
        return s.top();
    }
```

```
// 275. H-Index II
    int hIndex(vector<int>& citations) {
              int left=0, len = citations.size(), right= len-1,  mid;
        while(left<=right)
        {
            mid=(left+right)>>1;
            if(citations[mid]== (len-mid)) return citations[mid];
            else if(citations[mid] > (len-mid)) right = mid - 1;
            else left = mid + 1;
        }
        return len - (right+1);
    }
```

```
// 299. Bulls and Cows
    string getHint(string secret, string guess) {
            int aCnt = 0;
        int bCnt = 0;
        vector<int> sVec(10, 0); // 0 ~ 9 for secret
        vector<int> gVec(10, 0); // 0 ~ 9 for guess
        if (secret.size() != guess.size() || secret.empty()) { return "0A0B"; }
        for (int i = 0; i < secret.size(); ++i) {
            char c1 = secret[i]; char c2 = guess[i];
            if (c1 == c2) {
                ++aCnt;
            } else {
                ++sVec[c1-'0'];
                ++gVec[c2-'0'];
            }
        }
        // count b
        for (int i = 0; i < sVec.size(); ++i) {
            bCnt += min(sVec[i], gVec[i]);
        }
        return to_string(aCnt) + 'A' + to_string(bCnt) + 'B';
    }
```

```
// 306. Additive Number
 bool getFibo(string &s,int i,long long a,long long b,int n) {
  if(i==s.length()) return n>2;

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

   if(num>pow(10,17)) break;  //the max length of string is 35, which cannot fit 2 strings of len 18
   num= num*10+s[x]-'0';                    // so we break the loop

   bool chk=false;
   if(n<2) chk=getFibo(s,x+1,b,num,n+1);
   else if(a+b==num) chk= getFibo(s,x+1,b,num,n+1);
   if(chk) return true;
```

```
  if(num==0) break;
 }
 return false;
}

bool isAdditiveNumber(string num) {
 return getFibo(num,0,0,0,0);
}

// 316. Remove Duplicate Letters
    string removeDuplicateLetters(string s) {
        vector<int> lastIndex(26, 0);
        for (int i = 0; i < s.length(); i++){
            lastIndex[s[i] - 'a'] = i; // track the lastIndex of character presence
        }

        vector<bool> seen(26, false); // keep track seen
        stack<char> st;

        for (int i = 0; i < s.size(); i++) {
            int curr = s[i] - 'a';
            if (seen[curr]) continue; // if seen continue as we need to pick one char only
            while(st.size() > 0 && st.top() > s[i] && i < lastIndex[st.top() - 'a']){
                seen[st.top() - 'a'] = false; // pop out and mark unseen
                st.pop();
            }
            st.push(s[i]); // add into stack
            seen[curr] = true; // mark seen
        }

        string ans = "";
        while (st.size() > 0){
            ans += st.top();
            st.pop();
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }

// 318. Maximum Product of Word Lengths
    int maxProduct(vector<string>& words) {
        int n = size(words), ans = 0;
        vector<bitset<26> > chars(n);

        for(int i = 0; i < n; i++) {
            for(auto& ch : words[i])
                chars[i][ch - 'a'] = 1;

            for(int j = 0; j < i; j++)
                if(!checkCommon(chars[i], chars[j]))
                    ans = max(ans, int(size(words[i]) * size(words[j])));
        }
        return ans;
    }
```

```
    bool checkCommon(bitset<26>& a, bitset<26>& b) {
        for(int i = 0; i < 26; i++) if (a[i] && b[i]) return true;
        return false;
    }


// 319. Bulb Switcher
int bulbSwitch(int n) {
    int counts = 0;
    for (int i=1; i*i<=n; ++i)
        ++ counts;
    return counts;
}

// 322. Coin Change
    int ans= INT_MAX, ct = 0;
    int coinChange(vector<int>& coins, int amount) {
        int Max = amount + 1;
        vector<int> dp(amount + 1, Max);
        dp[0] = 0;
        for (int i = 1; i <= amount; i++) {
            for (int j = 0; j < coins.size(); j++) {
                if (coins[j] <= i) {
                    dp[i] = min(dp[i], dp[i - coins[j]] + 1);
                }
            }
        }
        return dp[amount] > amount ? -1 : dp[amount];
    }


// 324. Wiggle Sort II

// 334. Increasing Triplet Subsequence
    bool increasingTriplet(vector<int>& nums) {
        int c1 = INT_MAX, c2 = INT_MAX;
        for (int x : nums){
            if (x <= c1)
                c1 = x;
            else if (x <= c2)
                c2 = x;
            else
                return true;
        }
        return false;




        // int n = nums.size();
        // if(n<3) return 0;
        // for(int i = 0 ;i< n ; i++)
        //     for(int j = i+1 ; j< n; j++)
        //         for(int  k =j +1 ; k< n ; k++)
        //             if(nums[i] < nums[j])
        //                 if(nums[j] < nums[k])
```

```
    //                    return 1;
    // return 0;
}


// 343. Integer Break
// Input: n = 10
// Output: 36
// Explanation: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36.
    int dp[58 + 1][57 + 1];

    int recursion(int n, int cur)
    {
        if (n == 0 || cur == 0)
            return 1;
        if (dp[n][cur] != -1)
            return dp[n][cur];
        if (cur > n)
            return dp[n][cur] = recursion(n - 0, cur - 1);
        else
            return dp[n][cur] = max(recursion(n - 0, cur - 1), cur * recursion(n - cur, cur));
    }

    int integerBreak(int n)
    {
        memset(dp, -1, sizeof(dp));
        return recursion(n, n - 1);
    }


//  Count Numbers with Unique Digits
    int countNumbersWithUniqueDigits(int n) {
        if(!n) return 1;
        int ans=10,start=9, current=9;
        while(n-->1 && start){
            current *=(start--);
            ans += current;
        }
        return ans;
    }


// 654. Maximum Binary Tree
    TreeNode* constructMaximumBinaryTree(vector<int>& nums) {
        vector<TreeNode*> stk;
        for (int i = 0; i < nums.size(); ++i){
            TreeNode* cur = new TreeNode(nums[i]);
            while (!stk.empty() && stk.back()->val < nums[i]){
                cur->left = stk.back();
                stk.pop_back();
            }
            if (!stk.empty())
                stk.back()->right = cur;
            stk.push_back(cur);
        }
```

```
        return stk.front();
    }


// 153. Find Minimum in Rotated Sorted Array
    int findMin(vector<int>& nums) {
        int left = 0,  right = nums.size() - 1;
        while(left < right){
            if(nums[left] < nums[right])
                return nums[left];

            int mid = (left + right)/2;
            if(nums[mid] > nums[right])
                left = mid + 1;
            else
                right = mid;
        }
        return nums[left];
    }

// 162. Find Peak Element
    int findPeakElement(vector<int>& nums) {
        // way 1:traverse whole array o(n), o(1)
        // way 2: sliding window o(n), o(1)
        // way 3: priority queue  o(n), o(2)
        //way 4: binary
        int lo = 0 , mid, hi = nums.size()-1;

        while(lo<=hi){
            mid = (lo+hi)/2;
            if(lo == hi)
                break;
            if(nums[mid] > nums[mid+1])
                hi = mid;
            else lo = mid+1;
        }
        return lo;
    }

// 179. Largest Number
      string largestNumber(vector<int>& nums) {
        vector<string> container;

        for(int i : nums)
            container.push_back(to_string(i));

        sort(container.begin(),container.end(),compare);

        string result;

        for(int i=0;i<container.size();i++)
            result+=container[i];

        return result[0]=='0'? "0" : result;
    }
```

// 187. Repeated DNA Sequences

```
    vector<string> findRepeatedDnaSequences(string s) {
        unordered_map<string, int> counter;
        vector<string> res;
        if (s.size() < 10) return res;
        for (int i=0; i<s.size()-9; i++)
            counter[s.substr(i, 10)]++;
        for (auto a:counter)
            if (a.second > 1)
                res.push_back(a.first);
        return res;
    }
```

// 201. Bitwise AND of Numbers Range

```
    int rangeBitwiseAnd(int m, int n) {
        if ((m == 0) || (n == 0)) return 0;
        if ((int)log2(m) != (int)log2(n)) return 0;
        int res = m;
        for (long i = m; i <= n; i++)
            res &= i;
        return res;
    }
```

// 204. Count Primes

```
    int countPrimes(int n) {
        if (n==0 || n==1) return 0;
        vector<bool> prime(n, true);
        prime[0] = false, prime[1] = false;
        for (int i = 0; i < sqrt(n); ++i) {
            if (prime[i]) {
                for (int j = i*i; j < n; j += i) {
                    prime[j] = false;
                }
            }
        }
        return count(prime.begin(), prime.end(), true);
    }
```

// 209. Minimum Size Subarray Sum

```
    int minSubArrayLen(int tgt, vector<int>& a) {
        int l=0, r=0, t=0,ans=INT_MAX, n=a.size();
        while(r<n){
            t+=a[r++];
            while(t>=tgt)
                t-=a[l++], ans = min(ans, r-l+1);
        }
        return ans==INT_MAX?0:ans;
    }
```

// 216. Combination Sum III

```
    vector<vector<int>> ans;
    void f(vector<int>& cur, int cnum, int k, int n) {
        if(n < 0 || cur.size() > k) return;
```

```
        if(n == 0 && cur.size() == k) {
            ans.push_back(cur);
            return;
        }

        for(int i=cnum; i<=9; ++i) {
            cur.push_back(i);
            f(cur, i+1, k, n-i);
            cur.pop_back();
        }
    }
    vector<vector<int>> combinationSum3(int k, int n) {
        vector<int> cur;
        f(cur, 1, k, n);
        return ans;
    }


// 220. Contains Duplicate III
// abs(nums[i] - nums[j]) <= t and abs(i - j) <= k.
    bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t) {
         multimap <int,int> mp;
        for(int i=0;i<nums.size();i++) mp.insert(pair< int, int >(nums[i], i));

        multimap <int,int>::iterator it, itnext;

        for(it=mp.begin();it!=mp.end();it++){
            itnext = it;
            while(true){
                itnext++;
                if(itnext == mp.end()) break;
                long long a = (*it).first;
                long long b = (*itnext).first;
                if( b - a <= t)  {if(abs((*it).second - (*itnext).second) <= k) return true;}
                else break;
            }
        }
        return false;
    }

// 221. Maximal Square
    int maximalSquare(vector<vector<char>>& matrix) {
        int m = matrix.size();
        if(m==0) return 0;
        int n = matrix[0].size();
        vector<vector<int>> dp(m+1, vector<int>(n+1, 0));
        int ret = 0;

        for(int i = 1; i <= m; i ++){
            for(int j = 1; j <= n; j ++){
                if(matrix[i-1][j-1] == '0'){
                    dp[i][j] = 0;
                }else{
                    dp[i][j] = 1 + min({dp[i-1][j], dp[i][j-1], dp[i-1][j-1]});
                }
```

```
            ret = max(ret, dp[i][j]);
        }
    }
    return ret*ret;
}
```

// 222. Count Complete Tree Nodes

```
int countNodes(TreeNode* root) {
    if(!root)
        return 0;
    int hl=0, hr=0;
    TreeNode *l=root, *r=root;
    while(l) {
        hl++;
        l=l->left;
    }
    while(r) {
        hr++;
        r=r->right;
    }

    if(hl==hr)
        return pow(2,hl)-1;
    return 1 + countNodes(root->left) + countNodes(root->right);
}
```

// 223. Rectangle Area

```
int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int bx2, int by2) {
    int area1 = (ax2-ax1) * (ay2-ay1);
    int area2 = (bx2-bx1) * (by2-by1);
    int comarea = 0;

    int left = max(ax1, bx1);
    int right = min(ax2, bx2);
    int top = min(ay2, by2);
    int bottom = max(ay1, by1);

    if(left < right && top > bottom)
        comarea = (right-left) *(top-bottom);

    return area1 + area2 - comarea;
}
```

// 227. Basic Calculator II

```
int calculate(string s) {
    vector<int> vt;

    int num=0;
    char sign='+';
    for(int i=0;i<=s.length();i++){
        if(s[i]>='0'&&s[i]<='9'){
            num=num*10+(s[i]-'0');
        }
        else if(s[i]=='+'||s[i]=='-'||s[i]=='/'||s[i]=='*'||i==s.length()){
```

```
            if(sign=='+'){
                vt.push_back(num);
                num=0;
            }else if(sign=='-'){
                vt.push_back(-1*num);
                num=0;
            }else if(sign=='/'){
                vt[vt.size()-1]=vt.back()/num;
                num=0;
            }else if(sign=='*'){
                vt[vt.size()-1]=vt.back()*num;
                num=0;
            }

            if(i!=s.length()){
             sign=s[i];
            }
        }
    }

    int result=0;
    for(int i=0;i<vt.size();i++){
        result+=vt[i];
    }

    return result;
}
```

```
// 241. Different Ways to Add Parentheses
    int perform(int x, int y, char op) {
        if(op == '+') return x + y;
        if(op == '-') return x - y;
        if(op == '*') return x * y;
        return 0;
    }

    vector<int> diffWaysToCompute(string exp) {

        vector<int> results;
        bool isNumber = 1;

        for(int i = 0; i < exp.length(); i++) {
            // check if current character is an operator
            if(!isdigit(exp[i])) {

                // if current character is not a digit then
                // exp is not purely a number
                isNumber = 0;

                // list of first operands
                vector<int> left = diffWaysToCompute(exp.substr(0, i));

                // list of second operands
```

```
            vector<int> right = diffWaysToCompute(exp.substr(i + 1));

            // performing operations
            for(int x : left) {
                for(int y : right) {
                    int val = perform(x, y, exp[i]);
                    results.push_back(val);
                }
            }

        }
    }

    if(isNumber == 1) results.push_back(stoi(exp));
    return results;
}
```

```
// 371. Sum of Two Integers
    int getSum(int a, int b) {
        while(b) {
            unsigned c = a&b;
            a ^= b;
            b = c << 1;
        }
        return a;
    }

// 372. Super Pow
// Input: a = 2, b = [1,0]
// Output: 1024
#define ll long long
const ll mod = 1337;
ll powMod(ll a, ll b) {
    ll res = 1;
    while (b) {
        if (b & 1) res *= a, res %= mod;
        a *= a;
        a %= mod;
        b >>= 1;
    }
    return res;
}

int superPow(int a, vector<int>& b) {
    ll res = 1;
    for (int i = 0; i < b.size(); ++ i) {
        res = powMod(res, 10) * powMod(a, b[i]);
    }
    return static_cast<int>(res%mod);
}

// 373. Find K Pairs with Smallest Sums
    vector<vector<int>> kSmallestPairs(vector<int>& nums1, vector<int>& nums2, int k) {
```

```
    priority_queue<pair<int,pair<int,int>>> pq;
        for(int i=0;i<nums1.size();i++) {
            for(int j=0;j<nums2.size();j++) {
                int sum=nums1[i]+nums2[j];
                if (pq.size()<k) {
                    pq.push({sum,{nums1[i],nums2[j]}});
                }
                else if (sum<pq.top().first) {
                    pq.pop();
                    pq.push({sum,{nums1[i],nums2[j]}});
                } else {
                    break;
                }
            }
        }
        vector<vector<int>> ans;
        while(!pq.empty()) {
            ans.push_back({pq.top().second.first, pq.top().second.second});
            pq.pop();
        }
        reverse(ans.begin(),ans.end());
        return ans;
    }

// 376. Wiggle Subsequence
    int wiggleMaxLength(vector<int>& nums) {
        int size=nums.size(), f=1, d=1;
        for(int i=1; i<size; ++i){
              if(nums[i]>nums[i-1]) f=d+1;
            else if(nums[i]<nums[i-1]) d=f+1;
        }
        return min(size, max(f, d));
    }

// 377. Combination Sum IV
    int combinationSum4(vector<int>& nums, int target) {
        long dp[target + 1];
        dp[0] = 1;
        for (int i = 1; i <= target; i++) {
            dp[i] = 0;
            for (int n: nums) {
                if (i >= n) dp[i] += dp[i - n];
                if (dp[i] > INT_MAX) break;
            }
        }
        return dp[target];
    }

class Solution { // Coin Change 2
    public int change(int amount, int[] coins) {
        int[] dp = new int[amount + 1];
        dp[0] = 1;
        for (int coin : coins) {
            for (int i = 1; i <= amount; i++) {
                if (i >= coin) {
```

```
                dp[i] += dp[i - coin];
            }
        }
    }
    return dp[amount];
  }
}
class Solution { // this problem
  public int combinationSum4(int[] nums, int target) {
    int[] dp = new int[target + 1];
    dp[0] = 1;
    for (int i = 1; i <= target; i++) {
      for (int n : nums) {
        if (i >= n) {
          dp[i] += dp[i - n];
        }
      }
    }
    return dp[target];
  }
}
```

```
// 378. Kth Smallest Element in a Sorted Matrix
  int kthSmallest(vector<vector<int>>& matrix, int k) {
    int n = matrix.size();
    int lo = matrix[0][0], hi = matrix[n-1][n-1] + 1, mid, count, tmp;

    while (lo < hi) {
      mid = lo + (hi - lo) / 2, tmp = n - 1, count = 0;

  // For each row, we count the elements that are smaller then mid
      for (int i = 0; i < n; i++) {
        while (tmp >= 0 && matrix[i][tmp] > mid) tmp--;
        count += tmp + 1;
      }

      if (count < k) lo = mid + 1;
      else hi = mid;
    }

    return lo;
  }
```

```
// 386. Lexicographical Numbers
  vector<int> lexicalOrder(int n) {
    if(n==0) return {};
    vector<int> result;
    int current=1; //Initial element
    for(int i=0;i<n;i++){
      result.push_back(current); //Push current to the result.
      current*=10; // Add zero at the end of current.
      while(current>n){ //If current exceeds n.
        current/=10; //Fall back to last element.
        current++; //Get Next in order.
```

```
            while(current%10==0) current/=10; //Remove extra trailing zeros.
        }
    }
    return result;
}

// 400. Nth Digit lexocographical
  int findNthDigit(int n) {
     // step 1. calculate how many digits the number has.
    long base = 9, digits = 1;
    while (n - base * digits > 0)
    {
      n -= base * digits;
      base *= 10;
      digits ++;
    }

    // step 2. calculate what the number is.
    int index = n % digits;
    if (index == 0)
      index = digits;
    long num = 1;
    for (int i = 1; i < digits; i ++)
      num *= 10;
    num += (index == digits) ? n / digits - 1 : n / digits;;

    // step 3. find out which digit in the number is we wanted.
    for (int i = index; i < digits; i ++)
      num /= 10;
    return num % 10;
  }

// 390. Elimination Game
  int lastRemaining(int n) {
        if (n == 1) return 1;
    if (n <= 4) return 2;
    if (n % 2 != 0) n -= 1;
    if (n % 4 != 0) return 4 * lastRemaining(n/4);
    return 4 * lastRemaining(n/4) - 2;
  }


// 394. Decode String
// Input: s = "3[a]2[bc]"
// Output: "aaabcbc"
  string repeat(string str, int times) {
    string res = "";
    for (int i=0; i<times; i++)
      res += str;
    return res;
  }
  string decodeString(string s) {
    int i=0;
    while (i < s.size()) {
      if (s[i] != ']') {
```

```
            i++;
            continue;
        }

        int j = i;
        while (s[j] != '[')
            j--;

        string letters_to_repeat = s.substr(j+1, i-j-1);
        int k = j;
        j--;
        while ((j > 0) &&(isdigit(s[j])))
            j--;

        if (j != 0) j++;
        int times_to_repeat = stoi(s.substr(j, k-j));

        s.replace(j, i-j+1, repeat(letters_to_repeat, times_to_repeat));

        i = j+letters_to_repeat.size()*times_to_repeat;
    }
    return s;
}
```

```
// 395. Longest Substring with At Least K Repeating Characters
// Input: s = "ababbc", k = 2
// Output: 5
// Explanation: The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.
int longestSubstring(string s, int k) {
    int n = s.size();
    if ((n == 0) && (k > n))
        return 0;

    unordered_map<char, int> counter;
    for (auto letter : s)
        counter[letter]++;

    int sub1, sub2;

    for (int i=0; i<n; i++) {
        if (counter[s[i]] < k) {
            sub1 = longestSubstring(s.substr(0, i), k);
            sub2 = longestSubstring(s.substr(i+1), k);
            break;
        }
        if (i == n-1)
            return n;
    }

    return max(sub1, sub2);
}
```

```
// 397. Integer Replacement
```

```
// Input: n = 8
// Output: 3
// Explanation: 8 -> 4 -> 2 -> 1
    int f(int n){
        if(n==1) return 0;
        if(n&1) return 1+ min(f(n +1) , f(n-1));
        else return 1+ f(n/2);
    }
    int integerReplacement(int n) {
        if(n == INT_MAX) return 32;
        if(n==1) return 0;
        return f(n);
    }

// 402. Remove K Digits
    string removeKdigits(string num, int k) {
        string ans="";
        for(char &c:num){
            while(ans.size() && ans.back()>c && k){
                ans.pop_back();
                k--;
            }
            if(ans.size()||c!='0')ans.push_back(c);
        }
        while(ans.size() && k--)
            ans.pop_back();

        return (ans=="")?"0":ans;


// 406. Queue Reconstruction by Height
// Input: people = [[7,0],[4,4],[7,1],[5,0],[6,1],[5,2]]
// Output: [[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]
    static bool cmp(vector<int>& p1, vector<int>& p2){
        if(p1[0]!=p2[0]) return p1[0]>p2[0];
        else return p1[1]<p2[1];
    }
    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
        sort(people.begin(), people.end(), cmp);
        vector<vector<int>> res;
        for(int i = 0; i < people.size(); i++){
            res.insert(res.begin()+people[i][1], people[i]);
        }
        return res;
    }

// 413. Arithmetic Slices
    int numberOfArithmeticSlices(vector<int>& A) {
        vector<int> dp(A.size());
        int res = 0;
        for (int i = 2; i < A.size(); i++) {
            if (A[i] - A[i-1] == A[i-1] - A[i-2]) {
                dp[i] = dp[i-1] + 1;
                res += dp[i];
            }
```

```
        }
        return res;
    }


//417. Pacific Atlantic Water Flow
    int m, n;
    vector<vector<bool> > atlantic, pacific;
 vector<vector<int> > ans;
    vector<vector<int> > pacificAtlantic(vector<vector<int>>& mat) {
        if(!size(mat)) return ans;
        m = size(mat), n = size(mat[0]);
        atlantic = pacific = vector<vector<bool> >(m, vector<bool>(n, false));
        for(int i = 0; i < m; i++) dfs(mat, pacific, i, 0), dfs(mat, atlantic, i, n - 1);
        for(int i = 0; i < n; i++) dfs(mat, pacific, 0, i), dfs(mat, atlantic, m - 1, i);
        return ans;
    }
    void dfs(vector<vector<int> >& mat, vector<vector<bool> >& visited, int i, int j){
        if(visited[i][j]) return;
        visited[i][j] = true;
        if(atlantic[i][j] && pacific[i][j]) ans.push_back(vector<int>{i, j});
/*□  */  if(i + 1 <  m && mat[i + 1][j] >= mat[i][j]) dfs(mat, visited, i + 1, j);
/*□  */  if(i - 1 >= 0 && mat[i - 1][j] >= mat[i][j]) dfs(mat, visited, i - 1, j);
/*□  */  if(j + 1 <  n && mat[i][j + 1] >= mat[i][j]) dfs(mat, visited, i, j + 1);
/*□  */  if(j - 1 >= 0 && mat[i][j - 1] >= mat[i][j]) dfs(mat, visited, i, j - 1);
    }


// 419. Battleships in a Board
    int countBattleships(vector<vector<char>>& A) {
        int n = A.size(),m = A[0].size(),count=0;
        for( int i=0; i<n; i++ ){
            for( int j=0; j<m; j++ ){
                if( A[i][j]=='X' ){
                    if( i==0 && j==0 ) count++;
                    else if( i==0 ) {
                        if( A[i][j-1]!='X' )
                            count++;
                    }
                    else if( j==0 ){
                        if( A[i-1][j]!='X' )
                            count++;
                    }
                    else if( A[i-1][j]=='.' && A[i][j-1]=='.')
                        count++;
                }
            }
        }
        return count;
    }

// 423. Reconstruct Original Digits from English
    string originalDigits(string s) {
        vector<string> words = {"zero", "two", "four", "six", "eight", "one", "three", "five", "seven", "nine"};
        vector<int> nums = {0, 2, 4, 6, 8, 1, 3, 5, 7, 9};
```

```
        vector<int> distinct_char = {'z', 'w', 'u', 'x', 'g', 'o', 'r', 'f', 'v', 'i'};
        vector<int> counts(26, 0);
        string result;
        for(auto ch : s){ counts[ch-'a']++;}
        for(int i = 0; i < 10; i++){
            int count = counts[distinct_char[i]-'a'];
            for(int j = 0; j < words[i].size(); j++)
                counts[words[i][j]-'a'] -= count;
            while(count--)
                result += to_string(nums[i]);
        }
        sort(result.begin(), result.end());
        return result;
    }

// 424. Longest Repeating Character Replacement
    int characterReplacement(string s, int k) {
        int n = s.size();
        int i = 0, j = 0, maxi = 0;
        unordered_map<char,int>mp;
        int ans = -1;
        while(j < n)
        {
            mp[s[j]]++;
            maxi = max(maxi, mp[s[j]]);
            if((j-i+1) - maxi > k){
                mp[s[i]]--;
                i++;
            }
            ans = max(ans, (j-i+1));
            j++;
        }
        return ans;
    }

// 430. Flatten a Multilevel Doubly Linked List
    Node* flatten(Node* head, Node *parent = nullptr) {
        if (!head) {
            return nullptr;
        }

        Node *cur = head;
        while (cur->child || cur->next) {
            if (cur->child) {
                cur->next = flatten(cur->child, cur->next);
                cur->next->prev = cur;
                cur->child = nullptr;
            }

            cur = cur->next;
        }

        if (parent) {
      cur->next = parent;
            parent->prev = cur;
```

```
        }

        return head;
    }
```

// 435. Non-overlapping Intervals

```
    bool comp(vector<int> &a,vector<int> &b) {
 return a[1]<b[1];
}
 int eraseOverlapIntervals(vector<vector<int>>& intervals) {
  int ans=-1;
  if(intervals.size()==0) return 0;
  sort(intervals.begin(),intervals.end(),comp);

  vector<int> prev= intervals[0];

  for(vector<int> i: intervals) {
   if(prev[1]>i[0]) {
    ans++;
   }else prev=i;
  }
  return ans;
 }
```

// 436. Find Right Interval

```
    vector<int> findRightInterval(vector<vector<int>>& intervals) {
        int len = intervals.size();
        vector<int> res(len);
        pair<int, int> pos[len];
        auto endOfPos = pos + len;
        // populating and sorting the map of positions
        for (int i = 0; i < len; i++) pos[i] = {intervals[i][0], i};
        sort(pos, endOfPos);
        // finding the matching right interval
        for (int i = 0; i < len; i++) {
            auto p = lower_bound(pos, endOfPos, intervals[i][1], [](auto it, int val){return it.first < val;});
            res[i] = p == endOfPos ? -1 : p->second;
        }
        return res;
    }
```

// 437. Path Sum III

```
    int ans=0;
    int pathSum(TreeNode* root, int sum) {
        if(root){
            dfs(root,sum);
            pathSum(root->left,sum);
            pathSum(root->right,sum);
        }
        return ans;
    }
    void dfs(TreeNode* root, int sum){
```

```
        if(!root)return;
        if(root->val==sum)ans++;
        dfs(root->left,sum-root->val);
        dfs(root->right,sum-root->val);
    }

    unordered_map<int, int> map;
    int count = 0;

    void countPathSum(TreeNode* root, int target, int sum){
        if(!root)
            return;
        sum += root->val;
        if(sum == target)
            count++;
        if(map.find(sum - target) != map.end())
            count += map[sum - target];
        map[sum]++;
        countPathSum(root->left, target, sum);
        countPathSum(root->right, target, sum);
        map[sum]--;
    }

    int pathSum(TreeNode* root, int targetSum) {
        countPathSum(root, targetSum, 0);
        return count;
    }



// // 443. String Compression
// Input: chars = ["a","a","b","b","c","c","c"]
// Output: Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]
    int compress(vector<char>& chars) {
        if(chars.size()<2) return chars.size();
  int i=0,j=0;
  while(i<chars.size()) {
   chars[j] = chars[i];
   int cnt = 0;
   while(i < chars.size() && chars[i] == chars[j]) {
    cnt++;
    i++;
   }

   if(cnt == 1) {
    j++;
   } else {
    string str = to_string(cnt);
    for(auto ch: str)
     chars[++j] = ch;
    j++;
   }
  }

  return j;
```

```
    }
```

```
// 447. Number of Boomerangs
    int numberOfBoomerangs(vector<vector<int>>& points) {
        int ans = 0, n = points.size();
        for(int i=0;i<n;i++) {
            unordered_map<int,int>mp;
            for(int j=0;j<n;j++) {
                int dx = points[i][0] - points[j][0];
                int dy = points[i][1] - points[j][1];
                int dis = dx*dx + dy*dy;
                // we can pair jth point wi
                mp[dis]++;
            }
            for (auto& p : mp) {
                if (p.second > 1) {
                    ans += p.second * (p.second - 1);
                }
            }
        }
        return ans;
    }
```

```
// 452. Minimum Number of Arrows to Burst Balloons
    bool cmp(vector<int>& a, vector<int>& b) {return a[1] < b[1];}
    int findMinArrowShots(vector<vector<int>>& segments) {
        sort(segments.begin(), segments.end(), cmp);
        int ans = 0, arrow = 0;
        for (int i = 0; i < segments.size(); i ++) {
            if (ans == 0 || segments[i][0] > arrow) {
                ans ++;
                arrow = segments[i][1];
            }
        }
        return ans;
    }
```

```
// 523. Continuous Subarray Sum
bool checkSubarraySum(vector<int>& nums, int k) {

 int prefSum = 0;
 unordered_map<int, int> mp;
 for(int i=0; i<nums.size(); i++)
 {
  prefSum += nums[i];
  prefSum %= k;

  if(prefSum == 0 && i) return true;

  // cout << prefSum << " ";
  if(mp.find(prefSum) != mp.end())  // Found the required prefix sum
  {
```

```
   if(i - mp[prefSum] > 1) return true; // check if atleast 2 elements are there or not
  }
  else mp[prefSum] = i;
 }

 return false;
}

//168. Excel Sheet Column Title
    string convertToTitle(int n) {
        string s="";
        n--;
        while(n>=0){
            s+=('A'+n%26);
            n/=26;
            n--;
        }
        reverse(s.begin(),s.end());
        return s;
    }

// 202. Happy Number
    int help(int n){
        int sum = 0;
        while(n){
            int a = n%10;
            sum = (a*a )+sum;
            n=n/10;
        }
        return sum;
    }
    bool isHappy(int n) {
        int slow=n, fast =n;
        do{
            slow = help(slow);
            fast = help(help(fast));
        } while(slow!= fast);

        return (slow==1);
    }


// 453. Minimum Moves to Equal Array Elements
// Input: nums = [1,2,3]
// Output: 3
// Explanation: [1,2,3]  =>  [2,3,3]  =>  [3,4,3]  =>  [4,4,4]
int minMoves(vector<int>& nums) {
    int sum = 0 , mn = INT_MAX, n = nums.size();
    for (auto  i : nums){
        sum += i;
        mn = min(mn, i) ;
    }
    return  sum-(mn *n) ;
}
```

```
// 462. Minimum Moves to Equal Array Elements II
    int minMoves2(vector<int>& nums) {
        sort(begin(nums), end(nums));
        int moves = 0, median = nums[size(nums) / 2];
        for(auto num : nums) moves += abs(num - median);
        return moves;
    }

// 467. Unique Substrings in Wraparound String
// "...zabcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabcd....".
    int findSubstringInWraproundString(string p) {
        vector<int> cnt(26,0);
        int n=p.length(), mx=1;
        for(int i=0;i<n;i++){
            if(i>0 and (p[i]-p[i-1]==1 or p[i-1]-p[i]==25))
                mx++;
            else
                mx=1;
            cnt[p[i]-'a']=max(cnt[p[i]-'a'],mx);
        }
        int ans=0;
        for(int i=0;i<26;i++)
            ans+=cnt[i];
        return ans;
    }


// 475. Heaters
    int findRadius(vector<int>& houses, vector<int>& heaters) {
        int radius = 0;
        sort(heaters.begin(), heaters.end());
        for(auto house : houses) {
            if(house <= heaters.front()) {
                radius = max(radius, heaters.front() - house);
                continue;
            }
            if(house >= heaters.back()) {
                radius = max(radius, house - heaters.back());
                continue;
            }
            radius = max(radius, findAjacentHeaters(house, heaters));
        }
        return radius;
    }

    int findAjacentHeaters(int house, vector<int>& heaters) {
        int radius = 0;
        int l = 0;
        int r = heaters.size() - 1;
        while(l <= r) {
            int mid = l + (r - l) / 2;
            if(heaters[mid] == house) return 0;
            if(heaters[mid] < house) l = mid + 1;
            if(heaters[mid] > house) r = mid - 1;
```

```
        }
        return min(heaters[l]-house, house-heaters[r]);
    }
```

// 477. Total Hamming Distance

```
    int totalHammingDistance(vector<int>& nums) {
        int n=nums.size(), ans=0;
        for(int i=0;i<32;i++){
            int count=0; //Count of the no.of elements that have the 'i'th bit set
            for(int k=0;k<n;k++){
                count += (nums[k]>>i)&1;
            }
            ans += count*(n-count);
        }
        return ans;
    }
```

// 486. Predict the Winner

```
    int static check(vector<int>& nums,int i,int j,int chance){

        if(i>j) return (0);

        if(chance==0){
            return max(nums[i] + check(nums,i + 1,j,1),nums[j]+ check(nums,i,j-1,1));
        }else{
            return min(check(nums,i + 1,j,0) ,check(nums,i,j-1,0));
        }
    }
    bool PredictTheWinner(vector<int>& nums) {
       int player2=0;

        for(auto x:nums) player2+=x;


         int player1=check(nums,0,nums.size() - 1,0);
        player2-=player1;

        return player1>=player2;
    }
```

// 508. Most Frequent Subtree Sum

```
    int findSubtreeSum(TreeNode* root, unordered_map<int,int> &map,int &maxFrequency){

       if(root == NULL) return 0;

        int left = findSubtreeSum(root->left,map,maxFrequency);
        int right = findSubtreeSum(root->right,map,maxFrequency);

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

        map[currSubTreeSum]++ ;

        if(map[currSubTreeSum]>maxFrequency)
            maxFrequency = map[currSubTreeSum];
```

```
        return currSubTreeSum;

    }

    vector<int> findFrequentTreeSum(TreeNode* root) {
        unordered_map<int,int>map;
        int maxFrequency = 0 ;
        findSubtreeSum(root,map,maxFrequency);
        vector<int>result;

        for(auto element : map)
            if(element.second == maxFrequency)
                result.push_back(element.first);

        return result;
    }

// 513. Find Bottom Left Tree Value
    int ans=0;
    int depth;
    bool mila=false;
    void dfs(TreeNode* root, int l){
        if(root==NULL)
            return;

        dfs(root->left, l+1);
        if(root->left == NULL and root->right==NULL and l>depth)
            depth=l;
        dfs(root->right, l+1);
    }
    void helper(TreeNode* root, int d){
        if(root==NULL)
            return;
        helper(root->left, d+1);
        if(root->left==NULL and root->right==NULL and depth == d and !mila){
            ans = root->val;
            mila= true;
        }
        helper(root->right, d+1);
    }

    int findBottomLeftValue(TreeNode* root) {
        depth=0;
        dfs(root, 0);
        helper(root, 0);
        return ans;
    }

// 524. Longest Word in Dictionary through Deleting
    bool canFormByDeleting(string word, string str) {
        int word_i = 0, str_i = 0;
        while (word_i < word.size() && str_i < str.size()) {
            if (word[word_i] == str[str_i])
                word_i++;
```

```
            str_i++;
        }
        return word_i == word.size();
    }

    string findLongestWord(string s, vector<string>& d) {
        string res = "";
        for (auto str : d) {
            if (canFormByDeleting(str, s)) {
                if (str.size() > res.size() || (str.size() == res.size() && str < res))
                    res = str;
            }
        }
        return res;
    }
```

```
// 525. Contiguous Array
    int findMaxLength(vector<int>& nums) {
        int sum=0, maxLen=0;
        unordered_map<int, int> seen{{0, -1}};

        for(int i=0; i<nums.size(); i++){
            sum += nums[i]==1 ? 1 : -1;
            if(seen.count(sum)) maxLen = max(maxLen, i-seen[sum]);
            else seen[sum] = i;
        }
        return maxLen;
    }
```

```
// 526. Beautiful Arrangement
    bool seen[16] = {};
    int res = 0;
    int dfs(int n, int pos = 1) {
        if (pos > n) return res++;
        for (int i = 1; i <= n; i++) {
            if (!seen[i] && (i % pos == 0 || pos % i == 0)) {
                // marking i as seen
                seen[i] = true;
                dfs(n, pos + 1);
                // backtracking
                seen[i] = false;
            }
        }
        return res;
    }
    int countArrangement(int n) {
        if (n < 4) return n;
        return dfs(n);
    }
```

```
// 623. Add One Row to Tree
TreeNode* addOneRow(TreeNode* root, int v, int d, bool isLeft = true) {
    if ( d == 1 )
    {
```

```
        TreeNode *left = isLeft? root : NULL, *right = isLeft? NULL : root;
        return new TreeNode(v, left, right);
    }

    if ( root )
    {
        root->left = addOneRow(root->left, v, d - 1);
        root->right = addOneRow(root->right, v, d - 1, false);
    }

    return root;
}

// 611. Valid Triangle Number
int triangleNumber(vector<int>& nums) {
    int res = 0, n = nums.size();
    sort(nums.begin(), nums.end());
    for (int i = n-1; i >= 0; i--) {
        int lo = 0, hi = i-1;

        while (lo < hi) {
            if (nums[lo] + nums[hi] > nums[i]) {
                res += hi - lo;
                hi--;
            }

            else lo++;
        }
    }

    return res;
}

// 593. Valid Square
int d(vector<int>& p1, vector<int>& p2) {
    return (p1[0] - p2[0]) * (p1[0] - p2[0]) + (p1[1] - p2[1]) * (p1[1] - p2[1]);
}
bool validSquare(vector<int>& p1, vector<int>& p2, vector<int>& p3, vector<int>& p4) {
    unordered_set<int> s({ d(p1, p2), d(p1, p3), d(p1, p4), d(p2, p3), d(p2, p4), d(p3, p4) });
    return !s.count(0) && s.size() == 2;
}



// 583. Delete Operation for Two Strings
    int minDistance(string word1, string word2) {
        int lcs = LCS(word1, word2);
        return word1.size() + word2.size() - (2 * lcs);
    }
    int LCS(string w1, string w2) {
        int n = w1.size(), m = w2.size();
        vector<vector<int>> dp(n+1, vector<int>(m+1, -1));

        for (int i = 0; i <= n; i++) dp[i][0] = 0;
        for (int i = 0; i <= m; i++) dp[0][i] = 0;
```

```
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                dp[i][j] = w1[i-1] == w2[j-1] ? 1 + dp[i-1][j-1] : max(dp[i-1][j], dp[i][j-1]);
            }
        }

        return dp[n][m];
    }
```

```
// 581. Shortest Unsorted Continuous Subarray
    int findUnsortedSubarray(vector<int>& nums) {
        vector<int> sorted(nums);
        sort(sorted.begin(), sorted.end());
        int n = nums.size(), i = 0, j = n - 1;
        while (i < n && nums[i] == sorted[i]) {
            i++;
        }
        while (j > i && nums[j] == sorted[j]) {
            j--;
        }
        return j + 1 - i;
    }
```

```
// 576. Out of Boundary Paths
        int m, n;
    int memo[50][50][51];
    int DIR[5] = {0, 1, 0, -1, 0};
    int findPaths(int m, int n, int maxMove, int startRow, int startColumn) {
        this->m = m; this->n = n;
        memset(memo, -1, sizeof(memo));
        return dp(startRow, startColumn, maxMove);
    }
    int dp(int r, int c, int maxMove) {
        if (r < 0 || r == m || c < 0 || c == n) return 1;
        if (maxMove == 0) return 0;
        if (memo[r][c][maxMove] != -1) return memo[r][c][maxMove];
        int ans = 0;
        for (int i = 0; i < 4; ++i)
            ans = (ans + dp(r + DIR[i], c + DIR[i+1], maxMove - 1)) % 1000000007;
        return memo[r][c][maxMove] = ans;
    }
```

```
// 567. Permutation in String
    bool checkInclusion(string s1, string s2) {
        vector<int> cur(26), goal(26);
        for(char c : s1) goal[c - 'a']++;
        for(int i = 0; i < s2.size(); i++) {
            cur[s2[i] - 'a']++;
            if(i >= s1.size()) cur[s2[i - s1.size()] - 'a']--;
            if(goal == cur) return true;
        }
        return false;
    }
```

```
// 554. Brick Wall
int leastBricks(vector<vector<int>>& wall) {
        int rows = size(wall), maxBrickEdges = 0, idx;
    unordered_map<int, int> edgesFrequency;
    for(auto& row : wall) {
        idx = 0;
        for(int i = 0; i < size(row) - 1; i++)
            idx += row[i], edgesFrequency[idx]++;
    }
    for(auto& pair : edgesFrequency)
        maxBrickEdges = max(maxBrickEdges, pair.second);
    return rows - maxBrickEdges;
}


// 537. Complex Number Multiplication
    pair<int, int> parse(string num) {
        int i = num.find('+');
        double real = stoi(num.substr(0, i));
        double imaginary = stoi(num.substr(i+1, num.size()-i-2));
        pair<int, int> res(real, imaginary);
        return res;
    }

    string complexNumberMultiply(string num1, string num2) {
        pair<int, int> a = parse(num1), b = parse(num2);
        int real_a = a.first, imag_a = a.second;
        int real_b = b.first, imag_b = b.second;

        return to_string(real_a * real_b - imag_a * imag_b) + '+' + to_string(real_a * imag_b  + real_b * imag_a)+'i' ;
    }


// 538. Convert BST to Greater Tree
    void fill(TreeNode * root , int &ans){
        if(!root) return ;
        fill(root->right , ans);
        root->val += ans;
        ans = root->val;
        fill(root->left , ans);
    }
    TreeNode* convertBST(TreeNode* root) {
        int ans = 0 ;
        fill(root , ans);
        return root;
    }

// 539. Minimum Time Difference
    int findMinDifference(vector<string>& timePoints) {
            int m=timePoints.size();
        vector<int>time;

        for(int i=0;i<m;i++)
          time.push_back(stoi(timePoints[i].substr(0,2))*60 +stoi(timePoints[i].substr(3,2)));
```

```
        sort(time.begin(),time.end());
        int ans=INT_MAX;

        for(int i=1;i<m;i++)
         ans=min(time[i]-time[i-1],ans);

        ans=min(ans,time[0]-time[m-1]+24*60);
        return ans;
    }
```

```
// 540. Single Element in a Sorted Array
    int singleNonDuplicate(vector<int>& nums) {
        int low = 0, high = nums.size()-2 ;
        while( low<=high){
            int mid = (low+high)/2;
            if(nums[mid] == nums[mid^1])
                low = mid+1;
            else
                high = mid-1;
        }
        return nums[low];
    }
```

```
// 535. Encode and Decode TinyURL
    unordered_map<string, string> long_to_short;
    unordered_map<string, string> short_to_long;
    string encode(string longUrl) {
        long_to_short[longUrl] = "http://tinyurl.com/" + to_string(hash_function(longUrl));
        short_to_long[long_to_short[longUrl]] = longUrl;
        return long_to_short[longUrl];
    }
    string decode(string shortUrl) {
        return short_to_long[shortUrl];
    }
    uint64_t hash_function(const string& str) {
        uint64_t hash = 0x811c9dc5;
        uint64_t prime = 0x1000193;

        for(int i = 0; i < str.size(); ++i) {
            uint8_t value = str[i];
            hash = hash ^ value;
            hash *= prime;
        }

        return hash;
    }
```

```
// 1423. Maximum Points You Can Obtain from Cards
    int maxScore(vector<int>& C, int K) {
        int total = 0;
        for (int i = 0; i < K; i++) total += C[i];
        int best = total;
        for (int i = K - 1, j = C.size() - 1; ~i; i--, j--)
```

```
        total += C[j] - C[i], best = max(best, total);
    return best;
}
```

// 532. K-diff Pairs in an Array

```
int findPairs(vector<int>& nums, int k) {
    unordered_map<int,int> a;
    for(int i:nums) a[i]++;
    int ans=0;
    for(auto x:a){
        if(k==0){
            if(x.second>1)
            ans++;
        }
         else if(a.find(x.first+k)!=a.end())
            ans++;
    }
    return ans;
}
```

// 633. Sum of Square Numbers

```
#define lli long long int
bool judgeSquareSum(int c) {
   unordered_map<lli,lli>mp;
    for(lli i{0};i*i<=c;i++){
        mp[i*i]=i;
    }
    for(auto it:mp){
        lli first_num=it.first;
        lli sec_num=c-it.first;
        if(mp.find(sec_num)!=mp.end()){
          return true;
        }
    }
  return false;
}
```

```
int findLongestSubarray(int N, vector<long long>v){
    int ans=-1, len = 0;
    for(int i=0;i<n;i++){
        int temp = v[i], lenTemp =0;
        for(int j=i;j<n;j++){
            temp&=v[i];
            if(temp>ans){
                len = max(len, lenTemp);
            }
        }
    }
    return len;
}
```

// 638. Shopping Offers

```
    int solve(vector<int>& price, vector<vector<int>>& special, vector<int>& needs,int ind) {
        int n = needs.size();
        if(ind<0){
            int ans =0;
            for(int i=0;i<needs.size();i++)
                 ans += needs[i]*price[i];
            return ans;
        }
        else{
            bool lte = true;
            for(int i=0;i<n;i++){
                if(needs[i]<special[ind][i]){
                    lte = false;
                    break;
                }
            }
            if(lte){
                int op1 = solve(price,special,needs,ind-1);
                int op2 = 0;
                for(int i=0;i<n;i++)
                    needs[i] = needs[i] - special[ind][i];

                op2 = special[ind][n] + solve(price,special,needs,ind);

                for(int i=0;i<n;i++)
                {
                    needs[i] = needs[i] + special[ind][i];
                }

                return min(op1,op2);
            }
            else
            {
                return solve(price,special,needs,ind-1);
            }
        }
    }
    int shoppingOffers(vector<int>& price, vector<vector<int>>& special, vector<int>& needs) {
        int ind=special.size()-1;
        return solve(price,special,needs,ind);
    }


// 647. Palindromic Substrings
    int countSubstrings(string s) {
        vector<vector<int>> mem(s.size(), vector<int>(s.size(), -1));
        int count = 0;
        for(int i = 0; i < s.size(); ++i) {
            for(int j = i; j < s.size(); ++j) {
                count += solve(mem, s, i, j);
            }
        }
        return count;
    }
```

```
    int solve(vector<vector<int>>& mem, string& s, int i, int j) {
        if (i >= j) return 1;
        if (mem[i][j] >= 0) return mem[i][j];
        return mem[i][j] = s[i] == s[j] ? solve(mem, s, i+1, j-1) : 0;
    }
```

```
// 648. Replace Words
 string replaceWords(vector<string>& dict, string sentence) {
        unordered_map<string,int> dics;
        int max_len = 0;
        for(string s: dict){
            dics[s]++;
            max_len = max(max_len,(int)s.size());
        }
        string ans = "";
        stringstream ss(sentence);
        string word;
        while(ss>>word){
            int len = word.size();
            bool flag = 0;
            for(int i=1;i<=min(len,max_len);++i){
                if(dics.find(word.substr(0,i))!=dics.end()){
                    ans += " "+word.substr(0,i);
                    flag = 1;
                    break;
                }
            }
            if(!flag) ans+=" "+word;
        }
        return ans.substr(1);
    }
```

```
// 650. 2 Keys Keyboard
    int minKeyPress(int step, int value, int copy, int&n)
    {
        if(step>n || value>n) return INT_MAX;
        if(value==n) return step;
        if(dp[step][value]!=-1) return dp[step][value];

        return dp[step][value] = min(minKeyPress(step+1,value+copy,copy,n),minKeyPress(step+2,2*value,value,n));
    }

    int minSteps(int n) {
        if(n==1) return 0;
        memset(dp,-1,sizeof(dp));
        return minKeyPress(1,1,1,n);
    }
```

```
// 655. Print Binary Tree
    vector<vector<string>> printTree(TreeNode* root) {
        int h = get_height(root), w = get_width(root);
        vector<vector<string>> ans(h, vector<string>(w, ""));
        helper(ans, root, 0, 0, w-1);
```

```
        return ans;
    }
    int get_height(TreeNode* p) {
        if (!p) return 0;
        int left = get_height(p->left), right = get_height(p->right);
        return max(left, right)+1;
    }
    int get_width(TreeNode* p) {
        if (!p) return 0;
        int left = get_width(p->left), right = get_width(p->right);
        return max(left, right)*2+1;
    }
    void helper(vector<vector<string>>& ans, TreeNode* p, int level, int l, int r) {
        if (!p) return;
        int mid = l+(r-l)/2;
        ans[level][mid] = to_string(p->val);
        helper(ans, p->left, level+1, l, mid-1);
        helper(ans, p->right, level+1, mid+1, r);
    }

// 658. Find K Closest Elements
    vector<int> findClosestElements(vector<int>& arr, int k, int x) {
     //way 1: binary search ()sorted array given
        int left = 0, right = arr.size() - k;
        while (left < right) {
            int mid = (left + right) / 2;
            if (x - arr[mid] > arr[mid + k] - x)
                left = mid + 1;
            else
                right = mid;
        }
        return vector<int>(arr.begin() + left, arr.begin() + left + k);
    //way 2: priority queue
        // priority_queue<pair<int,int>> q;
        // for(auto it:arr){
        //     q.push({abs(it-x),it});
        //     if(q.size()>k)
        //         q.pop();
        // }
        // vector<int> ans;
        // while(!q.empty()){
        //     ans.push_back(q.top().second);
        //     q.pop();
        // }
        // sort(ans.begin(),ans.end());
        // return ans;
    }

// 665. Non-decreasing Array - atmost one swap
    bool checkPossibility(vector<int>& nums) {
        int cnt = 0;
        for(int i = 1; i < nums.size() && cnt<=1 ; i++){
            if(nums[i-1] > nums[i]){
                cnt++;
                if(i-2<0 || nums[i-2] <= nums[i])nums[i-1] = nums[i];
```

```
            else nums[i] = nums[i-1];
        }
    }
    return cnt<=1;
}
```

```
// 667. Beautiful Arrangement II
vector<int> constructArray(int n, int k) {
    int diff = n - k, lo = 1, hi = n;
    vector<int> out;
    int i = 0;
    while(i < diff){
        out.push_back(lo);
        lo++;
        i++;
    }
    bool flag = true;
    for(int i = out.size(); i < n ; i++){
        if(flag){
            out.push_back(hi);
            hi--;
            flag = false;
        }
        else{
            out.push_back(lo);
            lo++;
            flag = true;
        }
    }
    return out;
}
```

```
// 669. Trim a Binary Search Tree
    TreeNode* trimBST(TreeNode* root, int L, int R) {
        if (!root) return root;
        if (root->val >= L && root->val <= R) {
            root->left = trimBST(root->left, L, R);
            root->right = trimBST(root->right, L, R);
            return root;
        }
        if (root->val < L)
            return trimBST(root->right, L, R);
        return trimBST(root->left, L, R);
    }
```

```
// 670. Maximum Swap
// swap two digits at most once to get the maximum valued number.
    int maximumSwap(int num) {
        string n = to_string(num);
        unordered_map<int, int> last;
        for (int i=0; i<n.size(); i++)
            last[n[i] - '0'] = i;
```

```
        for (int i=0; i < n.size(); i++) {
            for (int j = 9; j > n[i]-'0'; j--) {
                if (last[j] > i) {
                    swap(n[last[j]], n[i]);
                    return stoi(n);
                }
            }
        }
        return stoi(n);
    }
```

```
// Largest number in K swaps
void solve(string str,string &ans,int ind,int k,int n){
    if(k==0)return;
    char mxchar=str[ind];
    for(int i=ind+1;i<n;i++){
        if(mxchar<str[i])mxchar=str[i];
    }
    if(mxchar!=str[ind])k--;
    for(int i=n-1;i>=ind;i--){
        if(str[i]==mxchar){
            swap(str[ind],str[i]);
            if(ans<str)ans=str;
            solve(str,ans,ind+1,k,n);
            swap(str[ind],str[i]);

        }
    }
    return;
}
string findMaximumNum(string str, int k)
{
    string ans=str;
    int n=str.length();
    solve(str,ans,0,k,n);
    return ans;
}
```

```
// 673. Number of Longest Increasing Subsequence
    int findNumberOfLIS(vector<int>& nums) {
        int n = nums.size();
        if (nums.size() <= 1)
            return nums.size();

        vector<int> dp_len(n, 1);
        vector<int> dp_count(n, 1);

        for (int i=1; i<n; i++) {
            for (int j=0; j<i; j++) {
                if (nums[j] < nums[i]) {
                    if (dp_len[i] <= dp_len[j]){
                        dp_len[i] = dp_len[j]+1;
                        dp_count[i] = dp_count[j];
```

```
                }
                else if (dp_len[j]+1 == dp_len[i])
                    dp_count[i] += dp_count[j];
            }
        }
    }

    int max_length = *max_element(dp_len.begin(), dp_len.end());
    int res = 0;
    for (int i=0; i<n; i++) {
        if (dp_len[i] == max_length)
            res += dp_count[i];
    }

    return res;
}


// 678. Valid Parenthesis String
bool checkValidString(string s) {
    stack<int> asterisk;
    stack<int> validPar;

    for(int i = 0; i < s.length(); i++){
        if(s[i] == '(') validPar.push(i);
        if(s[i] == '*') asterisk.push(i);
        if(s[i] == ')'){
            if(!validPar.empty())
                validPar.pop();
            else if(!asterisk.empty())
                asterisk.pop();
            else
                return false;
        }
    }

    while(!validPar.empty() && !asterisk.empty()){
        if(validPar.top() > asterisk.top()) return false;

        validPar.pop(); asterisk.pop();
    }

    return validPar.empty();
}


// 684. Redundant Connection
class DSU {
    vector<int> par, rank;
public:
    DSU(int n) : par(n), rank(n) {
        iota(begin(par), end(par), 0);
    }
    int find(int x) {
        if(x == par[x]) return x;
```

```
        return par[x] = find(par[x]);
    }
    bool Union(int x, int y) {
        int xp = find(x), yp = find(y);
        if(xp == yp) return false;                =
        if(rank[xp] > rank[yp]) par[yp] = par[xp];
        else if(rank[yp] > rank[xp]) par[xp] = par[yp];
        else par[xp] = yp, rank[yp]++;
        return true;
    }
};
class Solution {
public:
    vector<int> findRedundantConnection(vector<vector<int>>& e) {
        DSU ds(size(e) + 1);
        for(auto& E : e)
            if(!ds.Union(E[0], E[1])) return E;
        return { };
    }

// 686. Repeated String Match
     int repeatedStringMatch(string A, string B) {
        string s="";
        int count = 0;
        while(s.size()<B.size())
        {
            s+=A;
            count++;
        }
        if(s.find(B)!=string::npos)
            return count;
        s+=A;
        count++;
        if(s.find(B)!=string::npos)
            return count;

        return -1;

    }

// 687. Longest Univalue Path
    int longestUnivaluePath(TreeNode* root) {
        if (!root) return 0;
        int longestPath=0;
        go(root, longestPath);
        return longestPath;
    }
    int go(TreeNode* root, int& m){
        int l=root->left ? go(root->left, m) : 0;
        int r=root->right ? go(root->right, m) : 0;
        l=(root->left && root->left->val==root->val) ? l+1 : 0;
        r=(root->right && root->right->val==root->val) ? r+1 : 0;
        m=max(m,l+r);
        return max(l,r);
    }
```

```
// 688. Knight Probability in Chessboard
double knightProbability(int N, int K, int r, int c){
    if(K==0) return 1.0;
    vector<vector<double>> parentBoard(N,vector<double>(N,0.0));
    vector<vector<double>> childBoard(N,vector<double>(N,0.0));
    int rowoffset[] = {-2,-2,-1,-1,2,2,1,1};
    int coloffset[] = {1,-1,2,-2,1,-1,2,-2};

    int cx,cy;
    parentBoard[r][c] = 1.0;
    for(int i=0;i<K;i++)
    {
        for(int p=0;p<N;p++)
        {
            for(int q=0;q<N;q++)
            {
                double moveProb = parentBoard[p][q]/8.0;
                for(int w=0;w<8;w++)
                {
                    cx = p + rowoffset[w];
                    cy = q + coloffset[w];

                    if(cx>=0 && cx<N && cy>=0 && cy<N)
                        childBoard[cx][cy] += moveProb;
                }
            }
        }

        parentBoard = childBoard;
        fill(childBoard.begin(),childBoard.end(),vector<double>(N,0.0));
    }
    double knightProb = 0.0;
    for(int p=0;p<N;p++)
        for(int q=0;q<N;q++)
                knightProb+=parentBoard[p][q];

    return knightProb;
}


// 690. Employee Importance
    int getImportance(vector<Employee*> employees, int id) {
        unordered_map<int, Employee*>m;
        for(auto x: employees) m[x->id] = x;
        int sum = 0;
        DFS(m, id, sum);
        return sum;
    }

    void DFS(unordered_map<int, Employee*>& m, int id, int& sum){
        sum += m[id]->importance;
        for(auto x: m[id]->subordinates) DFS(m, x, sum);
    }
```

```
// 695. Max Area of Island
int maxAreaOfIsland(vector<vector<int>>& grid) {
    int max_area = 0;
    for(int i = 0; i < grid.size(); i++)
        for(int j = 0; j < grid[0].size(); j++)
            if(grid[i][j] == 1)max_area = max(max_area, AreaOfIsland(grid, i, j));
    return max_area;
}
int AreaOfIsland(vector<vector<int>>& grid, int i, int j){
    if( i >= 0 && i < grid.size() && j >= 0 && j < grid[0].size() && grid[i][j] == 1){
        grid[i][j] = 0;
        return 1 + AreaOfIsland(grid, i+1, j) + AreaOfIsland(grid, i-1, j) + AreaOfIsland(grid, i, j-1) + AreaOfIsland(gri
d, i, j+1);
    }
    return 0;
}


// 713. Subarray Product Less Than K
int numSubarrayProductLessThanK(vector<int>& nums, int k) {
    if(k <= 1) return 0;
    int prod = 1, res = 0, left = 0;
    for(int right = 0; right < nums.size(); right++) {
        prod *= nums[right];
        while(prod >= k) {
            prod /= nums[left];
            left++;
        }
        res += right - left + 1;
    }
    return res;
}


// 718. Maximum Length of Repeated Subarray
    int findLength(vector<int>& A, vector<int>& B) {
        int m = size(A), n = size(B), ans = 0, dp[m+1][n+1];
  memset(dp, -1, sizeof dp);
  for(int i = 0; i < m; i++) {
   for(int j = 0, len = 0; j < n; j++) {
    if(dp[i][j] == -1) {
     while(i + len < m and j + len < n and A[i+len] == B[j+len])
                    len++;
     while(len)
                    dp[i + len][j + len] = len--;
    }
    ans = max(ans, dp[i][j]);
   }
  }
  return ans;
    }

// 720. Longest Word in Dictionary
```

```
string longestWord(vector<string>& words) {
    sort(words.begin(), words.end());
    unordered_set<string> built;
    string res;
    for (string w : words) {
        if (w.size() == 1 || built.count(w.substr(0, w.size() - 1))) {
            res = w.size() > res.size() ? w : res;
            built.insert(w);
        }
    }
    return res;
}
```

```
// 721. Accounts Merge
    vector<vector<string>> accountsMerge(vector<vector<string>>& acts) {
        map<string, string> owner;
        map<string, string> parents;
        map<string, set<string>> unions;
        for (int i = 0; i < acts.size(); i++) {
            for (int j = 1; j < acts[i].size(); j++) {
                parents[acts[i][j]] = acts[i][j];
                owner[acts[i][j]] = acts[i][0];
            }
        }
        for (int i = 0; i < acts.size(); i++) {
            string p = find(acts[i][1], parents);
            for (int j = 2; j < acts[i].size(); j++)
                parents[find(acts[i][j], parents)] = p;
        }
        for (int i = 0; i < acts.size(); i++)
            for (int j = 1; j < acts[i].size(); j++)
                unions[find(acts[i][j], parents)].insert(acts[i][j]);

        vector<vector<string>> res;
        for (pair<string, set<string>> p : unions) {
            vector<string> emails(p.second.begin(), p.second.end());
            emails.insert(emails.begin(), owner[p.first]);
            res.push_back(emails);
        }
        return res;
    }
    string find(string s, map<string, string>& p) {
        return p[s] == s ? s : find(p[s], p);
    }
```

```
// 722. Remove Comments
    vector<string> removeComments(vector<string>& source) {
        vector<string> ans;
        string s;
        bool comment = false;
        for(int i = 0; i < source.size(); i++) {
            for(int j = 0; j < source[i].size(); j++) {
                if(!comment && j + 1 < source[i].size() && source[i][j] == '/' && source[i][j+1]=='/') break;
```

```
            else if(!comment && j + 1 < source[i].size() && source[i][j] == '/' && source[i][j+1]=='*') comment = tru
e, j++;
            else if(comment && j + 1 < source[i].size() && source[i][j] == '*' && source[i][j+1]=='/') comment = fals
e, j++;
            else if(!comment) s.push_back(source[i][j]);
        }

        if(!comment && s.size()) ans.push_back(s), s.clear();
    }
    return ans;
}
```

// 729. My Calendar I

```
    unordered_map<int, int>bookings;
    bool book(int s1, int e1) {
        for(auto& [s2, e2] : bookings)
            if( !(s1 >= e2 || s2 >= e1) )
       return false;
        bookings[s1] = e1;
        return true;
    }
```

// 731. My Calendar II

```
    map<int, int> mp;
    bool book(int start, int end)
    {
        mp[start]++;
        mp[end]--;
        int booked = 0;
        for (auto it = mp.begin(); it != mp.end(); it++)
        {
            booked += it->second;
            if (booked == 3)
            {
                mp[start]--;
                mp[end]++;
                return false;
            }
        }
        return true;
    }
```

// 735. Asteroid Collision

```
    vector<int> asteroidCollision(vector<int>& ast) {
        int n = ast.size();
        stack<int> s;

        for(int i = 0; i < n; i++) {
            if(ast[i] > 0 || s.empty())
                s.push(ast[i]);
            else{
                while(!s.empty() and s.top() > 0 and s.top() < abs(ast[i]))
                    s.pop();
                if(!s.empty() and s.top() == abs(ast[i]))
```

```
                s.pop();
            else {
                if(s.empty() || s.top() < 0) {
                    s.push(ast[i]);
                }
            }
        }
    }
    vector<int> res(s.size());
    for(int i = (int)s.size() - 1; i >= 0; i--) {
        res[i] = s.top();
        s.pop();
    }
    return res;
}
```

```
// 738. Monotone Increasing Digits
int monotoneIncreasingDigits(int N) {
 if(N < 10){
     return N;
 }
 string s = to_string(N);
 int index = s.length();
 int i;
 for(i = index -1 ; i > 0; i--){
     if(s[i-1] > s[i]){
         s[i-1]--;
         index = i;
     }
 }
 for(i = index; i < s.length(); i++){
     s[i] = '9';
 }
 N = stoi(s);
 return N;
}
```

```
// 739. Daily Temperatures
vector<int> dailyTemperatures(vector<int>& T) {
    int n = T.size();
    stack<int> s;
    vector<int> ans(n, 0);
    for (int i = 0; i < n; ++i) {
        while (!s.empty() and T[s.top()] < T[i]) {
            int j = s.top();
            s.pop();
            ans[j] = i - j;
        }
        s.push(i);
    }
    return ans;
}
```

```
// 740. Delete and Earn
// Pick any nums[i] and delete it to earn nums[i] points. Afterwards, you must delete every element equal to nums[i]
- 1 and every element equal to nums[i] + 1.
    int deleteAndEarn(vector<int>& nums) {
        int n = 10001;

        vector<int> sum(n, 0);
        vector<int> dp(n, 0);

        for(auto num: nums){
            sum[num] += num;
        }

        dp[0] = 0;
        dp[1] = sum[1];

        for(int i=2; i<n; i++)
            dp[i] = max(dp[i-2] + sum[i], dp[i-1]);

        return dp[n-1];
    }


// 743. Network Delay Time
    int networkDelayTime(vector<vector<int>>& times, int n, int k) {
       vector<pair<int,int>> adj[n+1];
        for(int i=0;i<times.size();i++)
                adj[times[i][0]].push_back({times[i][1],times[i][2]});
        vector<int> dist(n+1,INT_MAX);
        priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;
        pq.push({0,k});
        dist[k]=0;
        while(!pq.empty())
        {
            pair<int,int> t=pq.top();
            pq.pop();
            for(pair<int,int> it:adj[t.second])
            {
                if(dist[it.first]>t.first+it.second)
                {
                    dist[it.first]=t.first+it.second;
                    pq.push({dist[it.first],it.first});
                }
            }
        }
        int res=0;
        for(int i=1;i<=n;i++)
        {
            if(dist[i]==INT_MAX)
                return -1;
            res=max(res,dist[i]);
        }
  return res;
    }
```

```
//    752. Open the Lock
    int openLock(vector<string>& deadends, string target) {
 unordered_set<string> dead(begin(deadends), end(deadends)), seen({"0000"});
 if(dead.find("0000") != end(dead)) return -1;
 if(target == "0000") return 0;
 queue<string> q({"0000"});
 int n, minTurns = 0;
 while(!q.empty()) {
  n = size(q), minTurns++;
  for(int i = 0; i < n; i++) {
   auto cur_str = q.front(); q.pop();
   for(int j = 0; j < 4; j++)
    for(auto adj_str : turn(cur_str, j))
     if(seen.find(adj_str) == end(seen) && dead.find(adj_str) == end(dead))
      if(adj_str == target) return minTurns;
      else q.push(adj_str), seen.insert(adj_str);
  }
 }
 return -1;
}

    vector<string> turn(string s, int i) {
        vector<string> res(2, s);
        res[0][i] = '0' + (res[0][i] - '0' + 1) % 10;
        res[1][i] = '0' + (res[1][i] - '0' - 1 + 10) % 10;
        return res;
    }


// 754. Reach a Number
// During the ith move (starting from i == 1 to i == numMoves), you take i steps in the chosen direction.
    int reachNumber(int target) {
        target=abs(target);
        int sum=0;
        int steps=0;
        while(sum<target){
            steps++;
            sum+=steps;
        }
        if(sum==target) //if n*(n+1)/2==target
            return steps;
        int diff=sum-target;
        if(diff%2==0)
            return steps;
        else{
            if(steps%2==0)
                return steps+1;
            return steps+2;
        }
        return -1;
    }


// 756. Pyramid Transition Matrix
```

```
    unordered_map<string,vector<char>>dp;
    unordered_map<string,bool>mem;
    bool backtrack(string bottom,string next,int index){
        //if we only have a single length pattern then return true
        if(bottom.size()==1){
            return mem[bottom]=true;
        }
        else if(next.size()==bottom.size()-1){
            return backtrack(next,"",0);
        }
        else if(mem.count(bottom)){
            return mem[bottom];
        }
        else {
            string key=bottom.substr(index,2);
            for(int i=0;i<dp[key].size();i++){
                next.push_back(dp[key][i]);
                bool isAns=backtrack(bottom,next,index+1);
                if(isAns){
                    return mem[bottom]=true;
                }
                next.pop_back();
            }
            //else return false
            return mem[bottom]=false;
        }
    }
    bool pyramidTransition(string bottom, vector<string>& allowed) {
        //storing all the allowed triangular patterns
        for(int i=0;i<allowed.size();i++){
            dp[allowed[i].substr(0,2)].push_back(allowed[i][2]);
        }
        return backtrack(bottom,"",0);
    }


// 763. Partition Labels
    vector<int> partitionLabels(string s) {
        int n = s.size();
        vector<int> ans;

        if(n == 0) return ans;

        vector<int> last_pos (26, -1);

        for(int i=n-1; i>=0; --i) {
            if(last_pos[s[i]-'a'] == -1)
                last_pos[s[i]-'a'] = i;
        }

        int minp = -1, plen = 0;

        for(int i=0; i<n; ++i) {
            int lp = last_pos[s[i]-'a'];
            minp = max(minp, lp);
```

```
        ++plen;

        if(i == minp) {
            ans.push_back(plen);
            minp = -1;
            plen = 0;
        }
    }

    return ans;
}
```

```
// 764. Largest Plus Sign
    void fillDP(vector<vector<int>>& dp, vector<vector<int>>& mat, int n) {
        int down, right, up, left;

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

            for (int j = 0; j < n; j++) {
                right = mat[i][j] ? right+1 : 0;
                dp[i][j] = min(dp[i][j], right);

                down = mat[j][i] ? down+1 : 0;
                dp[j][i] = min(dp[j][i], down);
            }
        }

        for (int i = 0; i < n; i++) {
            up = 0, left = 0;
            for (int j = n-1; j >= 0; j--) {
                left = mat[i][j] ? left+1 : 0;
                dp[i][j] = min(dp[i][j], left);

                up = mat[j][i] ? up+1 : 0;
                dp[j][i] = min(dp[j][i], up);
            }
        }
    }
    int orderOfLargestPlusSign(int n, vector<vector<int>>& mines) {
        vector<vector<int>> dp(n, vector<int>(n, INT_MAX));
        vector<vector<int>> mat(n, vector<int>(n, 1));

        for (auto mine : mines)
            mat[mine[0]][mine[1]] = 0;

        fillDP(dp, mat, n);

        int res = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                res = max(res, dp[i][j]);
            }
        }
```

```
        return res;
    }

// 769. Max Chunks To Make Sorted
    int maxChunksToSorted(vector<int>& arr) {
            int n = arr.size();
        int max_ele = -1;
        int count  = 0;
        for(int i=0;i<n;i++){
            max_ele = max(max_ele,arr[i]);
            if(max_ele == i)
                count++;
        }
        return count;
    }

// 775. Global and Local Inversions
    bool isIdealPermutation(vector<int>& A) {
            int cmax = 0, n = A.size();
        for (int i = 0; i < n - 2; ++i) {
            cmax = max(cmax, A[i]);
            if (cmax > A[i + 2]) return false;
        }
        return true;
    }


// 777. Swap Adjacent in LR String
    bool canTransform(string start, string end) {
        int n = start.size();
        string s1, s2;
        for (int i = 0; i < n; ++i)
            if (start[i] != 'X') s1 += start[i];
        for (int i = 0; i < n; ++i)
            if (end[i] != 'X') s2 += end[i];
        if (s1 != s2) return false;
        for (int i = 0, j = 0; i < n && j < n;) {
            if (start[i] == 'X')
              i++;
            else if (end[j] == 'X')
              j++;
            else {
                if ((start[i] == 'L' && i < j) || (start[i] == 'R' && i > j)) return false;
                i++;
                j++;
            }
        }
        return true;
    }


// 779. K-th Symbol in Grammar
//  replace each occurrence of 0 with 01, and each occurrence of 1 with 10.
    int kthGrammar(int n, int k) {
        if(n==1 && k==1) return 0;
```

```
        int mid=pow(2,n-1)/2;

        if(k<=mid){
            return kthGrammar(n-1, k);
        }else{
            return !(kthGrammar(n-1, k-mid));
        }
    }


// 781. Rabbits in Forest
    int numRabbits(vector<int>& answers) {
        unordered_map<int, int> c;
        for (int i : answers) c[i]++;
        int res = 0;
        for (auto i : c) res += (i.second + i.first) / (i.first + 1) * (i.first + 1);
        return res;
    }


// 784. Letter Case Permutation
// Input: s = "a1b2"
// Output: ["a1b2","a1B2","A1b2","A1B2"]
    vector<string>ans;
    void solve(string curr,string s, int i){
        if(i==s.length()){
            ans.push_back(curr);
            return;
        }

        if(isdigit(s[i])){
            curr.push_back(s[i]);
            solve(curr,s,i+1);
        }
        else{
            string c1=curr;
            c1.push_back(tolower(s[i]));
            solve(c1,s,i+1);

            c1.pop_back();
            c1.push_back(toupper(s[i]));
            solve(c1,s,i+1);
            // string c2=curr;
            // c2.push_back(toupper(s[i]));
            // solve(c2,s,i+1);
        }
    }

    vector<string> letterCasePermutation(string S){
        ans.clear();
        solve("",S,0);
        return ans;
    }
```

```
// 788. Rotated Digits
// Input: n = 10
// Output: 4
// Explanation: There are four good numbers in the range [1, 10] : 2, 5, 6, 9.
    int rotatedDigits(int N) {
                int f[] = {1,1,2,0,0,2,2,0,1,2};
        int res = 0;
        for(int i = 1; i <= N; i++){
            int p = i;
            int s = 1;
            while(p){
                s *= f[p%10];
                p /= 10;
            }
            if(s >=2) res+=1;
        }
        return res;
    }

// 790. Domino and Tromino Tiling
    int MOD = pow(10, 9) + 7;
    int numTilings(int n) {
        if (n < 3) return n;
        vector<long> D(n+1, 0), T(n+1, 0);
        D[0] = 0, D[1] = 1, D[2] = 2;
        T[0] = 0, T[1] = 1, T[2] = 2;

        for (int i = 3; i <= n; i++) {
            D[i] = (D[i-1] + D[i-2] + 2*T[i-2]) % MOD;
            T[i] = (T[i-1] + D[i-1]) % MOD;
        }

        return D[n];
    }


// 794. Valid Tic-Tac-Toe State
    bool validTicTacToe(vector<string>& board) {
        int countX=0, countO=0;
        for (int i =0; i < 3; ++i){
            for (int j = 0; j < 3; ++j){
                if (board[i][j] == 'X') countX++;
                else if (board[i][j] == 'O') countO++;
            }
        }

        if (countX - countO >= 2 || countX - countO < 0) return false;

        cout << check(board, 'X') << endl;
        cout << check(board, 'O') << endl;

        if (check(board, 'X') > 0 && check(board, 'O') > 0) return false;

        if (countX == countO && check(board, 'X') > 0) return false;
```

```
        if (countX == countO + 1 && check(board, 'O') > 0) return false;

        return true;
    }


    int check(vector<string>& board, char ck) {

        int count = 0;
        for (int i = 0; i < 3; ++i){
            count = count + (board[i][0] == board[i][1] && board[i][0] == board[i][2] && board[i][0] == ck);
        }

        for (int i = 0; i < 3; ++i){
            count += (board[0][i] == board[1][i] && board[0][i] == board[2][i] && board[0][i] == ck );
        }

        count += (board[2][0] == board[1][1] && board[2][0] == board[0][2] && board[2][0] == ck);

        count += (board[0][0] == board[1][1] && board[0][0] == board[2][2] && board[0][0] == ck);

        return count;
    }


// 795. Number of Subarrays with Bounded Maximum
// Input: nums = [2,1,4,3], left = 2, right = 3
// Output: 3
// Explanation: There are three subarrays that meet the requirements: [2], [2, 1], [3].
    int numSubarrayBoundedMax(vector<int>& A, int L, int R) {
        int result=0, left=-1, right=-1;
        for (int i=0; i<A.size(); i++) {
            if (A[i]>R) left=i;
            if (A[i]>=L) right=i;
            result+=right-left;
        }
        return result;
    }


// 797. All Paths From Source to Target
int target;
vector<vector<int>> res;
vector<int> tmp;
void dfs(vector<vector<int>>& graph, int currNode = 0) {
    tmp.push_back(currNode);
    if (currNode == target)
        res.push_back(tmp);
    else
        for (int node: graph[currNode]) {
        dfs(graph, node);
    }
    tmp.pop_back();
}
```

```
vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
    target = graph.size() - 1;
    dfs(graph);
    return res;
}
    bool vis(int v, vector<int> &path)
    {
        for(auto i: path)
            if(i == v)
                return true;
        return false;
    }

    vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
        int src = 0, des = graph.size()-1;
        vector<vector<int>> res;
        vector<int> path;
        path.push_back(src);
        queue<vector<int>> q;
        q.push(path);
        while(!q.empty())
        {
            path = q.front();
            q.pop();
            int last_val = path.back();
            if(last_val == des)
                res.push_back(path);
            for(auto v: graph[last_val])
            {
                if(!vis(v, path))
                {
                    vector<int> temp(path);
                    temp.push_back(v);
                    q.push(temp);
                }
            }
        }
        return res;
    }
```

```
// 799. Champagne Tower
double champagneTower(int p, int r, int g) {
    double dp[101][101] = {};
    dp[0][0] = p;
    for (int y = 1; y <= r; y++) {
        for (int x = 0; x <= y; x++) {
            dp[y][x] = (x == y ? 0 : max(0.0, dp[y - 1][x] - 1) / 2.0) + (x ? max(0.0, dp[y - 1][x - 1] - 1) / 2.0 : 0);
        }
    }
    return min(1.0, dp[r][g]);
}
```

```
// 807. Max Increase to Keep City Skyline
    int maxIncreaseKeepingSkyline(vector<vector<int>>& grid) {
            int n = grid.size(), res = 0;
        vector<int> rows(n), cols(n);

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                rows[i] = max(rows[i], grid[i][j]);
                cols[j] = max(cols[j], grid[i][j]);
            }
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                res += min(rows[i], cols[j]) - grid[i][j];
            }
        }

        return res;
    }


// 811. Subdomain Visit Count
    vector<string> subdomainVisits(vector<string>& cpdomains) {
        unordered_map<string, int> count;
        for (auto cd : cpdomains) {
            int i = cd.find(" ");
            int n = stoi(cd.substr (0, i));
            string s = cd.substr (i + 1);
            for (int i = 0; i < s.size(); ++i)
                if (s[i] == '.')
                    count[s.substr(i + 1)] += n;
            count[s] += n;
        }
        vector<string> res;
        for (auto k : count)
            res.push_back (to_string(k.second) + " " + k.first);
        return res;
    }

// 814. Binary Tree Pruning
    TreeNode* pruneTree(TreeNode* root) {
        if (!root) return NULL;
        root->left = pruneTree(root->left);
        root->right = pruneTree(root->right);
        if (!root->left && !root->right && root->val == 0) return NULL;
        return root;
    }


// 820. Short Encoding of Words
   int minimumLengthEncoding(vector<string>& words) {
        unordered_set<string> s(words.begin(), words.end());
        for (string w : s)
```

```
        for (int i = 1; i < w.size(); ++i)
            s.erase(w.substr(i));
        int res = 0;
        for (string w : s) res += w.size() + 1;
        return res;
    }
```

```
// 822. Card Flipping Game
    int flipgame(vector<int>& f, vector<int>& b) {
        unordered_set<int> same;
        for (int i = 0; i < f.size(); ++i)
            if (f[i] == b[i])
                same.insert(f[i]);
        int res = 3000;
        for (int & i : f) if (same.count(i) == 0) res = min(res, i);
        for (int & i : b) if (same.count(i) == 0) res = min(res, i);
        return res % 3000;
    }
```

```
// 825. Friends Of Appropriate Ages
// age[y] <= 0.5 * age[x] + 7
// age[y] > age[x]
// age[y] > 100 && age[x] < 100
    unordered_map <int,int> map;
    int findRequests (vector<int>&ages,int index) {
        if (map.find(ages[index]) != map.end()) {
            return map[ages[index]];
        }
        int left = 0;
        int right = index-1;
        double target = (double) (0.5*ages[index]) + 7;
        while (left <= right) {
            int mid = left + (right-left)/2;
            if (ages[mid] <= target) {
                left = mid+1;
            } else {
                right = mid-1;
            }
        }

        map[ages[index]] = index-left;
        return index-left; // len between index-1 and left.
    }
    int numFriendRequests(vector<int>& ages) {
        sort(ages.begin(), ages.end());
        int count = 0;
        for (int i = ages.size()-1; i >= 0 ; i--) {
            count+=findRequests(ages, i);
        }
        return count;
    }
```

```
// 826. Most Profit Assigning Work
```

```
    int maxProfitAssignment(vector<int>& difficulty, vector<int>& profit, vector<int>& worker) {
        vector<pair<int, int>> jobs;
        int N = profit.size(), res = 0, i = 0, best = 0;
        for (int j = 0; j < N; ++j)
            jobs.push_back(make_pair(difficulty[j], profit[j]));
        sort(jobs.begin(), jobs.end());
        sort(worker.begin(), worker.end());
        for (int & ability : worker) {
            while (i < N && ability >= jobs[i].first)
                best = max(jobs[i++].second, best);
            res += best;
        }
        return res;
    }



// 831. Masking Personal Information
    vector<string> country = {"", "+*-", "+**-", "+***-"};
    string maskPII(string S) {
        string res;
        int at = S.find("@");
        if (at != string::npos) {
            transform(S.begin(), S.end(), S.begin(), ::tolower);
            return S.substr(0, 1) + "*****" + S.substr(at - 1);
        }
        S = regex_replace(S, regex("[^0-9]"), "");
        return country[S.size() - 10] + "***-***-" + S.substr(S.size() - 4);
    }



// 833. Find And Replace in String
// Input: s = "abcd", indices = [0, 2], sources = ["a", "cd"], targets = ["eee", "ffff"]
// Output: "eeebffff"
    string findReplaceString(string s, vector<int>& in, vector<string>& src, vector<string>& tgt) {
        // string temp = s;
        int n=in.size();
        vector<pair<int,pair<string, string>>> v;
        for(int i=0;i<n;i++){
            v.push_back({in[i],{src[i],tgt[i]}});
        }
        sort(v.begin(),v.end());
        reverse(v.begin(),v.end());
        for(int i=0;i<n;i++){
            string temp;
            temp=s;
            int idx=v[i].first; string src1=v[i].second.first, tgt1=v[i].second.second;
            if(src1 == temp.substr(idx,src1.size())){
                string left=temp.substr(0,idx),right=temp.substr(idx+src1.size());
                temp=left+tgt1+right;

                cout<<temp<<" "<<idx<<" "<<src1<<" "<<tgt1<<" "<<left<<" "<<right<<"\n";
            }
            s=temp;
        }
        return s;
```

```
    }


// 838. Push Dominoes
string pushDominoes(string d) {
    d = 'L' + d + 'R';
    string res = "";
    for (int i = 0, j = 1; j < d.length(); ++j) {
        if (d[j] == '.') continue;
        int middle = j - i - 1;
        if (i > 0)
            res += d[i];
        if (d[i] == d[j])
            res += string(middle, d[i]);
        else if (d[i] == 'L' && d[j] == 'R')
            res += string(middle, '.');
        else
            res += string(middle / 2, 'R') + string(middle % 2, '.') + string(middle / 2, 'L');
        i = j;
    }
    return res;
}


// 841. Keys and Rooms
bool canVisitAllRooms(vector<vector<int>>& rooms) {
 int n = size(rooms);
 vector<bool>visited(n, false);
 dfs(rooms, visited, 0);
 for(auto v : visited)
        if(!v)
            return false;
 return true;
}
void dfs(vector<vector<int> >& rooms , vector<bool>& visited, int i){
 visited[i] = true;
 for(auto& room : rooms[i])
  if(!visited[room])
            dfs(rooms, visited, room);
}


// 842. Split Array into Fibonacci Sequence
    vector<int> ans;
    vector<int> splitIntoFibonacci(string S) {
        vector<int> x;
        fibonacciSeq(S, x, 0);
        return ans;
    }

    void fibonacciSeq(string S, vector<int> vec, int index) {
        if (index == S.length() && vec.size() > 2) {
            ans = vec;
            return;
        }
```

```
        long num = 0;
        for (int i = index; i < S.length(); i++) {
            num = num * 10 + S[i] - '0';
            if (num > INT_MAX) return;
            if (S[index] == '0' && i > index) return;
            if (vec.size() < 2 || num == ((long)vec.back() + (long)vec[vec.size()-2])) {
                vec.push_back(num);
                fibonacciSeq(S, vec, i + 1);
                vec.pop_back();
            }
        }
    }
```

```
// 845. Longest Mountain in Array
int longestMountain(vector<int>& A) {
    int N = A.size(), res = 0;
    vector<int> up(N, 0), down(N, 0);
    for (int i = N - 2; i >= 0; --i) if (A[i] > A[i + 1]) down[i] = down[i + 1] + 1; else down[i]=0;
    for (int i = 0; i < N; ++i) {
        if (i > 0 && A[i] > A[i - 1]) up[i] = up[i - 1] + 1;
        else up[i]=0;
        if (up[i] && down[i]) res = max(res, up[i] + down[i] + 1);
    }
    return res;
}
```

```
// 846. Hand of Straights
   bool isNStraightHand(vector<int>& nums, int k)
    {
        if(nums.size()%k!=0)
            return false;
        map<int,int> count;
        map<int,int>::iterator it;
        int freq;
        for(int &i:nums) count[i]++;

        for(it=count.begin();it!=count.end();it++)
            if(it->second)
            {
                freq=it->second;
                for(int i=0;i<k;i++)
                    if(count[it->first+i]<freq)
                        return false;
                    else
                        count[it->first+i]-=freq;
            }
        return true;
    }
```

```
// 848. Shifting Letters
    string shiftingLetters(string S, vector<int> sh) {
```

```
        for (int i = sh.size() - 1, m = 0; i >= 0; --i, m %= 26)
            S[i] = ((S[i] - 'a') + (m += sh[i])) % 26 + 'a';
        return S;
    }
```

```
// 849. Maximize Distance to Closest Person
    int maxDistToClosest(vector<int>& seats) {
        int res = 0, n = seats.size(), last = -1;
        for (int i = 0; i < n; ++i) {
            if (seats[i] == 1) {
                res = last < 0 ? i : max(res, (i - last) / 2);
                last = i;
            }
        }
        res = max(res, n - last - 1);
        return res;
    }
```

```
// 851. Loud and Rich
    vector<int> loudAndRich(vector<vector<int>>& richer, vector<int>& quiet) {
             int n = quiet.size();
        vector<int> indg(n,0), ans(n,INT_MAX);
        vector<vector<int>> g(n);
        queue<int> q;


        for(int i=0;i<richer.size();i++){
            g[richer[i][0]].push_back(richer[i][1]);
            indg[richer[i][1]]++;
        }


        for(int i=0;i<n;i++){
            ans[i]=i;
            if(indg[i]==0)
                q.push(i);
        }

        while(!q.empty()){
            int curr = q.front();
            q.pop();

            for(int i : g[curr]){

                if(quiet[ans[i]] > quiet[ans[curr]])
                    ans[i] = ans[curr];
                indg[i]--;
                if(indg[i]==0)
                    q.push(i);
            }
        }
        return ans;
    }
```

```
// 853. Car Fleet
int carFleet(int target, vector<int>& position, vector<int>& speed)
{
   if (position.empty() || speed.empty()) return 0;

   vector<pair<int, double>> intervals;
   for (int i = 0; i < position.size(); i++)
      intervals.push_back(make_pair(position[i], (double)(target - position[i])/(double)(speed[i])));
   sort(intervals.begin(), intervals.end());

   int fleetCnt = 0; pair<int,double> p;
   p = intervals[intervals.size()-1];
   for (int i = intervals.size()-2; i>=0; i--) {
      if (intervals[i].second > p.second) {
         fleetCnt++; p = intervals[i];
      }
   }
   return fleetCnt+1;
}


// 856. Score of Parentheses
// "()" has score 1.
// AB has score A + B, where A and B are balanced parentheses strings.
// (A) has score 2 * A, where A is a balanced parentheses string.
   int scoreOfParentheses(string S) {
      stack<int> stack;
      int cur = 0;
      for (char i : S)
         if (i == '(') {
            stack.push(cur);
            cur = 0;
         }
         else {
            cur += stack.top() + max(cur, 1);
            stack.pop();
         }
      return cur;
   }


// 861. Score After Flipping Matrix
   void flipRow(vector<vector<int>>& A,int row,int n){
      for(int j = 0;j < n;j++){
         if(A[row][j] == 0)
            A[row][j] = 1;
         else
            A[row][j] = 0;
      }
   }
   void flipCol(vector<vector<int>>& A,int col,int m){
      for(int i = 0;i < m;i++){
         if(A[i][col] == 0)
            A[i][col] = 1;
```

```
            else
                A[i][col] = 0;
        }
    }
    int matrixScore(vector<vector<int>>& A) {
        int m = A.size();
        int n = A[0].size();
        vector<int> col(n,0);

        for(int i = 0;i < m;i++){
            if(A[i][0] == 0)
                flipRow(A,i,n);
            for(int j = 0;j < n;j++){
                if(A[i][j] == 1)
                    col[j]++;
            }
        }

        for(int j = 0;j < n;j++){
            if(col[j] <= m/2)
                flipCol(A,j,m);
        }

        int result = 0,sum;
        for(vector<int> v : A){
            sum = 0;
            for(int j = v.size()-1;j >= 0 ;j--){
                if(v[j] == 1)
                    sum += pow(2,v.size()-1-j);
            }
            result += sum;
        }
        return result;
    }



// 863. All Nodes Distance K in Binary Tree
    vector<int> distanceK(TreeNode* root, TreeNode* target, int K) {
        unordered_map<TreeNode*, TreeNode*> parent_track; // node -> parent
        unordered_map<TreeNode*, bool> visited;
        queue<TreeNode*> queue;
        queue.push(root);
        while(!queue.empty()) {
            TreeNode* current = queue.front(); queue.pop();
            if(current->left) {
                parent_track[current->left] = current;
                queue.push(current->left);
            }
            if(current->right) {
                parent_track[current->right] = current;
                queue.push(current->right);
            }
        }
        queue.push(target);
```

```
        visited[target] = true;
        int curr_level = 0;
        while(!queue.empty()) {
            int size = queue.size();
            if(curr_level++ == K) break;
            for(int i=0; i<size; i++) {
                TreeNode* current = queue.front(); queue.pop();
                if(current->left && !visited[current->left]) {
                    queue.push(current->left);
                    visited[current->left] = true;
                }
                if(current->right && !visited[current->right]) {
                    queue.push(current->right);
                    visited[current->right] = true;
                }
                if(parent_track[current] && !visited[parent_track[current]]) {
                    queue.push(parent_track[current]);
                    visited[parent_track[current]] = true;
                }
            }
        }
        vector<int> result;
        while(!queue.empty()) {
            TreeNode* current = queue.front(); queue.pop();
            result.push_back(current->val);
        }
        return result;
    }


// 873. Length of Longest Fibonacci Subsequence

// 874. Walking Robot Simulation
    int robotSim(vector<int>& commands, vector<vector<int>>& obstacles) {
        unordered_set<string> uset;
        for(int i = 0;i<obstacles.size();i++){
            uset.insert(to_string(obstacles[i][0]) + "." + to_string(obstacles[i][1]));
        }

        int dir = 0, x = 0, y= 0, dx, dy, res = 0;
        int move[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        for(int i = 0;i<commands.size();i++){
            if(commands[i] == -1)dir++;
            else if(commands[i] == -2) dir--;
            else{

                for(int j = 0;j<commands[i];j++){
                    dx = x + move[dir][0];
                    dy = y + move[dir][1];
                    string key = to_string(dx) + "." + to_string(dy);
                    if(uset.count(key)){
                        break;
                    }
                    x = dx, y = dy;
                    res = max(res, x*x + y*y);
```

```
            }
        }
        dir = (dir + 4) % 4;
    }
    return res;
}
```

```
// 875. Koko Eating Bananas
    int minEatingSpeed(vector<int>& piles, int H) {
         int l = 1, r = 1000000000;
        while (l < r) {
            int m = (l + r) / 2, total = 0;
            for (int p : piles)
                total += (p + m - 1) / m;
            if (total > H)
                l = m + 1;
            else
                r = m;
        }
        return l;
    }
```

```
// 881. Boats to Save People
    int numRescueBoats(vector<int>& people, int limit) {
        sort(people.begin(),people.end());
        int i = 0, j = people.size() - 1,cnt = 0;
        while(i <= j){
            if(people[i] + people[j] <= limit)  ++i,--j;
            else --j;
            ++cnt;
        }
        return cnt;
    }
```

```
// 893. Groups of Special-Equivalent Strings
    int numSpecialEquivGroups(vector<string>& words) {
        map<pair<vector<int>,vector<int>>,int> m;
        for(int i=0;i<words.size();i++)
        {
            vector<int> even(26,0);
            vector<int> odd(26,0);
            for(int j=0;j<words[i].size();j++)
                if(j%2==0) even[words[i][j]-'a']++;
                else odd[words[i][j]-'a']++;
            m[{even,odd}]++;
        }
        return m.size();
    }
```

```
// 898. Bitwise ORs of Subarrays
    int subarrayBitwiseORs(vector<int>& A) {
        unordered_set<int> res, cur, cur2;
```

```
        for (int i: A) {
            cur2 = {i};
            for (int j: cur) cur2.insert(i|j);
            for (int j: cur = cur2) res.insert(j);
        }
        return res.size();
    }
```

```
// 901. Online Stock Span
class StockSpanner {
    vector<int> prices;
    vector<int> prevGreater;
    int index;
public:
    StockSpanner() {
        index = -1;
    }

    int next(int price) {
        prices.push_back(price);
        index++;

        if(index == 0) {
            prevGreater.push_back(-1);
        } else {
            if(prices[index - 1] > prices[index]) {
                prevGreater.push_back(index - 1);
            } else {
                int j = prevGreater[index - 1];
                while(j != -1 && prices[j] <= price) {
                    j = prevGreater[j];
                }
                prevGreater.push_back(j);
            }
        }

        return index - prevGreater[index];
    }
```

```
// 911. Online Election
    map<int, int> m;
    TopVotedCandidate(vector<int> persons, vector<int> times) {
        int n = persons.size(), lead = -1;
        unordered_map<int, int> count;
        for (int i = 0; i < n; ++i) {
            lead = ++count[persons[i]] >= count[lead] ? persons[i] : lead;
            m[times[i]] = lead;
        }
    }

    int q(int t) {
        return (--m.upper_bound(t))-> second;
    }
```

```
// 915. Partition Array into Disjoint Intervals
int partitionDisjoint(vector<int>& nums) {
    int n = nums.size();
    int max_1, curr, ans = 1;
    max_1 = curr = nums[0];
    for(int i = 1; i < n; i++){
        if(nums[i] < max_1){
            max_1 = curr;
            ans = i + 1;
        }
        else if(nums[i] > curr){
            curr = nums[i];
        }
    }
    return ans;
}


// 916. Word Subsets
vector<string> wordSubsets(vector<string>& A, vector<string>& B) {
    vector<int> count(26), tmp(26);
    int i;
    for (string b : B) {
        tmp = counter(b);
        for (i = 0; i < 26; ++i)
            count[i] = max(count[i], tmp[i]);
    }
    vector<string> res;
    for (string a : A) {
        tmp = counter(a);
        for (i = 0; i < 26; ++i)
            if (tmp[i] < count[i])
                break;
        if (i == 26) res.push_back(a);
    }
    return res;
}
vector<int> counter(string& word) {
    vector<int> count(26);
    for (char c : word) count[c - 'a']++;
    return count;
}


// 918. Maximum Sum Circular Subarray
    int maxSubarraySumCircular(vector<int>& nums) {
        int total = 0, maxSum = nums[0], curMax = 0, minSum = nums[0], curMin = 0;
        for (int& a : nums) {
            curMax = max(curMax + a, a);
            maxSum = max(maxSum, curMax);
            curMin = min(curMin + a, a);
            minSum = min(minSum, curMin);
            total += a;
```

```
        }
        return maxSum > 0 ? max(maxSum, total - minSum) : maxSum;
    }
```

```
// 923. 3Sum With Multiplicity
    int threeSumMulti(vector<int>& arr, int target) {
            unordered_map<int, int> m;
        int res = 0, mod = 1e9 + 7;
        for (int i = 0; i < arr.size(); i++) {
            res = (res + m[target - arr[i]]) % mod;
            for (int j = 0; j < i; j++) {
                m[arr[i] + arr[j]]++;
            }
        }
        return res;
    }
```

```
// 926. Flip String to Monotone Increasing
   int minFlipsMonoIncr(string s) {
            int flips = 0, counter = 0;
        for (auto c : s) {
            if (c == '1') counter++;
            else flips++;

            flips = min(flips, counter);
        }
        return flips;
    }
```

```
// 931. Minimum Falling Path Sum
int minFallingPathSum(vector<vector<int>>& A) {
    for (auto i = 1; i < A.size(); ++i)
        for (auto j = 0; j < A.size(); ++j)
            A[i][j] += min({ A[i-1][j], A[i-1][max(0,j-1)], A[i-1][min((int)A.size()-1,j+1)] });
    return *min_element(begin(A[A.size() - 1]), end(A[A.size() - 1]));
}
```

```
// 932. Beautiful Array
// nums is a permutation of the integers in the range [1, n].
// For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].
    vector<int> beautifulArray(int n) {
        vector<int> ans; ans.push_back(1) ;
        while(ans.size() < n){
            vector<int> temp ;
            for(auto i:ans){
                if(i*2 - 1 <= n) temp.push_back(i*2 - 1) ;
            }
            for(auto i:ans){
                if(i*2 <= n) temp.push_back(i*2) ;
            }
            ans = temp ;
```

```
    }
    return ans ;
  }
```

```
// 934. Shortest Bridge
  vector<vector<int> > mat;
  vector<vector<int> > vis;
  int m,n;
  int x[4]={-1,0,1,0};
  int y[4]={0,1,0,-1};
  queue<pair<int,int> > que;
  void dfs(int i,int j){
    vis[i][j]=1;
    que.push({i,j});
    for(int dir=0;dir<4;dir++){
      int xd=i+x[dir];
      int yd=j+y[dir];
      if(xd>=0 && yd>=0 && xd<=m-1 && yd<=n-1){
        if(!vis[xd][yd] && mat[xd][yd]==1){
          dfs(xd,yd);
        }
      }
    }
  }
  int shortestBridge(vector<vector<int>>& A) {
    m=A.size();
    if(m==0) return 0;
    n=A[0].size();
    cout<<m<<" "<<n;
    mat=A;
    vis.resize(m,vector<int>(n,0));
    int flag=0;
    for(int i=0;i<m;i++){
      for(int j=0;j<n;j++){
        if(mat[i][j]==1){
          dfs(i,j);
          flag=1;
          break;
        }
      }
      if(flag) break;
    }
    int l=0;
    while(!que.empty()){
      int len=que.size();
      l++;
      while(len--){
        pair<int,int> poi=que.front();
        que.pop();
        vis[poi.first][poi.second]=1;
        for(int dir=0;dir<4;dir++){
          int xd=poi.first+x[dir];
          int yd=poi.second+y[dir];
          if(xd>=0 && yd>=0 && xd<=m-1 && yd<=n-1){
```

```
                if(!vis[xd][yd] && mat[xd][yd]==1){
                    return l-1;
                }
                else if(!vis[xd][yd] && mat[xd][yd]==0){
                    vis[xd][yd]=1;
                    que.push({xd,yd});
                }
            }
        }
    }
  }
  return -1;
}
```

```
// 935. Knight Dialer
    const int MOD=1e9+7;
  int knightDialer(int n) {
        vector<vector<int>> paths= {{4, 6}, {6, 8}, {7, 9}, {4, 8}, {0, 3, 9}, {}, {0, 1, 7}, {2, 6}, {1, 3}, {2, 4}};
  vector<vector<long>> dp (n+1, vector<long>(10));

  for (int j = 0; j < 10; j++)
    dp[1][j] = 1;
  for (int i = 2; i < n + 1; i++) {
    for (int j = 0; j < 10; j++) {
      for (int p : paths[j]) {
        dp[i][j] += dp[i - 1][p];
      }
      dp[i][j] %= (int)MOD;
    }
  }
  long sum = 0;
  for (int j = 0; j < 10; j++)
    sum += dp[n][j];
  return (int) (sum % MOD);
  }
```

```
// 939. Minimum Area Rectangle
  int minAreaRect(vector<vector<int>>& points) {
    int ans = INT_MAX;
    map<pair<int,int>,bool>mp;
    for(int i=0;i<points.size();i++)
    {
      pair<int,int> current_coordinate = {points[i][0] , points[i][1]};
      mp[current_coordinate] = true;
    }
    for(int i=0;i<points.size();i++)
    {
      for(int j=i+1;j<points.size();j++)
      {
        int x1 = points[i][0];
        int x2 = points[j][0];
        int y1 = points[i][1];
        int y2 = points[j][1];
```

```
            if((x1 != x2) && (y1 != y2))
            {
                int x3 = x1;
                int y3 = y2;

                int x4 = x2;
                int y4 = y1;
                if((mp.find({x3,y3}) != mp.end())  &&  (mp.find({x4,y4}) != mp.end()))
                {
                    int current_area = abs(y3 - y4) * abs(x3 - x4);
                    ans = min(ans,current_area);
                }


            }
        }
    }
    if(ans == INT_MAX)
        return 0;
    return ans;

}
```

```
// 945. Minimum Increment to Make Array Unique
int minIncrementForUnique(vector<int>& A) {
    int s = A.size();
    int res=0;
    if (s<2)  return 0;
    sort(A.begin(),A.end());
    for (int i=1; i<s; ++i) {
        if (A[i]<=A[i-1]){
            res+=A[i-1]+1 -A[i];
            A[i]= A[i-1]+1;
        }
    }
    return res;
}
```

```
// 946. Validate Stack Sequences
 bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {
    stack<int> s ;   // an empty stack
    int j = 0;
    for(int i= 0; i<pushed.size(); ++i){
        s.push(pushed[i]);
        while(!s.empty() && s.top() == popped[j]) {
            s.pop();
            ++j;
        }
    }
    return s.empty();
}
```

```
// 948. Bag of Tokens
    int bagOfTokensScore(vector<int>& tokens, int P) {
        sort(tokens.begin(), tokens.end());
        int res = 0, points = 0, i = 0, j = tokens.size() - 1;
        while (i <= j) {
            if (P >= tokens[i]) {
                P -= tokens[i++];
                res = max(res, ++points);
            } else if (points > 0) {
                points--;
                P += tokens[j--];
            } else {
                break;
            }
        }
        return res;
    }

// 949. Largest Time for Given Digits
    string largestTimeFromDigits(vector<int>& a) {
        string ans = ""; int mx = -1, h1 = -1, h2 = -1, m1 = -1, m2 = -1;

        for(int i=0; i<4; ++i) {
            if(a[i] > 2) continue;

            for(int j=0; j<4; ++j) {
                if(j == i) continue;
                if(a[i] == 2 && a[j] > 3) continue;

                for(int k=0; k<4; ++k) {
                    if(k == j || k == i) continue;
                    if(a[k] > 5) continue;

                    int l = 6-i-j-k;
                    if(l == k || l == j || l == i) continue;

                    int val = (a[l] + (a[k] * 10)) + (a[j] + (a[i] * 10)) * 60; // value of time in minutes.

                    if(mx < val) {
                        mx = val;
                        h1 = a[i], h2 = a[j], m1 = a[k], m2 = a[l];
                    }
                }
            }
        }

        if(h1 == -1 || h2 == -1 || m1 == -1 || m2 == -1) return "";

        ans = to_string(h1) + to_string(h2) + ":" + to_string(m1) + to_string(m2);

        return ans;
    }
```

```
// 950. Reveal Cards In Increasing Order
// Take the top card of the deck, reveal it, and take it out of the deck.
// If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.
// If there are still unrevealed cards, go back to step 1. Otherwise, stop.
    vector<int> deckRevealedIncreasing(vector<int>& deck) {
            sort(deck.rbegin(), deck.rend());
        deque<int> d;
        d.push_back(deck[0]);
        for (int i = 1; i < deck.size(); i++) {
            d.push_front(d.back());
            d.pop_back();
            d.push_front(deck[i]);
        }
        vector<int> res(d.begin(), d.end());
        return res;
    }


// 951. Flip Equivalent Binary Trees
    bool flipEquiv(TreeNode* root1, TreeNode* root2) {
        if (!root1 && !root2) return true;
        if (!root1 && root2 || root1 && !root2 || root1->val!=root2->val ) return false;
        return flipEquiv( root1->left, root2->left ) && flipEquiv( root1->right, root2->right )
            || flipEquiv( root1->right, root2->left ) && flipEquiv( root1->left, root2->right );
    }


// 954. Array of Doubled Pairs
// if it is possible to reorder arr such that arr[2 * i + 1] = 2 * arr[2 * i]
    bool canReorderDoubled(vector<int>& A) {
        unordered_map<int, int> c;
        for (int a : A) c[a]++;
        vector<int> keys;
        for (auto it : c)
            keys.push_back(it.first);
        sort(keys.begin(), keys.end(), [](int i, int j) {return abs(i) < abs(j);});
        for (int x : keys) {
            if (c[x] > c[2 * x])
                return false;
            c[2 * x] -= c[x];
        }
        return true;
    }


// 955. Delete Columns to Make Sorted II
bool check(vector<string> str)
{
    for(int i=1;i<str.size();i++)
    {
        if(str[i]<str[i-1]) return false;
    }
    return true;
}
```

```
int minDeletionSize(vector<string>& A) {
   int count=0;int len=A[0].length();
   vector<string>full_sorted(A.size());
   for(int i=0;i<len;i++)
   {
      vector<string>temp=full_sorted;
      for(int k=0;k<A.size();k++) temp[k]+=A[k][i];
      if(check(temp)) full_sorted=temp;
      else count++;
   }
   return count;
}


// 957. Prison Cells After N Days
// If a cell has two adjacent neighbors that are both occupied or both vacant, then the cell becomes occupied.
// Otherwise, it becomes vacant
   vector<int> prisonAfterNDays(vector<int>& cells, int N) {
      N = N%14 == 0 ? 14 : N%14 ;
           for (N; N > 0; --N) {
         vector<int> cells2(8, 0);
         for (int i = 1; i < 7; ++i)
            cells2[i] = cells[i - 1] == cells[i + 1] ? 1 : 0;
         cells = cells2;
      }
      return cells;
   }


// 958. Check Completeness of a Binary Tree
   bool isCompleteTree(TreeNode* root) {
      vector<TreeNode*> bfs;
      bfs.push_back(root);
      int i = 0;
      while (i < bfs.size() && bfs[i]) {
         bfs.push_back(bfs[i]->left);
         bfs.push_back(bfs[i]->right);
         i++;
      }
      while (i < bfs.size() && !bfs[i])
         i++;
      return i == bfs.size();
   }


// 962. Maximum Width Ramp
// for which i < j and nums[i] <= nums[j]. The width of such a ramp is j - i.
   int maxWidthRamp(vector<int>& A) {
      stack<int> s;
      int res = 0, n = A.size();
      for (int i = 0; i < n; ++i)
         if (s.empty() || A[s.top()] > A[i])
            s.push(i);
```

```
        for (int i = n - 1; i > res; --i)
            while (!s.empty() && A[s.top()] <= A[i])
                res = max(res, i - s.top()), s.pop();
        return res;
    }
```

```
// 967. Numbers With Same Consecutive Differences
// Input: n = 3, k = 7
// Output: [181,292,707,818,929]
// Explanation: Note that 070 is not a valid number, because it has leading zeroes.
    set<int>ans;
    void f(string s, int k, int n){
        if(k==0){
            int nums=0;
            nums=stoi(s);
            cout<<s<<"\n";
            ans.insert(nums);
            return;
        }

        int back= s.back()-'0';
        if(back+n <=9){
            int i=back+n;
            f(s+to_string(i),k-1,n);
        }
        if(back-n>=0){
            int i=back-n;
            f(s+to_string(i),k-1,n);
        }
    }
    vector<int> numsSameConsecDiff(int k, int n) {
        for(int i=1;i<=9;i++)
            f(to_string(i),k-1,n);
        vector<int>finalAns;
        for(auto &v:ans)finalAns.push_back(v);
        return finalAns;
    }
```

```
// 973. K Closest Points to Origin
    vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
        vector<vector<int>> answer;
        vector<pair<double,vector<int>>> distances;
        for(auto i:points){
            double distance=sqrt(pow(i[0],2)+pow(i[1],2));
            distances.push_back(make_pair(distance,i));
        }
        sort(distances.begin(),distances.end());
        for(int i=0;i<k;i++)answer.push_back(distances[i].second);
        return answer;
    }
```

```
// 974. Subarray Sums Divisible by K
    int subarraysDivByK(vector<int>& nums, int k) {
        int count = 0, curr = 0;
        unordered_map<int, int> m = {{0, 1}};

        for (int i = 0; i < nums.size(); i++) {
            curr = (curr + nums[i] % k + k) % k;
            count += m[curr];
            m[curr]++;
        }

        return count;
    }


// 978. Longest Turbulent Subarray
// For i <= k < j:
// arr[k] > arr[k + 1] when k is odd, and
// arr[k] < arr[k + 1] when k is even.
// Or, for i <= k < j:
// arr[k] > arr[k + 1] when k is even, and
// arr[k] < arr[k + 1] when k is odd.
      int maxTurbulenceSize(vector<int>& A) {
        int increase = 1; int decrease = 1; int max_len = 1;
        for(int i=0; i < A.size()-1; i++){
            if(A[i] > A[i+1]){
                increase = decrease + 1;
                decrease = 1;
            }
            else if(A[i] < A[i+1]){
                decrease = increase + 1;
                increase = 1;
            }
            else{
                increase = 1;
                decrease = 1;
            }
            max_len = max(max_len, max(increase, decrease));
        }
        return max_len;
    }


// 979. Distribute Coins in Binary Tree
    int res = 0;
    int distributeCoins(TreeNode* root) {
        dfs(root);
        return res;
    }
    int dfs(TreeNode* root) {
        if (!root) return 0;
        int left = dfs(root->left), right = dfs(root->right);
        res += abs(left) + abs(right);
        return root->val + left + right - 1;
    }
```

```
    int move = 0;
    int rec(TreeNode* root){
        if(!root) return 0;
        int l = rec(root->left);
        int r = rec(root->right);
        int totalNeed = root->val - 1 + l + r;
        move+= abs(totalNeed);
        return totalNeed;
    }
    int distributeCoins(TreeNode* root) {
        rec(root);
        return move;
    }
```

```
// Input: days = [1,4,6,7,8,20], costs = [2,7,15]
// Output: 11
// Explanation: For example, here is one way to buy passes that lets you travel your travel plan:
// On day 1, you bought a 1-day pass for costs[0] = $2, which covered day 1.
// On day 3, you bought a 7-day pass for costs[1] = $7, which covered days 3, 4, ..., 9.
// On day 20, you bought a 1-day pass for costs[0] = $2, which covered day 20.
// In total, you spent $11 and covered all the days of your travel.
    int mincostTickets(vector<int>& days, vector<int>& costs) {
       vector<int> dp(366,0);
        int ans=0,j=0;
        for(int i = 1;i<=365;i++){
            if(j<days.size() && days[j]==i){
                int op1 = dp[i-1]+costs[0];
                int op7 = dp[max(i-7,0)]+costs[1];
                int op30 = dp[max(i-30,0)]+costs[2];
                dp[i] = min({op1,op7,op30});
                j++;
            }else{
                dp[i]=dp[i-1];
            }
        }
        return dp[365];
    }
```

```
// 984. String Without AAA or BBB
    string strWithout3a3b(int A, int B) {
        string res;
        while (A && B) {
            if (A > B) {
                res += "aab";
                A--;
            } else if (B > A) {
                res += "bba";
                B--;
            } else {
                res += "ab";
            }
            A--;
            B--;
```

```
    }
    while (A--) res += "a";
    while (B--) res += "b";
    return res;
}
```

```
// 985. Sum of Even Numbers After Queries
// For each query i, first, apply nums[indexi] = nums[indexi] + vali, then print the sum of the even values of nums
    vector<int> sumEvenAfterQueries(vector<int>& nums, vector<vector<int>>& queries) {
        int size = queries.size();
        int sum = 0;
        vector<int> answer;

        for(int num : nums)
            if(num % 2 == 0)
                sum += num;

        for(int i = 0; i < size; i++){
            int val = queries[i][0], index = queries[i][1];
            int valToPush = sum;
            if(nums[index] % 2 == 0)
                valToPush -= nums[index];
            nums[index] += val;
            if(nums[index] % 2 == 0)
                valToPush += nums[index];
            sum = valToPush;
            answer.push_back(valToPush);
        }
        return answer;
    }
```

```
// 986. Interval List Intersections
// Input: firstList = [[0,2],[5,10],[13,23],[24,25]], secondList = [[1,5],[8,12],[15,24],[25,26]]
// Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
    vector<vector<int>> intervalIntersection(vector<vector<int>>& A, vector<vector<int>>& B) {
        vector<vector<int>> res;
        for(int i = 0, j = 0; i < A.size() && j < B.size();) {
            int lo = max(A[i][0], B[j][0]), hi = min(A[i][1], B[j][1]);
            if(lo <= hi) res.push_back({lo, hi});
            if(hi == A[i][1]) i++;
            else j++;
        }
        return res;
    }
```

```
// 991. Broken Calculator
// multiply the number on display by 2, or
// subtract 1 from the number on display.
    int brokenCalc(int X, int Y) {
        int res = 0;
```

```
    while (Y > X) {
        if (Y % 2) Y++;
        else Y /= 2;
        res++;
    }
    return res + X - Y;
}
```

// 1003. Check If Word Is Valid After Substitutions
// Insert string "abc" into any position in t. More formally, t becomes tleft + "abc" + tright, where t == tleft + tright. Note that tleft and tright may be empty.

```
bool isValid(string S) {
    vector<char> stack;
    for (char c : S) {
        if (c == 'c') {
            int n = stack.size();
            if (n < 2 || stack[n - 1] != 'b' || stack[n - 2] != 'a') return false;
            stack.pop_back(), stack.pop_back();
        } else {
            stack.push_back(c);
        }
    }
    return stack.size() == 0;
}
```

// 1004. Max Consecutive Ones III

```
int longestOnes(vector<int>& nums, int k) {
    int ans=0, n = size(nums);
    for(int l=0, r=0; r < n; r++) {
        if(nums[r] == 0)
            if(k == 0)
                while(nums[l++] != 0);
            else k--;
        ans = max(ans, r - l + 1);
    }
    return ans;
}
```

// 1008. Construct Binary Search Tree from Preorder Traversal

```
int i = 0;
TreeNode* bstFromPreorder(vector<int>& preorder, int max_val = INT_MAX) {
    if (i == preorder.size() || preorder[i] > max_val) return NULL;

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

    root->left = bstFromPreorder(preorder, root->val);
    root->right = bstFromPreorder(preorder, max_val);

    return root;
}
```

```
// 1010. Pairs of Songs With Total Durations Divisible by 60
    int numPairsDivisibleBy60(vector<int>& time) {
        for(int i=0;i<time.size();i++){
            time[i] = time[i]%60;
        }
        unordered_map<int,int>m;
        int ans=0;
        for(int i=0;i<time.size();i++){
            if(time[i]==0) ans+=m[0];
            else ans+=m[60-time[i]];
            m[time[i]]++;
        }
        return ans;
    }


// 1011. Capacity To Ship Packages Within D Days
    int countDays(vector<int>& ws, int tot_cap, int cur_cap = 0, int res = 1) {
      for (auto w : ws) {
        cur_cap += w;
        if (cur_cap > tot_cap) ++res, cur_cap = w;
      }
      return res;
    }
    int shipWithinDays(vector<int>& ws, int D) {
      auto r = accumulate(begin(ws), end(ws), 0);
      auto l = max(r / D, *max_element(begin(ws), end(ws)));
      while (l < r) {
        auto m = (l + r) / 2;
        if (countDays(ws, m) <= D) r = m;
        else l = m + 1;
      }
      return l;
    }


// 1014. Best Sightseeing Pair
    int maxScoreSightseeingPair(vector<int>& a) {
        int n = a.size(), maxScore = 0;
        int maxLeft = a[0] + 0;
        for(int j = 1; j < n; j++) {
            maxScore = max(maxScore, maxLeft + a[j] - j);
            maxLeft = max(maxLeft, a[j] + j);
        }
        return maxScore;
    }


// 1016. Binary String With Substrings Representing 1 To N
// if the binary representation of all the integers in the range [1, n] are substrings of s
 bool helper(string &s,int num){
  string str;
   while(num){
    if(num & 1) // 1
   str+='1';
```

```
   else // 0
  str+='0';
   num>>=1; // Right Shift by 1
  }
 reverse(str.begin(),str.end());
 // cout<<str<<" ";
 return s.find(str)!=-1;
}
bool queryString(string s, int n) {
 for(int i=1;i<=n;i++){
  if(!helper(s,i)) return false;
 }
 return true;
}
```

```
// 1017. Convert to Base -2
    string base2(int N) {
        string res = "";
        while (N) {
            res = to_string(N & 1) + res;
            N = N >> 1;
        }
        return res == ""  ? "0" : res;
    }

    string baseNeg2(int N) {
        string res = "";
        while (N) {
            res = to_string(N & 1) + res;
            N = -(N >> 1);
        }
        return res == "" ? "0" : res;
    }
```

```
// 1019. Next Greater Node In Linked List
    vector<int> nextLargerNodes(ListNode* head) {
        vector<int> res, stack;
        for (ListNode* node = head; node; node = node->next) {
            while (stack.size() && res[stack.back()] < node->val) {
                res[stack.back()] = node->val;
                stack.pop_back();
            }
            stack.push_back(res.size());
            res.push_back(node->val);
        }
        for (int i: stack) res[i] = 0;
        return res;
    }
```

```
// 1023. Camelcase Matching
    bool isSubSeq(string &s, string &t) {
        int j = 0;
```

```
        for(char c:s) {
            if(c >= 'A' && c <= 'Z' && c != t[j]) return false;
            if(c == t[j]) j++;
        }
        return j == t.size();
    }

    vector<bool> camelMatch(vector<string>& queries, string pattern) {
        vector<bool> ans;

        for(auto &q: queries) {
            ans.push_back(isSubSeq(q, pattern));
        }

        return ans;
    }
```

```
// 1027. Longest Arithmetic Subsequence
    int longestArithSeqLength(vector<int>& A) {
                int res = 2, n = A.size();
        vector<vector<int>> dp(n, vector<int>(2000, 0));
        for (int i = 0; i < n; ++i)
            for (int j = i + 1; j < n; ++j)  {
                int d = A[j] - A[i] + 1000;
                dp[j][d] = max(2, dp[i][d] + 1);
                res = max(res, dp[j][d]);
            }
        return res;
    }
```

```
// 1029. Two City Scheduling
    int twoCitySchedCost(vector<vector<int>>& costs) {
        vector<int> diff;
        int mincost = 0;
        for(int i = 0; i<costs.size(); ++i){
            mincost += costs[i][0];
            diff.push_back(costs[i][1] - costs[i][0]);
        }
        sort(diff.begin(),diff.end());
        for(int i=0; i<costs.size()/2; ++i)
             mincost += diff[i];
        return  mincost;
    }
```

```
// 1034. Coloring A Border
    void dfs(vector<vector<int>>& g, int r, int c, int cl) {
  if (r < 0 || c < 0 || r >= g.size() || c >= g[r].size() || g[r][c] != cl) return;
  g[r][c] = -cl;
  dfs(g, r - 1, c, cl), dfs(g, r + 1, c, cl), dfs(g, r, c - 1, cl), dfs(g, r, c + 1, cl);
  if (r > 0 && r < g.size() - 1 && c > 0 && c < g[r].size() - 1 && cl == abs(g[r - 1][c]) &&
    cl == abs(g[r + 1][c]) && cl == abs(g[r][c - 1]) && cl == abs(g[r][c + 1]))
```

```
  g[r][c] = cl;
}
vector<vector<int>> colorBorder(vector<vector<int>>& grid, int r0, int c0, int color) {
 dfs(grid, r0, c0, grid[r0][c0]);
 for (auto i = 0; i < grid.size(); ++i)
  for (auto j = 0; j < grid[i].size(); ++j) grid[i][j] = grid[i][j] < 0 ? color : grid[i][j];
 return grid;
}


// 1035. Uncrossed Lines
  int BottomUpDP(vector<int>& A, vector<int>& B,int n,int m){
    vector<vector<int>>dp(n+1,vector<int>(m+1,0));
    for(int i=1;i<=n;i++)
      for(int j=1;j<=m;j++)
        A[i-1]==B[j-1]?dp[i][j] = max( dp[i][j], dp[i-1][j-1] + 1 ):dp[i][j]= max(dp[i][j-1],dp[i-1][j]);
    return dp[n][m];
  }
  int maxUncrossedLines(vector<int>& A, vector<int>& B) {
    int n = A.size(),m = B.size();
    return BottomUpDP(A,B,n,m);
  }

// 1041. Robot Bounded In Circle
  bool isRobotBounded(string instructions) {
    int x = 0, y = 0, i = 0;
    vector<vector<int>> d = {{0, 1}, {1, 0}, {0, -1}, { -1, 0}};
    for (char & ins : instructions)
      if (ins == 'R')
        i = (i + 1) % 4;
      else if (ins == 'L')
        i = (i + 3) % 4;
      else
        x += d[i][0], y += d[i][1];
    return x == 0 && y == 0 || i > 0;
  }


// 1042. Flower Planting With No Adjacent
  vector<int> gardenNoAdj(int n, vector<vector<int>>& paths) {
    vector <int> answer(n, 1);
    vector <vector <int>> graph(n);

    for (vector v: paths) {
      graph[v[0] - 1].push_back(v[1] - 1);
      graph[v[1] - 1].push_back(v[0] - 1);
    }

    for (int garden = 0; garden < n; garden++) {
      vector <bool> colors(4, false);
      for (int c: graph[garden]) {
        colors[answer[c]] = true;
      }

      for (int i = 4; i > 0; i--)
```

```
            if (!colors[i])
                answer[garden] = i;
        }

        return answer;
    }
```

```
// 1052. Grumpy Bookstore Owner
int maxSatisfied(vector<int>& customers, vector<int>& grumpy, int X) {

 int normal=0, win=0, maxwin=0;;
 for(int i=0; i<customers.size(); i++){
  normal+=(!grumpy[i])*customers[i];
  win+=customers[i]*grumpy[i];
  if(i>=X) win-=customers[i-X]*grumpy[i-X];
  maxwin=max(maxwin, win);
 }
 return normal+maxwin;
}
```

```
// 1054. Distant Barcodes
// Rearrange the barcodes so that no two adjacent barcodes are equal.
    vector<int> rearrangeBarcodes(vector<int>& b) {
        unordered_map<int,int> m;
        for(auto i:b) m[i]++;
        priority_queue<pair<int,int>> pq;
        for(auto i:m) pq.push({i.second,i.first});
        vector<int> v(b.size(),0);
        int i=0;
        while(!pq.empty()){
            pair<int,int> p=pq.top();
            pq.pop();
            while(p.first--){
                if(i>=b.size()) i=1;
                v[i]=p.second;
                i+=2;
            }
        }
        return v;
    }
```

```
// 1072. Flip Columns For Maximum Number of Equal Rows
 int maxEqualRowsAfterFlips(vector<vector<int>>& M) {
  unordered_map<string, int> map;

  for (auto& r : M) {
   string s(r.size(), 'T');
   for (int i = 1; i < r.size(); i++) {
    if (r[i] != r[0]) s[i] = 'F';
   }
   map[s]++;
```

```
 }

 int ans = 0;
 for (auto& p : map)
  ans = max(ans, p.second);

 return ans;
}
```

```
// 1073. Adding Two Negabinary Numbers
vector<int> addNegabinary(vector<int>& arr1, vector<int>& arr2) {
 reverse(arr1.begin(),arr1.end());
 reverse(arr2.begin(),arr2.end());
 int len(max(arr1.size(),arr2.size()));
 int carry=0;
 vector<int> ans;
 for(int i=0;i<len+2;i++){
  int cur1=i<arr1.size() ? arr1[i]:0;
  int cur2=i<arr2.size() ? arr2[i]:0;
  int sum=cur1+cur2+carry;
  int r=sum%(-2);
  carry=sum/(-2);
  if(r<0){
   carry++;
   r+=abs(-2);
  }
  ans.push_back(r);

 }
 while(ans.size()>1 && ans.back()==0){
  ans.pop_back();
 }
 reverse(ans.begin(),ans.end());
 return ans;
}
```

```
// 1079. Letter Tile Possibilities
// Return the number of possible non-empty sequences of letters you can make using the letters printed on those tiles.

    void backtrack(string tiles, int level, int &count){
        count++;
        for(int i=level; i<tiles.length(); i++){

            if(i!=level && tiles[i]==tiles[level])
                continue;
            swap(tiles[i], tiles[level]);
            backtrack(tiles, level+1, count);
        }
    }
    int numTilePossibilities(string tiles) {
        int count=-1;
        sort(tiles.begin(), tiles.end());
        backtrack(tiles, 0, count);
```

```
        return count;
    }


// 1091. Shortest Path in Binary Matrix
// All the visited cells of the path are 0.
    vector<int> x_points = {-1,-1,-1,0,0,1,1,1};
    vector<int> y_points = {-1,0,1,-1,1,-1,0,1};

    bool isValidPoint(int x, int y, int n, int m) {
        return x >= 0 && x <= n && y >= 0 && y <= m;
    }

    int shortestPathBinaryMatrix(vector<vector<int>>& grid) {
        int n = grid.size()-1, m = grid[0].size()-1;

        if (grid[0][0] || grid[n][m])
            return -1;

        queue<vector<int> > q;
        vector<int> curr;

        q.push({0, 0});
        grid[0][0] = 1;

        while (!q.empty() && !grid[n][m]) {
            curr = q.front();
            q.pop();

            for (int i = 0; i < 8; i++) {
                int x = curr[0] + x_points[i];
                int y = curr[1] + y_points[i];

                if (isValidPoint(x, y, n, m) && grid[x][y] == 0) {
                    grid[x][y] = grid[curr[0]][curr[1]] + 1;
                    q.push({x, y});
                }
            }
        }
        return grid[n][m] ? grid[n][m] : -1;
    }




// 1123. Lowest Common Ancestor of Deepest Leaves
    int getDepth(TreeNode* root) {
        if (!root)
            return 0;
        return max(getDepth(root->right), getDepth(root->left)) + 1;
    }

    TreeNode* lcaDeepestLeaves(TreeNode* root) {
        if (!root)  return NULL;
```

```
        int right_depth = getDepth(root->right);
        int left_depth = getDepth(root->left);

        if (right_depth == left_depth)
            return root;

        if (right_depth > left_depth)
            return lcaDeepestLeaves(root->right);

        else
            return lcaDeepestLeaves(root->left);
    }
```

```
// 1138. Alphabet Board Path
// board = ["abcde", "fghij", "klmno", "pqrst", "uvwxy", "z"]
    string alphabetBoardPath(string target) {
                unordered_map<char, pair<int,int>> mp;
        for(int i = 0; i < 26; ++i) {
            mp[i+'a'] = {i/5, i%5};
        }
        target = 'a' + target;
        string path;
        int dx = 0, dy = 0;
        for(int i = 1; i < target.size(); ++i) {
            auto cur = mp[target[i]];
            auto prev = mp[target[i-1]];
            dx = cur.first - prev.first;
            dy = cur.second - prev.second;
            if(dy < 0) path += string(-dy, 'L');
            if(dx < 0) path += string(-dx, 'U');
            if(dy > 0) path += string(dy, 'R');
            if(dx > 0) path += string(dx, 'D');
            path += '!';
        }
        return path;
    }
```

```
// 1144. Decrease Elements To Make Array Zigzag
     int movesToMakeZigzag(vector<int>& nums) {
        int n=nums.size();
        if(n<=1) return 0;
        const int MAX=numeric_limits<int>::max();
        int ans1=0,ans2=0;
        for(int i=0; i<n; i++){
            int left=(i-1>=0)?nums[i-1]:MAX;
            int right=(i+1<n)?nums[i+1]:MAX;
            int smallest = min(left, right);
            if(i%2 == 0)
                ans1 += max(0, nums[i]-smallest+1);
            else
                ans2 += max(0, nums[i]-smallest+1);
        }
```

```
        return min(ans1, ans2);
    }
```

// 1155. Number of Dice Rolls With Target Sum

```
 int numRollsToTarget(int d, int f, int target) {
        Integer[][] mem=new Integer[d+1][target+1];
        return numRollsToTarget(d,f,target,mem);
    }
    int numRollsToTarget(int d, int f, int target, Integer[][] mem) {
        if(d==0||target<0) return target==0?1:0;
        if(mem[d][target]!=null) return mem[d][target];
        int ways=0;
        for(int i=1;i<=f;i++) ways=(ways+numRollsToTarget(d-1,f,target-i,mem))%1000000007;
        return mem[d][target]=ways;
    }
 int numRollsToTarget(int d, int f, int target) {
        int[][] dp=new int[d+1][target+1];
        dp[0][0]=1;
        for(int i=1;i<=d;i++)
            for(int j=1;j<=target;j++)
                for(int k=1;k<=f;k++)
                    dp[i][j]=(dp[i][j]+(k<=j?dp[i-1][j-k]:0))%1000000007;
        return dp[d][target];
    }
```

// 1161. Maximum Level Sum of a Binary Tree

```
int maxLevelSum(TreeNode* root) {
        int current_max = INT_MIN;
        int current_max_level = 0;
        queue<TreeNode*> q;
        q.push(root);
        int level = 0;
        while(!q.empty()) {
            int current_sum = 0;
            ++level;
            for (int i = q.size(); i > 0; --i) {
                TreeNode* n = q.front();
                q.pop();
                current_sum += n->val;
                if (n->left != NULL)
                    q.push(n->left);
                if (n->right != NULL)
                    q.push(n->right);
            }
            if (current_sum > current_max) {
                current_max = current_sum;
                current_max_level = level;
            }
        }
        return current_max_level;
    }
```

// 1162. As Far from Land as Possible

```
int maxDistance(vector<vector<int>>& grid) {
 int locMax = 0;
 queue<pair<int, int>> q;
 for (int i = 0; i < grid.size(); ++i) {
  for (int j = 0; j < grid[i].size(); ++j) {
   if (grid[i][j] == 1) q.push({ i, j });
  }
 }

 while (!q.empty()) {
  pair<int, int> cur = q.front();
  q.pop();
  pair<int, int> dirs[4] = { {0, 1}, {0, -1}, {1, 0}, {-1, 0} };
  for (auto dir : dirs) {
   int nx = dir.first + cur.first;
   int ny = dir.second + cur.second;
   if (nx >= 0 && nx < grid.size() && ny >= 0 && ny < grid[0].size() && grid[nx][ny] == 0) {
    q.push({ nx, ny });
    grid[nx][ny] = grid[cur.first][cur.second] + 1;
    locMax = max(grid[nx][ny], locMax);
   }
  }
 }
 return (locMax > 0 ? locMax-1 : -1);
  }

// 1170. Compare Strings by Frequency of the Smallest Character
// For each query queries[i], count the number of words in words such that f(queries[i]) < f(W) for each W in words.
    int f(string word) {
        char c = *min_element(word.begin(), word.end());
        return count(word.begin(), word.end(), c);
    }

    vector<int> numSmallerByFrequency(vector<string>& queries, vector<string>& words) {
        vector<int> f_words, res;
        int n = words.size();

        for (auto word : words) f_words.push_back(f(word));
        sort(f_words.begin(), f_words.end());

        for (auto q : queries) {
            int idxFirstLarger = upper_bound(f_words.begin(), f_words.end(), f(q)) - f_words.begin();
            res.push_back(n - idxFirstLarger);
        }

        return res;
    }
```

1186. Maximum Subarray Sum with One Deletion

```
    int maximumSum(vector<int>& arr) {
        int n = arr.size() , mx = INT_MIN ;
        vector<vector<int>> dp(n+1,vector<int>(2,-1e9));
        for(int i = 1 ; i <= n ; ++i)
```

```
        {
            dp[i][0] = max(dp[i-1][0] + arr[i-1],arr[i-1]);
            dp[i][1] = max(dp[i-1][0] , dp[i-1][1] + arr[i-1]);
            mx = max({mx,dp[i][0],dp[i][1]});
        }
        return mx;
    }
```

```
// 1190. Reverse Substrings Between Each Pair of Parentheses
    string reverseParentheses(string s) {
        deque<string> st;
        string cur = "";
        for(int i=0;i<s.size();++i){
            if(s[i] == '('){
                st.push_back(cur);
                cur = "";
            }
            else if(s[i] == ')'){
                reverse(cur.begin(),cur.end());
                cur = st.back()+cur;
                st.pop_back();
            }else cur += s[i];
        }
        return cur;
    }
```

```
// 1191. K-Concatenation Maximum Sum
    int kadane(vector<int>&arr){
        int maxsum=arr[0];
        int currsum=arr[0];
        for(int i=1;i<arr.size();i++){
            currsum=max(currsum+arr[i],arr[i]);
            maxsum=max(maxsum,currsum);
        }
        return maxsum%1000000007;
    }
    int kadaneoftwo(vector<int>&arr){
        vector<int>arrx(arr.size()*2,0);
        for(int i=0;i<arr.size();i++){
            arrx.push_back(arr[i]);
        }
        for(int i=0;i<arr.size();i++){
            arrx.push_back(arr[i]);
        }
        int maxsum=kadane(arrx);
        return maxsum%1000000007;
    }

    int kConcatenationMaxSum(vector<int>& arr, int k) {
        if(arr.size()==0)
            return 0;
        long long sum=accumulate(arr.begin(),arr.end(),0)%1000000007;
        if(k==1)
```

```
        return kadane(arr);
    if(sum%1000000007<0)
        return kadaneoftwo(arr);
    if(sum%1000000007>=0)
        return kadaneoftwo(arr)+(((k-2)*sum)%1000000007)%1000000007;
    return 0;
}
```

```
// 1208. Get Equal Substrings Within Budget
int equalSubstring(string s, string t, int maxCost) {
    int n=s.size(),arr[n];
    for(int i=0;i<n;i++)
        arr[i]=abs(s[i]-t[i]);
    int cost=0,start=0,maxlen=INT_MIN;
    for(int i=0;i<n;i++){
        cost+=arr[i];
        while(cost>maxCost)
            cost-=arr[start++];
        maxlen=max(maxlen,i-start+1);
    }
    return maxlen;
}
```

```
// 1209. Remove All Adjacent Duplicates in String II
string removeDuplicates(string s, int k) {
    int n = s.size();
    if(n<k) return s;

    stack<pair<char,int>> stk;
    for(int i=0; i<n; ++i){
        if(stk.empty() || stk.top().first != s[i]) stk.push({s[i],1});
        else{
            auto prev = stk.top();
            stk.pop();
            stk.push({s[i], prev.second+1});
        }
        if(stk.top().second==k) stk.pop();
    }

    string ans = "";
    while(!stk.empty()){
        auto cur = stk.top();
        stk.pop();
        while(cur.second--){
            ans.push_back(cur.first);
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

```
// 1218. Longest Arithmetic Subsequence of Given Difference
```

```
int longestSubsequence(vector<int>& arr, int diff) {
 unordered_map<int,int> mp;
 int ans=0;
 for(int i=0;i<arr.size();i++){
   if(mp.find(arr[i]-diff)!=mp.end()){
    int curr=mp[arr[i]-diff]+1;
    mp[arr[i]]=curr;
    ans=max(ans,curr);
   }
  else{
   mp[arr[i]]=1;
  }
 }
 return max(ans,1);
}
```

```
// 1219. Path with Maximum Gold
    int getMaximumGold(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        int maxGold = 0;
        for (int r = 0; r < m; r++)
            for (int c = 0; c < n; c++)
                maxGold = max(maxGold, findMaxGold(grid, m, n, r, c));
        return maxGold;
    }

    int DIR[5] = {0, 1, 0, -1, 0};
    int findMaxGold(vector<vector<int>>& grid, int m, int n, int r, int c) {
        if (r < 0 || r == m || c < 0 || c == n || grid[r][c] == 0) return 0;
        int origin = grid[r][c];
        grid[r][c] = 0; // mark as visited
        int maxGold = 0;
        for (int i = 0; i < 4; i++)
            maxGold = max(maxGold, findMaxGold(grid, m, n, DIR[i] + r, DIR[i + 1] + c));
        grid[r][c] = origin; // backtrack
        return maxGold + origin;
    }
```

```
// 1222. Queens That Can Attack the King
 vector<vector<int>> queensAttacktheKing(vector<vector<int>>& queens, vector<int>& king) {
        vector<vector<int>>ans;
        vector<vector<int>>seen(8,vector<int>(8,0));
        for(auto queen:queens)
            seen[queen[0]][queen[1]]=1;

        for(int dx=-1;dx<=1;dx++){
            for(int dy=-1;dy<=1;dy++){
                if(dx==0 && dy==0) continue;
                int x=king[0], y=king[1];
                while(x+dx>=0 && y+dy>=0 && x+dx<8 && y+dy<8){
                    x+=dx;
                    y+=dy;
                    if(seen[x][y]){
```

```
                ans.push_back({x,y});
                break;
            }
        }
    }
}
return ans;
}
```

```
// 1233. Remove Sub-Folders from the Filesystem
    vector<string> removeSubfolders(vector<string>& folder) {
    sort(folder.begin(), folder.end());
    vector<string> res;

    for (auto s : folder) {
        if (res.size() == 0) {
            res.push_back(s);
            continue;
        }

        string parent = res[res.size()-1];
        if (s.substr(0, parent.size()+1) != parent+'/')
            res.push_back(s);
    }

    return res;
}
```

```
// 1234. Replace the Substring for Balanced String
int balancedString(string s) {
    unordered_map<int, int> count;
    int n = s.length(), res = n, i = 0, k = n / 4;
    for (int j = 0; j < n; ++j) count[s[j]]++;

    for (int j = 0; j < n; ++j) {
        count[s[j]]--;
        while (i < n && count['Q'] <= k && count['W'] <= k && count['E'] <= k && count['R'] <= k) {
            res = min(res, j - i + 1);
            count[s[i++]] += 1;
        }
    }
    return res;
}
```

```
// 1238. Circular Permutation in Binary Representation
// p[0] = start
// p[i] and p[i+1] differ by only one bit in their binary representation.
// p[0] and p[2^n -1] must also differ by only one bit in their binary representation.
vector<int> circularPermutation(int n, int start) {
    vector<int> res;
    for (int i = 0; i < 1 << n; ++i)
```

```
        res.push_back(start ^ i ^ i >> 1);
    return res;
}
```

// 1239. Maximum Length of a Concatenated String with Unique Characters

```
int len{0};
int maxLength(vector<string>& arr) {
    checkLen("", arr, 0);
    return len;
}

void checkLen(string str, vector<string> &arr, int itr) {
    if (!isUnique(str)) return;
    if (str.size() > len) len = str.size();
    for (int i = itr; i < arr.size(); i++)
        checkLen(str+arr[i], arr, i+1);
}

bool isUnique(string word) {
    set<char> st;
    for (auto ele : word) {
        if (st.find(ele) != st.end()) return false;
        st.insert(ele);
    }
    return true;
}
```

// 1247. Minimum Swaps to Make Strings Equal

```
 int minimumSwap(string s1, string s2) {
    int x=0, y=0;
    for(int i=0;i<s1.length(); ++i)
        if(s1[i]!=s2[i])
            (s1[i]=='x') ? x++ : y++;
    if((x+y)%2)  return -1;
    int ans=(x+y)/2;
    if (x%2)  ans+=1;
    return ans;
}
```

// 1248. Count Number of Nice Subarrays k odd numbers in it

```
int numberOfSubarrays(vector<int>& nums, int k) {
    unordered_map<int, int> m;
    const int n = nums.size();
    int rst = 0;
    int acc = 0;
    m[0] = 1;
    for (int i = 0; i < n; ++i) {
        acc += (nums[i]%2);
        rst += m[acc-k];
        m[acc]++;
    }
    return rst;
```

```
    }
};


// 1249. Minimum Remove to Make Valid Parentheses
// Input: s = "lee(t(c)o)de)"
// Output: "lee(t(c)o)de"
// Explanation: "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.
    string minRemoveToMakeValid(string s) {
        stack<int>st;
        for(int i=0;i<s.length();++i){
            if(s[i]=='(')
                st.push(i);
            else if(s[i]==')')
                if(st.empty())
                    s[i]='#';
                else
                    st.pop();
        }
        while(!st.empty()){
            s[st.top()]='#';
            st.pop();
        }

        string ans="";
        for(int i=0;i<s.length();++i){
            if(s[i]!='#')
                ans.push_back(s[i]);
        }

        return ans;
    }


// 1254. Number of Closed Islands
    void DFS(vector<vector<int>>& board, int x, int y, int c) {
        if (x < 0 || x >= board.size() || y < 0 || y >= board[0].size() || board[x][y] != 0) return;

        board[x][y] = c;

        DFS(board, x + 1, y, c);
        DFS(board, x - 1, y, c);
        DFS(board, x, y + 1, c);
        DFS(board, x, y - 1, c);
    }
    int closedIsland(vector<vector<int>>& board) {
        int n = board.size(), m = board[0].size();
        for (int i = 0; i < n; i++) {
            if (board[i][0] == 0) DFS(board, i, 0, 1);
            if (board[i][m-1] == 0) DFS(board, i, m-1, 1);
        }
        for (int i = 0; i < m; i++) {
            if (board[0][i] == 0) DFS(board, 0, i, 1);
            if (board[n-1][i] == 0) DFS(board, n-1, i, 1);
        }
```

```
        int ans=0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (board[i][j] == 0) DFS(board, i, j, 1),ans++;
            }
        }
        return ans;
    }
```

```
// 1262. Greatest Sum Divisible by Three
    int maxSumDivThree(vector<int>& vec) {
       vector<vector<int>> dp(vec.size() + 1, vector<int>(3));

       dp[0][0] = 0;
       dp[0][1] = INT_MIN;
       dp[0][2] = INT_MIN;
       for (unsigned int i = 1; i <= vec.size(); i++)
       {
           int ind = i-1;
           dp[i][0] = max(dp[i-1][0], dp[i-1][(vec[ind]) % 3] + vec[ind]);
           dp[i][1] = max(dp[i-1][1], dp[i-1][(vec[ind]+1) % 3] + vec[ind]);
           dp[i][2] = max(dp[i-1][2], dp[i-1][(vec[ind]+2) % 3] + vec[ind]);
       }

       return dp[vec.size()][0];
    }
```

```
// 1267. Count Servers that Communicate
    int countServers(vector<vector<int>>& grid)
    {
        vector<int> rows(grid.size(),0),columns(grid[0].size(),0);
        for(int i=0;i<grid.size();i++)
            for(int j=0;j<grid[i].size();j++)
                if(grid[i][j])
                    rows[i]++,columns[j]++;
        int result=0;
        for(int i=0;i<grid.size();i++)
            for(int j=0;j<grid[i].size();j++)
                if(grid[i][j]&&(columns[j]>1||rows[i]>1))
                    result++;
        return result;
    }
```

```
// 1277. Count Square Submatrices with All Ones
    int countSquares(vector<vector<int>>& matrix) {
        int result = 0;
        for(int i = 0; i<matrix.size(); i++){
            for(int j = 0; j<matrix[0].size(); j++){
                if(i > 0 && j > 0 && matrix[i][j]>0)
                    matrix[i][j] = min(matrix[i-1][j-1], min(matrix[i-1][j], matrix[i][j-1])) + 1;
                result += matrix[i][j];
            }
        }
```

```
        return result;
    }

// 1282. Group the People Given the Group Size They Belong To
// Input: groupSizes = [3,3,3,3,3,1,3]
// Output: [[5],[0,1,2],[3,4,6]]
    vector<vector<int>> groupThePeople(vector<int>& groupSizes) {
        vector<vector<int>> ans{};
        unordered_map<int, vector<int>> dict{}; // K: group size, V: indices
        for (int i=0; i<groupSizes.size(); i++) {
            int key = groupSizes[i];
            if (dict.count(key) > 0) { // check existing groups to fill
                dict[key].push_back(i);
            } else { // create a new group
                dict[key] = vector<int>{i};
            }
            if (dict[key].size() == key) { // group is full
                ans.push_back(dict[key]);
                dict.erase(key);
            }
        }
        return ans;
    }


// 1288. Remove Covered Intervals
    int removeCoveredIntervals(vector<vector<int>>& intervals) {
        sort(intervals.begin(),intervals.end());
        int x1 = intervals[0][0];
        int x2 = intervals[0][1];
  int res = 1;

        for(int i= 1; i<intervals.size(); ++i){
            if(intervals[i][0] > x1 && intervals[i][1] > x2)
                ++res;
            if(intervals[i][1] > x2)
            {
                x1 = intervals[i][0];
                x2 = intervals [i][1];
            }
        }
        return res;
    }


// 1291. Sequential Digits
// An integer has sequential digits if and only if each digit in the number is one more than the previous digit.
// Input: low = 100, high = 300
// Output: [123,234]
    vector<int> ans;
    void dfs(int low, int high, int i, int num){
        if (num >= low and num <= high)
            ans.push_back(num);
        if (num > high or i>9)
            return;
```

```
        dfs(low, high, i+1, num*10 + i);
    }

    vector<int> sequentialDigits(int low, int high) {
        for(int i=1; i<=9; i++)
            dfs(low, high, i, 0);
        sort(ans.begin(), ans.end());
        return ans;
    }
```

```
// 1296. Divide Array in Sets of K Consecutive Numbers
   bool isPossibleDivide(vector<int>& nums, int k)
    {
        if(nums.size()%k!=0)
            return false;
        map<int,int> count;
        map<int,int>::iterator it;
        int freq;
        for(int &i:nums)  count[i]++;
        for(it=count.begin();it!=count.end();it++)
            if(it->second)
            {
                freq=it->second;
                for(int i=0;i<k;i++)
                    if(count[it->first+i]<freq)
                        return false;
                    else
                        count[it->first+i]-=freq;
            }
        return true;
    }
```

```
// 1297. Maximum Number of Occurrences of a Substring
   unordered_map<int, int> char_count;
    map<string, int> substr_count;

    int maxFreq(string s, int maxLetters, int minSize, int maxSize) {
        if(minSize > s.length())  return 0;
        int result = 0;
        for(int i = 0; i < minSize; i++) {
            char_count[s[i]]++;
        }
        if(char_count.size() <= maxLetters) {
            substr_count[s.substr(0, minSize)]++;
            result = max(result, substr_count[s.substr(0, minSize)]);
        }
        for(int right = minSize; right < s.length(); right++) {
            char_count[s[right - minSize]]--;
            if(char_count[s[right - minSize]] == 0)
                char_count.erase(s[right - minSize]);
            char_count[s[right]]++;
```

```
        if(char_count.size() <= maxLetters) {
            substr_count[s.substr(right - minSize + 1, minSize)]++;
            result = max(result, substr_count[s.substr(right - minSize + 1, minSize)]);
        }
    }
    return result;
}
```

// 1300. Sum of Mutated Array Closest to Target
// return the integer value such that when we change all the integers larger than value in the given array to be equal to value, the sum of the array gets as close as possible (in absolute difference) to target.

```
int diff(int mid, vector<int>& arr, int target)
{
    int sum = 0;
    for(int i = 0; i < arr.size(); i++)
        sum += min(mid, arr[i]);
    return abs(target - sum);
}

int findBestValue(vector<int>& arr, int target)
{
    int n = arr.size();
    int s = 0;
    int e = target;

    while(s < e)
    {
        int mid = s + (e - s)/2;
        if(diff(mid, arr, target) <= diff(mid + 1, arr, target))
            e = mid;
        else
            s = mid + 1;
    }

    return s;


}
```

// 1302. Deepest Leaves Sum

```
int getDepth(TreeNode* root) {
    if (!root) return 0;
    return max(getDepth(root->left), getDepth(root->right)) + 1;
}

int getSumForHeight(TreeNode* root, int h, int curr_h) {
    if (!root) return 0;
    if (curr_h == h-1) return root->val;
    return (getSumForHeight(root->left, h, curr_h + 1) + getSumForHeight(root->right, h, curr_h + 1));
}

int deepestLeavesSum(TreeNode* root) {
```

```
        int depth = getDepth(root);
        return getSumForHeight(root, depth, 0);
    }
```

```
// 1305. All Elements in Two Binary Search Trees
    vector<int> ans;
    void inorder(TreeNode* root){
        if(!root)
            return;
        inorder(root->left);
        ans.push_back(root->val);
        inorder(root->right);
    }
    vector<int> getAllElements(TreeNode* root1, TreeNode* root2) {
        inorder(root1);
        inorder(root2);
        sort(ans.begin(), ans.end());
        return ans;
    }
```

```
// 1310. XOR Queries of a Subarray
    vector<int> xorQueries(vector<int>& A, vector<vector<int>>& queries) {
        vector<int> res;
        for (int i = 1; i < A.size(); ++i)
            A[i] ^= A[i - 1];
        for (auto &q: queries)
            res.push_back(q[0] > 0 ? A[q[0] - 1] ^ A[q[1]] : A[q[1]]);
        return res;
    }
```

```
// 1314. Matrix Block Sum
    vector<vector<int>> matrixBlockSum(vector<vector<int>>& mat, int K) {
        vector<vector<int>> prefix_sum = prefixSum(mat);
        vector<vector<int>> answer = prefix_sum;

        for (int i = 0; i < mat.size(); i++) {
            for (int j = 0; j < mat[0].size(); j++) {
                int upper_i = ((i+K) >= mat.size()) ? (mat.size()-1) : (i+K);
                int upper_j = ((j+K) >= mat[0].size()) ? (mat[0].size()-1) : (j+K);

                int lower_i = ((i-K) <= 0 ) ? 0 : (i-K);
                int lower_j = ((j-K) <= 0 ) ? 0 : (j-K);


                answer[i][j] = prefix_sum[upper_i][upper_j] - ((lower_i == 0) ? 0 : prefix_sum[lower_i-1][upper_j]) - ((lo
wer_j == 0) ? 0 : prefix_sum[upper_i][lower_j-1]) + ((lower_i == 0) || (lower_j == 0)  ? 0 : prefix_sum[lower_i-1][lo
wer_j-1]);
            }
        }


        return answer;
    }
```

```
    vector<vector<int>> prefixSum(vector<vector<int>>& mat) {
        vector<vector<int>> prefix_sum = mat;
        for (int i = 0; i < mat.size(); i++) {
            int sum = 0;
            for (int j = 0; j < mat[0].size(); j++) {
                sum = sum + mat[i][j];
                if (i > 0) {
                    prefix_sum[i][j] = sum + prefix_sum[i-1][j];
                } else {
                    prefix_sum[i][j] = sum;
                }
            }
        }
        return prefix_sum;
    }


// 1315. Sum of Nodes with Even-Valued Grandparent
    int sumEvenGrandparent(TreeNode* root, int p = 1, int gp = 1) {
        return root ? sumEvenGrandparent(root->left, root->val, p)
               + sumEvenGrandparent(root->right, root->val, p)
               + (gp % 2 ? 0 : root->val)  : 0;
    }


// 1318. Minimum Flips to Make a OR b Equal to c
    int minFlips(int a, int b, int c) {
        int ai,bi,ci;
        int res=0;
        while(a>0 || b>0 || c>0){
            ai=a%2;bi=b%2;ci=c%2;
            if(ci==1 && ai+bi==0)
                res++;
            else if(ci==0)
                res+=ai+bi;
            a/=2;b/=2;c/=2;
        }
        return res;
    }

// 1324. Print Words Vertically
    vector<string> printVertically(string str)
    {
        int col=1,row=0,x;
        string temp;
        vector<string>s;
        stringstream iss(str);
        while (iss >> temp)
        {
            s.push_back(temp);
            col++;
            x=temp.size();
            row=max(row,x);
        }
```

```
        //created a 2d matrix
        int n=max(row,col);
        string a[n][n];
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
                a[i][j]=' ';
        }

        int k,j; col=0;
        for(auto ti: s)
        {
            k=0,j=col;
            for(auto it : ti)
            {
                a[k][j]=it;
                k++;
            }
            col++;//next column for next string.
        }

        vector<string>res;
        for(int i=0;i<row;i++)
        {
            temp="";
            for(int j=0;j<col;j++)
            {
                temp+=a[i][j];
            }
            while(*temp.rbegin()==' ' && !temp.empty())
                temp.pop_back();
            res.push_back(temp);
        }
        return res;
    }
```

```
// 1328. Break a Palindrome
    string breakPalindrome(string palindrome) {
        string res;
        if (palindrome.size() == 1) return res;
        for (int i = 0; i < palindrome.size() / 2; i++){
            if(palindrome[i] != 'a'){
                palindrome[i] = 'a';
                return palindrome;
            }
        }
        palindrome[palindrome.size() - 1] = 'b';
        return palindrome;
    }
```

```
// 1329. Sort the Matrix Diagonally
    vector<vector<int>> diagonalSort(vector<vector<int>>& mat) {
        unordered_map<int, vector<int> > tmp;
```

```
        int m = mat.size(), n = mat[0].size();

        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                tmp[i-j].push_back(mat[i][j]);

        for(int i = 1-n; i < m; i++)
            sort(tmp[i].begin(),tmp[i].end());

        for(int i = m-1; i >= 0; i--)
            for(int j = n-1; j >= 0; j--) {
                mat[i][j] = tmp[i-j].back();
                tmp[i-j].pop_back();
            }
        return mat;
    }


// 1333. Filter Restaurants by Vegan-Friendly, Price and Distance
    static bool cmp(vector<int>& a,vector<int>& b){
        if(a[1]==b[1])return a[0]>b[0];
        return a[1]>b[1];
    }
    vector<int> filterRestaurants(vector<vector<int>>& res,int veg, int p, int d) {
        vector<vector<int>> v;
        for(auto k:res){
            if((k[2]==veg || veg==0) && k[3]<=p && k[4]<=d)v.push_back(k);
        }
        sort(v.begin(),v.end(),cmp);
        vector<int> ans;
        for(auto k:v)ans.push_back(k[0]);
        return ans;
    }


// 1338. Reduce Array Size to The Half
// Return the minimum size of the set so that at least half of the integers of the array are removed.
// You are given an integer array arr. You can choose a set of integers and remove all the occurrences of these intege
rs in the array.
    int minSetSize(vector<int>& arr) {
        unordered_map<int, int> counter;
        priority_queue<int> q;
        int res = 0, removed = 0;

        for (auto a : arr) counter[a]++;
        for (auto c : counter) q.push(c.second);

        while (removed < arr.size() / 2) {
            removed += q.top();
            q.pop();
            res++;
        }

        return res;
    }
```

// 1339. Maximum Product of Splitted Binary Tree

```
int mod = 1e9 +7;
long long ans = 0;
int updateNodes(TreeNode *root){
    if(not root) return 0;

    int leftSum = updateNodes(root->left);
    int rightSum = updateNodes(root->right);

    root->val += leftSum+rightSum;

    return root->val;
}

void traverse(TreeNode *root, int totalSum){
    if(not root) return;

    ans = max(ans,( (long long)(totalSum-root->val) *  (long long)(root->val)));

    traverse(root->left, totalSum);
    traverse(root->right, totalSum);
}

int maxProduct(TreeNode* root) {
    int totalSum = updateNodes(root);
    traverse(root, totalSum);
    return ans%mod;
}
```

// 1344. Angle Between Hands of a Clock

```
double angleClock(int hour, int minutes) {
    double(abs(5.5*minutes - 30*hour));
    if(double(abs(5.5*minutes - 30*hour))>180)
        return 360-double(abs(5.5*minutes - 30*hour));
    else
        return double(abs(5.5*minutes - 30*hour));
}
```

// 1347. Minimum Number of Steps to Make Two Strings Anagram

```
int minSteps(string s, string t) {
    int Freq[26]={0}; int count=0;
    for(int i=0;i<s.size();i++)
        Freq[s[i]-'a']++,Freq[t[i]-'a']--;
    for(int i=0;i<26;i++)
        if(Freq[i]>0)
            count+=Freq[i];
    return count;
}
```

// 1353. Maximum Number of Events That Can Be Attended

```
 int maxEvents(vector<vector<int>>& events)
{
   int n=events.size();
   priority_queue<int,vector<int>,greater<int>> pq;

   sort(events.begin(),events.end());
   int ans=0;
   int i=0;
   for(int d=1;d<=100000;d++)
   {
      while(!pq.empty()&&pq.top()<d)
      {
         pq.pop();
      }

      while(i<n && events[i][0]==d  )
      {
         pq.push(events[i][1]);
         i++;
      }
      if(!pq.empty())
      {
         pq.pop();
         ans++;
      }

      if(pq.empty()&&i==n)
         break;
   }
   return ans;
}
```

```
// 1358. Number of Substrings Containing All Three Characters
 int numberOfSubstrings(string s) {
   int left = 0 , right = 0 , end = s.size() - 1;
   unordered_map<char,int> map;
   int count = 0;

   while(right != s.size()){
      map[s[right]] += 1;
      while(map['a'] and map['b'] and map['c']){
         count += 1 + (end - right);
         map[s[left]] -= 1;
         left++;
      }
      right++;
   }
   return count;
 }
```

```
// 1371. Find the Longest Substring Containing Vowels in Even Counts
 int findTheLongestSubstring(string s) {
   unordered_set<char>vowel = {'a','e','i','o','u'};
```

```
        unordered_map<int,int>M;
        M[0] = -1;
        int mask = 0;
        int n = s.length();
        int max_len = 0;
        for(int i = 0; i < n; i++)
        {
            if(vowel.find(s[i]) != vowel.end())
            {
                mask ^= 1 << (s[i] - 'a');
                if(M.find(mask) == M.end())
                    M[mask] = i;
            }
            max_len = max(max_len,i - M[mask]);
        }
        return max_len;
    }
```

```
// 1376. Time Needed to Inform All Employees
      int dfs(vector<int> &informTime,vector<vector<int>> &Adjlist,int headID){
        int time=0;
        int node=headID;
        for(int &neighb :Adjlist[node]){
            time=max(time,dfs(informTime,Adjlist,neighb));
        }
        return informTime[node]+time;
    }
    int numOfMinutes(int n, int headID, vector<int>& manager, vector<int>& informTime) {
        vector<vector<int>> Adjlist(n+1);
        for(int i=0;i<n;i++){
           if(manager[i]==-1) continue;
            Adjlist[manager[i]].push_back(i);
        }
        return dfs(informTime,Adjlist,headID);

    }
```

```
// 1395. Count Number of Teams
// Choose 3 soldiers with index (i, j, k) with rating (rating[i], rating[j], rating[k]).
// A team is valid if: (rating[i] < rating[j] < rating[k]) or (rating[i] > rating[j] > rating[k]) where (0 <= i < j < k < n).
    int numTeams(vector<int>& arr) {
              int n = arr.size();
        int result = 0;
        for(int i = 1 ; i < n-1 ; i++){
            int leftSmall = 0, leftLarge = 0;
            int rightSmall = 0, rightLarge = 0;
            //left part
            for(int j = 0 ; j < i ; j++){
                if(arr[j] < arr[i]){
                    leftSmall++;
                }
                if(arr[j] > arr[i]){
                    leftLarge++;
                }
```

```
            }
            for(int j = i+1 ; j < n ; j++){
                if(arr[j] < arr[i]){
                    rightSmall++;
                }
                if(arr[j] > arr[i]){
                    rightLarge++;
                }
            }
            result += leftSmall * rightLarge + leftLarge * rightSmall;
        }
        return result;
    }
};
```

```
// 1400. Construct K Palindrome Strings

    bool canConstruct(string s, int k) {
        if(s.size() < k) return false;

        unordered_map<char, int> count;
        for(char ch : s) count[ch]++;

        int oddCount = 0;

        for(auto itr = count.begin(); itr != count.end(); itr++) {
            if(itr->second & 1) oddCount++;
        }

        if(oddCount <= k) return true;
        return false;
    }
```

```
// If the current number is even, you have to divide it by 2.
// If the current number is odd, you have to add 1 to it.
    int numSteps(string s) {
        int n=s.size()-1; int ans=0;
        while(n>0){
            if(s[n]=='1')ans++;
            ans++;
            n--;
        }
        return ans;
    }
```

```
// 1415. The k-th Lexicographical String of All Happy Strings of Length n
// consists only of letters of the set ['a', 'b', 'c'].
// s[i] != s[i + 1] for all values of i from 1 to s.length - 1 (string is 1-indexed).
    vector<string>ans;
    void solve(int ind , int n , stack<char>&s , string temp)
```

```
    {
        if(temp.size()==n)
        {
            ans.push_back(temp);
            return;
        }
        for(int i = 0 ; i < 3 ; i++)
        {
            if(!s.empty() && s.top()==char(97+i))
            {
                continue;
            }
            temp.push_back(97+i);
            s.push(97+i);
            solve(i+1,n,s,temp);
            temp.pop_back();
            s.pop();
        }
        return;
    }
    string getHappyString(int n, int k) {
        stack<char>s;
        string temp;
        solve(0,n,s,temp);
        if(k>ans.size())
        {
            return "";
        }
        return ans[k-1];
    }
};
```

```
// 1419. Minimum Number of Frogs Croaking
int minNumberOfFrogs(string croak) {
        int c = 0, r = 0, o = 0, a = 0, k = 0, in_use = 0, answer = 0;
        for (char d:croak) {
            switch(d) {
                case 'c':
                    c++;
                    in_use++;
                    break;
                case 'r':
                    r++;
                    break;
                case 'o':
                    o++;
                    break;
                case 'a':
                    a++;
                    break;
                case 'k':
                    k++;
                    in_use--;
                    break;
```

```
        }
  answer = max(answer, in_use);
        if ((c < r) || (r < o) || (o < a) || (a < k))
            return -1;
    }
    if (in_use == 0 && c == r && c == o && c == a && c == k)
        return answer;

    return -1;
}
```

```
// 1424. Diagonal Traverse II
 vector<int> findDiagonalOrder(vector<vector<int>>& matrix) {
    vector<int> res;
    map<int, vector<int>> mp;

    for(int i = 0 ; i < matrix.size() ; i++)
        for(int j = 0 ; j < matrix[i].size() ; j++)
            mp[i + j].push_back(matrix[i][j]);

    for(auto i : mp) {
        reverse(i.second.begin(), i.second.end());
        for(auto k : i.second)
            res.push_back(k);
    }
    return res;
 }
```

```
// 1433. Check If a String Can Break Another String
 bool help(string& s1, string& s2){
    for(int i=0; i<s1.size(); i++){
        if(s1[i]<s2[i]){
            return false;
        }
    }
    return true;
 }
 bool checkIfCanBreak(string s1, string s2) {
    sort(s1.begin(), s1.end());
    sort(s2.begin(), s2.end());
    return help(s1, s2) || help(s2, s1);
 }
```

```
// 1442. Count Triplets That Can Form Two Arrays of Equal XOR
 int countTriplets(vector<int>& a) {
    int n = a.size(), ans = 0;
    for(int i = 0; i < n; i++) {
        int curr = a[i];
        for(int j = i+1; j < n; j++) {
            curr = (curr ^ a[j]);
            if(curr == 0) {
```

```
                ans += j-i;
            }
        }
    }
    return ans;
}
```

```
// 1448. Count Good Nodes in Binary Tree
// good if in the path from root to X there are no nodes with a value greater than X.
    void dfs(TreeNode* root,int maxi,int &goodnode){
        if(!root) return;

        if(root->val>=maxi){
            goodnode++;
            maxi = root->val;
        }
        dfs(root->left,maxi,goodnode);
        dfs(root->right,maxi,goodnode);
    }
    int goodNodes(TreeNode* root) {
        if(!root) return 0;
        if(!root->left && !root->right) return 1;
        int goodnode = 1;
        dfs(root->left,root->val,goodnode);
        dfs(root->right,root->val,goodnode);

        return goodnode;
    }
```

```
// 1451. Rearrange Words in a Sentence
    string arrangeWords(string text) {
        text[0] = tolower(text[0]);
        stringstream ss(text), rs;
        string word;
        map<int, string> m;
        while (ss >> word) m[word.size()] += word + " ";

        for (const auto& pair: m) rs << pair.second;
        string result = rs.str();
        result.pop_back();
        result[0] = toupper(result[0]);
        return result;
    }
```

```
// 1456. Maximum Number of Vowels in a Substring of Given Length
   bool isVowel(char ch){
        return (ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u');
    }
    int maxVowels(string s, int k) {
        int i=0,j=0;
        int ans=0,cnt=0;
        while(j<s.length()){
```

```
            if(isVowel(s[j]))
              cnt++;
            if(j-i+1==k){
                ans=max(ans,cnt);
                if(isVowel(s[i]))
                    cnt--;
                i++;
            }
            j++;
        }
        return ans;
    }
```

// 1461. Check If a String Contains All Binary Codes of Size K

```
    bool hasAllCodes(string s, int k) {
        set<string>m;
        int i=0,j=0,n=s.size();
        while(i+k<=n){
            m.insert(s.substr(i,k)); i++;
        }
        return m.size()==(1<<k);
    }
```

// 1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

```
        const long long int mod = 1000000007;
    int maxArea(int hh, int ww, vector<int>& h, vector<int>& v) {
        // Sort
        sort(h.begin(), h.end());
        sort(v.begin(), v.end());


        // Horizontal
        vector<int> heights = {h[0]};
        int nh = h.size();
        for (int i=1; i<nh; i++) {
            heights.push_back(h[i]-h[i-1]);
        }
        heights.push_back(hh-h[nh-1]);

        // Vertical
        vector<int> lengths = {v[0]};
        int nv = v.size();
        for (int i=1; i<nv; i++) {
            lengths.push_back(v[i]-v[i-1]);
        }
        lengths.push_back(ww-v[nv-1]);

        // Take max
        long long int a = *max_element(heights.begin(), heights.end());
        long long int b = *max_element(lengths.begin(), lengths.end());

        // Multiply and return
        return (int)(a%mod*b%mod);
```

```
    }
```

```
// 1466. Reorder Routes to Make All Paths Lead to the City Zero
// reverse min paths- reverse dijkstra
    int minReorder(int n, vector<vector<int>>& connections) {
        vector<pair<int,int>> graph[n];
        for(auto i: connections) {
            graph[i[0]].push_back({1, i[1]});
            graph[i[1]].push_back({0, i[0]});
        }
        vector<bool> vis(n, 0);
        int ans = 0;
  queue<int> q;
  q.push(0); vis[0]=1;
        while(!q.empty()) {
            int node = q.front(); q.pop();
            for(auto [w,v]: graph[node]) {
                if(vis[v]) continue;
                ans += w;
    q.push(v); vis[v] = 1;
            }
        }
        return ans;
    }
```

```
// 1481. Least Number of Unique Integers after K Removals
   int findLeastNumOfUniqueInts(vector<int> &arr, int k){
        priority_queue<int, vector < int>, greater<int>> st;
        unordered_map<int, int> m;
        for (int i = 0; i < arr.size(); i++)
            m[arr[i]]++;
        for (auto i: m)
            st.push(i.second);
        while (k)
        {
            if (st.top() <= k)
            {
                k -= st.top();
                st.pop();
            }
            else
            {
                return st.size();
            }
        }
        return st.size();
    }
```

```
// 1482. Minimum Number of Days to Make m Bouquets
    bool isvalid(vector<int> &nums, int mid, int k, int m) {
        int ans = 0, count = 0;
        for(int i = 0;i<nums.size();i++){
```

```
            if(mid>=nums[i]) count++;
            else count = 0;
            if(count==k){
                ans++;
                count = 0;
            }
        }
        return ans>=m;
    }
    int minDays(vector<int>& bloomDay, int m, int k) {
        int n = bloomDay.size();
        int ans = 0;
        if(m*k > n) return -1;

        int start = 0, end = *max_element(bloomDay.begin(), bloomDay.end());

        while(start<=end)
        {
            int mid = (start+(end-start)/2);
            if(isvalid(bloomDay, mid, k, m))
            {
                ans = mid;
                end = mid-1;
            }
            else start = mid+1;
        }
        return ans;
    }


// 1492. The kth Factor of n
    int kthFactor(int n, int k) {
        vector<int>v;
        vector<int>v2;
        v.push_back(1);
        for(int i=2;i*i<=n;i++){
            if(n%i==0)
            {
                v.push_back(i);
                if(i*i!=n)
                    v2.insert(v2.begin(),n/i);
            }
        }
        v.insert(v.end(),begin(v2),end(v2));
        v.push_back(n);
        if(k>v.size())  return -1;
        return v[k-1];
    }

//    1493. Longest Subarray of 1's After Deleting One Element
    int longestSubarray(vector<int>& nums) {
        int n = nums.size();
        int maxcount = 0;
        int count = 0;
        int beg = 0, end = 0;
```

```
        while(end < n){
            if(nums[end] == 0)
                count++;
            if(count >= 2){
                count--;
                maxcount = max(maxcount, end-beg-count);
                while(beg < n && nums[beg] != 0)
                    beg++;
                beg++;
            }
            end++;
        }
        if(count == 0)
            count++;
        maxcount = max(maxcount, end-beg-count);

        return maxcount;
    }
```

```
// 1497. Check If Array Pairs Are Divisible by k
// We want to divide the array into exactly n / 2 pairs such that the sum of each pair is divisible by k.
    bool canArrange(vector<int>& arr, int k) {
        if(arr.size()&1) return false;
        unordered_map<int,int>m;
        for(auto x:arr)
            m[(x%k + k)%k]++;

        for(auto x:arr){
            int rem=(x%k + k)%k;
            if(rem==0){
                if(m[rem] & 1) return false;
            }
            else if(m[rem] != m[k - rem]) return false;
        }
        return true;
    }
```

```
// 1498. Number of Subsequences That Satisfy the Given Sum Condition
    int md=1e9+7;
    int numSubseq(vector<int>& nums, int tg) {
        sort(begin(nums),end(nums));
        int l=0,r=size(nums)-1,res=0;
        vector<int>pw2(r+1,1);
        for(int i=1;i<=r;++i)pw2[i]=(pw2[i-1]*2)%md;
        while(l<=r){
            if(nums[l]+nums[r]>tg)--r;
            else res=(res+pw2[r-l++])%md;
        }
        return res%md;
    }
```

```
// 1509. Minimum Difference Between Largest and Smallest Value in Three Moves
    int minDifference(vector<int>& nums) {
        if(nums.size() <= 4) return 0;
        int  n = nums.size();
```

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

        int op1 = nums[n-4] - nums[0];
        int op2 = nums[n-3] - nums[1];
        int op3 = nums[n-2] - nums[2];
        int op4 = nums[n-1] - nums[3];

        return min( op1, min( op2, min(op3, op4)));
    }

// 1557. Minimum Number of Vertices to Reach All Nodes
    vector<int> findSmallestSetOfVertices(int n, vector<vector<int>>& edges) {
        vector<int> res, seen(n);
        for (auto& e: edges)
            seen[e[1]] = 1;
        for (int i = 0; i < n; ++i)
            if (seen[i] == 0)
                res.push_back(i);
        return res;
    }


// 1559. Detect Cycles in 2D Grid
 int dx[4] = {1,0,-1,0}; int dy[4] = {0,1,0,-1};
    bool containsCycle(vector<vector<char>>& grid) {
        int m = grid.size(), n = grid[0].size();
        vector <vector <int>> visited(m,vector<int>(n,0));
        for(int i = 0; i<m; i++)
            for(int j = 0; j<n; j++)
                if(!visited[i][j])
                    if(detect(i,j,-1,-1,m,n,grid,visited))
                        return true;
        return false;
    }
    bool isvalid(int x, int y, int m, int n, char c, vector <vector<char>>&grid){
        if(x<0 or x>=m or y<0 or y>=n) return false;
        if(grid[x][y]!=c) return false;
        return true;
    }
    bool detect(int x, int y, int px, int py, int m, int n,vector <vector <char>>&grid, vector<vector<int>>&visited){
    if(visited[x][y]) return true;
        visited[x][y] = 1;

    for(int i = 0; i<4; i++){
            if(isvalid(x+dx[i],y+dy[i],m,n,grid[x][y],grid)){
                if((x+dx[i]!=px or y+dy[i]!=py) and detect(x+dx[i],y+dy[i],x,y,m,n,grid,visited)) return true;
            }
        }
        return false;
    }

// 1513. Number of Substrings With Only 1s
    int numSub(string s) {
        int res = 0, count = 0, mod = 1e9 + 7;
        for (char c: s) {
```

```
            count = c == '1' ? count + 1 : 0;
            res = (res + count) % mod;
        }
        return res;
    }
```

```
// 1578. Minimum Time to Make Rope Colorful
    int minCost(string s, vector<int>& cost) {
        if (s.empty()) return 0;
        int minCost = 0;
        stack<int> stk; // stack to maintain indices parsed till now
        stk.push(0);
        for (int i = 1; i < s.length(); ++i ) {
            if (!stk.empty() && s[stk.top()] == s[i]) {
                if (cost[stk.top()] <= cost[i]) {
                    minCost += cost[stk.top()];
                    stk.pop();
                    stk.push(i);
                } else {
                    minCost += cost[i];
                }
            } else {
                stk.push(i);
            }
        }
        return minCost;
    }
```

```
// 1584. Min Cost to Connect All Points
    int minCostConnectPoints(vector<vector<int>>& points) {
        if (points.size() < 1) return 0;
        auto dist = [](int x1, int y1, int x2, int y2) {
            return abs (x1-x2) + abs(y1-y2);
        };

        unsigned sz = points.size();
        vector<int> minDists(sz,0);
  //Initialize
        int result = 0;
        minDists[0] = INT_MAX;
        for (auto i = 1 ; i < sz; i++) {
            minDists[i] = dist(points[0][0],points[0][1],points[i][0],points[i][1]);
        }

  //Build Spanning Tree
        for (auto i = 1; i < sz; i++) {
            auto it = min_element(minDists.begin(), minDists.end());
            result += *it;
            int index = it - minDists.begin();
            *it = INT_MAX;
            for (auto i = 0 ; i < sz; i++) {
                if (minDists[i] == INT_MAX) continue;
                minDists[i] = min(minDists[i], dist(points[i][0], points[i][1], points[index][0], points[index][1]));
```

```
        }
    }
    return result;
}
```

// 1590. Make Sum Divisible by P
// Return the length of the smallest subarray that you need to remove, or -1 if it's impossible.

```
int minSubarray(vector<int>& A, int p) {
    int n = A.size(), res = n, need = 0, cur = 0;
    for (auto a : A)
        need = (need + a) % p;
    unordered_map<int, int> last = {{0, -1}};
    for (int i = 0; i < n; ++i) {
        cur = (cur + A[i]) % p;
        last[cur] = i;
        int want = (cur - need + p) % p;
        if (last.count(want))
            res = min(res, i - last[want]);
    }
    return res < n ? res : -1;
}
```

// 1631. Path With Minimum Effort

```
bool isValid(vector<vector<int>>& h, int x, int y) {
    return x < n and x >= 0 and y < m and y >= 0;
}

bool recDFS(vector<vector<int>>& h, int k, int x, int y) {
    visited[x][y] = true;
    if (x == n-1 && y == m-1)
        return true;

    for (int i = 0; i < 4; i++) {
        int x_curr = x + x_points[i];
        int y_curr = y + y_points[i];
        if (isValid(h, x_curr, y_curr) && !visited[x_curr][y_curr] && abs(h[x_curr][y_curr] - h[x][y]) <= k)
            if (recDFS(h, k, x_curr, y_curr)) return true;
    }

    return false;
}

bool possibleLessEqK(vector<vector<int>>& h, int k) {
    visited.assign(n,vector<bool> (m, false));
    return recDFS(h, k, 0, 0);
}

int minimumEffortPath(vector<vector<int>>& heights) {
    n = heights.size();
    m = heights[0].size();

    int lo = 0, hi = 1e6, mid;
    while (lo < hi) {
        mid = lo + (hi - lo) / 2;
```

```
            if (possibleLessEqK(heights, mid))
                hi = mid;
            else
                lo = mid + 1;
        }

        return lo;
    }
```

```
// 1647. Minimum Deletions to Make Character Frequencies Unique
    int minDeletions(string s) {
      int ans = 0;
      unordered_map<char, int> char_counts;
      for (const char& c : s) char_counts[c]++;
      unordered_set<int> seen;
      for (auto[k, v] : char_counts) {
        while (seen.find(v) != seen.end()) {
          v--;
          ans++;
        }
        // add it
        if (v > 0) seen.insert(v);
      }

      return ans;
    }
```

```
// 1760. Minimum Limit of Balls in a Bag
// Input: nums = [9], maxOperations = 2
// Output: 3
// Explanation:
// - Divide the bag with 9 balls into two bags of sizes 6 and 3. [9] -> [6,3].
// - Divide the bag with 6 balls into two bags of sizes 3 and 3. [6,3] -> [3,3,3].
// The bag with the most number of balls has 3 balls, so your penalty is 3 and you should return 3.
    int minimumSize(vector<int>& A, int k) {
        int left = 1, right = 1e9;
        while (left < right) {
            int mid = (left + right) / 2, count = 0;
            for (int a : A)
                count += (a - 1) / mid;
            if (count > k)
                left = mid + 1;
            else
                right = mid;
        }
        return left;
    }
```

```
// 1743. Restore the Array From Adjacent Pairs
    vector<int> restoreArray(vector<vector<int>>& adjacentPairs) {
    unordered_map<int, vector<int>> ps;
```

```
    for (auto &p : adjacentPairs) {
        ps[p[0]].push_back(p[1]);
        ps[p[1]].push_back(p[0]);
    }
    vector<int> res;
    for (auto &p : ps) {
        if (p.second.size() == 1) {
            res.push_back(p.first);
            res.push_back(p.second[0]);
            break;
        }
    }
    while (res.size() < adjacentPairs.size() + 1) {
        auto tail = res.back(), prev = res[res.size() - 2];
        auto &next = ps[tail];
        if (next[0] != prev)
            res.push_back(next[0]);
        else
            res.push_back(next[1]);
    }
    return res;
}



    bool valid(int rsize,int csize,int r,int c){
        if(rsize<=r||csize<=c||c<0||r<0)return false;
        return true;
    }
    int DFS(vector<vector<int>>& grid,int r,int c){
        if(r==grid.size())return c;
        if(valid(grid.size(),grid[0].size(),r,c)){
            if(grid[r][c]==1){
                if(valid(grid.size(),grid[0].size(),r,c+1)&&grid[r][c+1]==1){
                    return DFS(grid,r+1,c+1);
                }
            }else{
                if(valid(grid.size(),grid[0].size(),r,c-1)&&grid[r][c-1]==-1){
                    return DFS(grid,r+1,c-1);
                }
            }
        }
        return -1;
    }
    vector<int> findBall(vector<vector<int>>& grid){
        int row = grid.size(),col = grid[0].size();
        vector<int>ans(col,0);
        for(int c = 0;c<col;c++){
            ans[c] = DFS(grid,0,c);
        }
    return ans;
    }
// Find minimum and maximum element in an array
pair<long long, long long> f(long long a[], int l, int r)
{
```

```
    if (l == r)
        return {a[l], a[l]};

    int m = (r - l) / 2 + l;
    auto p1 = f(a, l, m);
    auto p2 = f(a, m + 1, r);

    return {min(p1.first, p2.first), max(p1.second, p2.second)};
}
pair<long long, long long> getMinMax(long long a[], int n)
{
    return f(a, 0, n - 1);
}

void sort012(int a[], int n)
{
    int l = 0, m = 0, r = n - 1;
    while (m <= r)
    {
        if (a[m] == 1)
            m++;
        else if (a[m] == 0)
            swap(a[l++], a[m++]);
        else
            swap(a[m], a[r--]);
    }
}

void movingNegToLeft(int a[], int n)
{
    int l = 0, r = 0;
    while (r < n)
    {
        if (a[r] < 0)
            swap(a[l++], a[r++]);
        else
            r++;
    }
}

// Input:N = 5 A[] = {1, 2, 3, 4, 5}
// Output: 5 1 2 3 4
void rotate(int arr[], int n)
{
    for (int i = n - 1; i > 0; i--)
        swap(arr[i], arr[i - 1]);
}

long long maxSubarraySum(int arr[], int n)
{
    long long ans = INT_MIN, temp = 0;
    for (int i = 0; i < n; i++)
    {
        temp += arr[i];
```

```
        ans = max(ans, temp);
        if (temp < 0)
        {
            temp = 0;
        }
    }
    return ans == INT_MIN ? -1 : ans;
}

int getMinDiff(int arr[], int n, int k)
{
    sort(arr, arr + n);
    int minele, maxele;
    int result = arr[n - 1] - arr[0]; // b-a
    for (int i = 1; i <= n - 1; i++)
    {
        if (arr[i] >= k) // height not negative
        {
            maxele = max(arr[i - 1] + k, arr[n - 1] - k);
            minele = min(arr[0] + k, arr[i] - k);
            result = min(result, maxele - minele);
        }
    }
    return result;
}

int minJumps(int nums[], int n)
{
    if (nums[0] == -1)
        return -1;
    if (n == 1)
        return 0;

    int i = 0, maxReachable = 0, lastJumpedPos = 0, jumps = 0;
    while (lastJumpedPos < n - 1)
    {
        maxReachable = max(maxReachable, i + nums[i]);
        if (i == lastJumpedPos)
        {
            lastJumpedPos = maxReachable;
            jumps++;
        }
        i++;
    }
    return jumps;
}

//merge unsorted arrays
void merge(int arr1[], int arr2[], int n, int m)
{
    int i = n - 1, j = 0;
    while (i >= 0 && j < m)
    {
        if (arr1[i] > arr2[j])
        {
```

```
            swap(arr1[i], arr2[j]);
            i--;
            j++;
        }
        else
            break;
    }
    sort(arr1, arr1 + n);
    sort(arr2, arr2 + m);
}
//Using Extra O(n) space, Time complexity : O(n)
    // long long arr[m+n],i = 0,j = 0, k = 0;
    // while(j < n && k < m){
    //     if(arr1[j] < arr2[k])
    //         arr[i++] = arr1[j++];
    //     else
    //         arr[i++] = arr2[k++];
    // }
    // while(j < n) arr[i++] = arr1[j++];
    // while(k < m) arr[i++] = arr2[k++];

    // j=0;
    // for(int i = 0; i < n; i++)
    //     arr1[i] = arr[j++];

    // for(int i = 0; i < m; i++)
    //     arr2[i] = arr[j++];

int nextGap(int gap)
{
    if (gap <= 1)
        return 0;
    return (gap / 2) + (gap % 2);
}

void merge(int* arr1, int* arr2, int n, int m)
{
    int i, j, gap = n + m;
    for (gap = nextGap(gap);
         gap > 0; gap = nextGap(gap))
    {
        // comparing elements in the first array.
        for (i = 0; i + gap < n; i++)
            if (arr1[i] > arr1[i + gap])
                swap(arr1[i], arr1[i + gap]);

        // comparing elements in both arrays.
        for (j = gap > n ? gap - n : 0;
             i < n && j < m;
             i++, j++)
            if (arr1[i] > arr2[j])
                swap(arr1[i], arr2[j]);

        if (j < m) {
            // comparing elements in the second array.
```

```
            for (j = 0; j + gap < m; j++)
                if (arr2[j] > arr2[j + gap])
                    swap(arr2[j], arr2[j + gap]);
        }
    }
}

void alternatePositiveNegative(int a[], int n)
{
    int n; cin >> n;
    long long a[n+10];
    for(int i=0;i<n;i++) cin>>a[i];
        int l=0, r=0;
    while(r<n){
        if(a[r]<0) swap(a[l++],a[r++]);
        else r++;
    }
    for(int i=0;i<n;i++) cout<<" "<<a[i]; cout<<"\n";
    for(int i=0;i<n/2;i+=2){
        cout<<" "<<a[i] << " " <<a[i+(n/2)]<<"\n";
        swap(a[i],a[i+(n/2)]);
    }
    for(int i=0;i<n;i++) cout<<" "<<a[i]; cout<<"\n";
}


 long long maxProduct(vector<int> a, int n) {
     if(!a.size()) return 0;
     long long ans= a[0], mxP= a[0], mnP= a[0];
     for(int i=1;i<n;i++){
         if(a[i]<0) swap(mxP, mnP);
         mxP = max(mxP*a[i], (long long)(a[i]));
         mnP = min(mnP*a[i], (long long)(a[i]));
         ans = max(mxP, ans);
     }
     return ans;
 }

    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> s(begin(nums), end(nums));
        int longest = 0;
        for(auto& num : s) {
            if(s.count(num - 1)) continue;
            int j = 1;
            while(s.count(num + j)) j++;
            longest = max(longest, j);
        }
        return longest;
    }

    int longestConsecutive(vector<int>& nums) {
        // way1: brute force
        // way2: hashmap
        // way3: sorting
```

```
        // if(!size(nums)) return 0;
        // sort(begin(nums), end(nums));
        // int longest = 0, cur_longest = 1;
        // for(int i = 1; i < size(nums); i++)
        //     if(nums[i] == nums[i - 1]) continue;
        //     else if(nums[i] == nums[i - 1] + 1) cur_longest++;
        //     else longest = max(longest, cur_longest), cur_longest = 1;
        // return max(longest, cur_longest);
    }

int majorityElement(vector<int>& nums)
{
int c(-1), cnt(0); // c = candidate, cnt = counter
for(auto n: nums)
{
if (cnt == 0)
c = n;
cnt += (n == c) ? 1 : -1;
}
return c;
}


    int minSubArrayLen(int tgt, vector<int>& a) {
        int l=0, r=0, t=0,ans=INT_MAX, n=a.size();
        while(r<n){
            t+=a[r++];
            while(t>=tgt)
                t-=a[l++], ans = min(ans, r-l+1);
        }
        return ans==INT_MAX?0:ans;
    }

        string countAndSay(int n) {
        string ans="1";
        n--;
        while(n--){
            string res=ans, subAns;
            for (int i = 0; i < res.size(); i++) {
                int count = 1;
                while (i + 1 < res.size() && res[i] == res[i+1]) {
                    count++;
                    i++;
                }
                subAns += to_string(count) + res[i];
            }
            ans = subAns;
            cout<< ans<<"\n";
        }
        return ans;
        }



int countRev (string s)
```

```
{
   int len = s.length();
   if(len%2!=0) return -1;
   int open = 0, close = 0;

   for(int i = 0;i<len;i++)
   {
      char ch = s[i];
      if(ch == '{') open++;
      else if(ch == '}') {
         if(open > 0) open--;
         else close++;
      }
   }

   int count = 0;

   count+=open/2;
   count+=close/2;

   count += (open%2) + (close%2);

   return count;
}
```

```
struct Node* addTwoLists(struct Node* first, struct Node* second)
{
   // code here
   Node* firstr=reverse(first);
   Node* secondr=reverse(second);
   Node* ans=new Node(0);
   Node* ansh=ans;
   int sum=0,carry=0;
   while(firstr||secondr)
   {
      sum=(firstr?firstr->data:0)+ (secondr?secondr->data:0)+carry;
      carry=sum>=10?1:0;
      int add=sum%10;
      ans->next=new Node(add);

      if(firstr){firstr=firstr->next;}

      if(secondr){secondr=secondr->next;}

      ans=ans->next;
   }
   if(carry>0) ans->next=new Node(carry);

   return reverse(ansh->next);
}
```

```
  Node* findIntersection(Node* head1, Node* head2)
{
  Node *head3 =new Node(-1);
  Node *p3 = head3;

  while(head1!=NULL && head2 != NULL){
     if(head1->data == head2->data){
        p3->next = head1;
        head1 = head1->next;
        head2 = head2->next;
        p3 = p3->next;
     }
     else if(head1->data < head2->data){
        head1 = head1->next;
     }
     else if(head1->data > head2->data){
        head2 = head2->next;
     }
  }

  return head3->next;
}


 void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
       int i=m-1,j=n-1,k=m+n-1;
       while(i>=0&&j>=0)
       {
          if(nums1[i]>nums2[j])
          {
             nums1[k]=nums1[i];
             i--;
             k--;
          }
          else
          {
             nums1[k]=nums2[j];
             j--;
             k--;
          }
       }
       while(i>=0)
          nums1[k--]=nums1[i--];
       while(j>=0)
          nums1[k--]=nums2[j--];
   }


int knapSack(int w, int wt[], int val[], int n)
   {
     int t[n + 1][w + 1];

     for (int i = 0; i < n + 1; i++) {
        for (int j = 0; j < w + 1; j++) {
           if (i == 0 || j == 0) {
```

```
                t[i][j] = 0;
            } else {
                if (wt[i - 1] <= j) {
                    t[i][j] = max(val[i - 1] + t[i - 1][j - wt[i - 1]], t[i - 1][j]);
                } else if (wt[i - 1] > j) {
                    t[i][j] = t[i - 1][j];
                }
            }
        }
    }

    return t[n][w];
}


struct Node* reverseList(struct Node *head){
    Node* cur=head;
    Node* prv=NULL;
    Node* nextptr;
    while(cur!=NULL){
        nextptr=cur->next;
        cur->next=prv;
        prv=cur;
        cur=nextptr;
    }
    return prv;
}
struct Node* middle(struct Node *head){
    Node* slow=head;
    Node* fast=head;
    while(fast!=NULL && fast->next!=NULL){
        slow=slow->next;
        fast=fast->next->next;
    }
    return slow;
}
bool isPalindrome(Node *head)
{
    //Your code here
    if(head==NULL){
        return true;
    }
    Node* mid=middle(head);
    Node* last=reverseList(mid);
    Node* curr=head;
    while(last!=NULL){
        if(last->data!=curr->data){
            return false;
        }
        last=last->next;
        curr=curr->next;
    }
    return true;
}
```

```
  Node* reverseDLL(Node * head)
{
   if(!head) return head;
   auto node = head;
   while(1){
      swap(node->prev, node->next);
      if(node->prev)
         node = node->prev;
      else
         return node;
   }
   return 0;
}

//container with most water
   // way 1: brute force o(n)
   //wa 2: sliding window two pointer o ( n)
   int maxArea(vector<int>& H) {
      int ans = 0, i = 0, j = H.size()-1, res = 0;
      while (i < j) {
         if (H[i] <= H[j]) {
            res = H[i] * (j - i);
            i++;
         }
         else {
            res = H[j] * (j - i);
            j--;
         }
         if (res > ans) ans = res;
      }
      return ans;
   }


      int deleteAndEarn(vector<int>& nums) {
      int n = 10001;

      vector<int> sum(n, 0);
      vector<int> dp(n, 0);

      for(auto num: nums){
         sum[num] += num;
      }

      dp[0] = 0;
      dp[1] = sum[1];

      for(int i=2; i<n; i++)
         dp[i] = max(dp[i-2] + sum[i], dp[i-1]);

      return dp[n-1];
   }
```

```
    int trapRainWater2(vector<vector<int>>& mat)
    {
        int n=mat.size();
        int m=mat[0].size();
        priority_queue<pair<int,pair<int,int>>,vector<pair<int,pair<int,int>>>,greater<pair<int,pair<int,int>>>>pq;
        vector<vector<bool>>vis(n,vector<bool>(m,false));

        for(int i=0;i<n;i++)
            for(int j=0;j<m;j++)
                if(i==0||j==0||i==n-1||j==m-1)
                {
                    pq.push({mat[i][j],{i,j}});
                    vis[i][j]=true;
                }

        vector<pair<int,int>>d={{-1,0},{1,0},{0,-1},{0,1}};
        int ans=0;
        while(pq.size()){
            auto temp=pq.top(); pq.pop();
            int val=temp.first;
            int x=temp.second.first;
            int y=temp.second.second;
            for(auto it:d)
            {
                int nx=x+it.first;
                int ny=y+it.second;
                if(nx>=0&&ny>=0&&nx<n&&ny<m&&vis[nx][ny]==false)
                {
                    vis[nx][ny]=true;
                    ans+=max(0,val-mat[nx][ny]);
                    pq.push({max(val,mat[nx][ny]),{nx,ny}});
                }
            }
        }
        return ans;
    }


//eggdrop binary approach
    int find(int k,int n,vector<vector<int>> &memo)
    {   if(n==0||n==1) return n;
        if(k==1) return n;
        if(memo[k][n]!=-1) return memo[k][n];
        int ans=1000000,l=1,h=n,temp=0;

        while(l<=h)
        {
            int mid=(l+h)/2;
            int left=find(k-1,mid-1,memo);
            int right=find(k,n-mid,memo) ;
            temp=1+max(left,right);
            if(left<right){
              l=mid+1;
            }
            else
```

```
            {
                h=mid-1;
            }
            ans=min(ans,temp);
        }
        return memo[k][n]=ans;

    }
    int superEggDrop(int K, int N) {
        //K -> egg , N -> floor
        vector<vector<int>> memo(K+1,vector<int> (N+1,-1));
        return find(K,N,memo);

    }
```

```
// 1249. Minimum Remove to Make Valid Parentheses
// Input: s = "lee(t(c)o)de)"
// Output: "lee(t(c)o)de"
    string minRemoveToMakeValid(string s) {
        stack<int>st;
        for(int i=0;i<s.length();++i){
            if(s[i]=='(')
                st.push(i);
            else if(s[i]==')')
                if(st.empty())
                    s[i]='#';
                else
                    st.pop();
        }
        while(!st.empty()){
            s[st.top()]='#';
            st.pop();
        }

        string ans="";
        for(int i=0;i<s.length();++i){
            if(s[i]!='#')
                ans.push_back(s[i]);
        }

        return ans;
    }
```

```
// 22. Generate Parentheses
    vector <string> valid;
    void generate( string &s, int open , int close)
    {
        if(open==0 && close==0){
            valid.push_back(s);
            return;
        }

        if(open > 0 ){
```

```
            s.push_back('(');
            generate(s, open-1, close);
            s.pop_back();
        }

        if(close > 0){
            if(open < close){
                s.push_back(')');
                generate(s, open, close-1);
                s.pop_back();
            }
        }
    }
    vector<string> generateParenthesis(int n) {
        string s;
        generate(s,n,n);
        return valid;
    }


//Basic Calculator 2
    int calculate(string s) {
        vector<int> vt;
        int num=0;
        char sign='+';

        for(int i=0;i<=s.length();i++){
            if(s[i]>='0'&&s[i]<='9'){
                num=num*10+(s[i]-'0');
            }
            else if(s[i]=='+'||s[i]=='-'||s[i]=='/'||s[i]=='*'||i==s.length()){
                if(sign=='+'){
                    vt.push_back(num);
                    num=0;
                }else if(sign=='-'){
                    vt.push_back(-1*num);
                    num=0;
                }else if(sign=='/'){
                    vt[vt.size()-1]=vt.back()/num;
                    num=0;
                }else if(sign=='*'){
                    vt[vt.size()-1]=vt.back()*num;
                    num=0;
                }

                if(i!=s.length()){
                 sign=s[i];
                }
            }
        }

        int result=0;
        for(int i=0;i<vt.size();i++)
            result+=vt[i];
        return result;
```

```
    }



// 12. Integer to Roman
        const int val[13] = {1000,900,500,400,100,90,50,40,10,9,5,4,1};
    const string rom[13] = {"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};

    string intToRoman(int N) {
        string ans = "";
        for (int i = 0; N; i++)
            while (N >= val[i]) ans += rom[i], N -= val[i];
        return ans;
    }



// 273. Integer to English Words
    static string numberToWords(int n) {
        if(n == 0) return "Zero";
        else return int_string(n).substr(1);
    }
    static const char * const below_20[] =  {"One", "Two", "Three", "Four","Five","Six","Seven","Eight","Nine","Te
n", "Eleven","Twelve","Thirteen","Fourteen","Fifteen","Sixteen","Seventeen","Eighteen","Nineteen"}
    static const char * const below_100[] = {"Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Nine
ty"}
    static string int_string(int n) {
        if(n >= 1000000000)   return int_string(n / 1000000000) + " Billion" + int_string(n - 1000000000 * (n / 100000
0000));
        else if(n >= 1000000) return int_string(n / 1000000) + " Million" + int_string(n - 1000000 * (n / 1000000));
        else if(n >= 1000)    return int_string(n / 1000) + " Thousand" + int_string(n - 1000 * (n / 1000));
        else if(n >= 100)     return int_string(n / 100) + " Hundred" + int_string(n - 100 * (n / 100));
        else if(n >= 20)      return string(" ") + below_100[n / 10 - 2] + int_string(n - 10 * (n / 10));
        else if(n >= 1)       return string(" ") + below_20[n - 1];
        else return "";
        }


// 1344. Angle Between Hands of a Clock
    double angleClock(int hour, int minutes) {
        double(abs(5.5*minutes - 30*hour));
        if(double(abs(5.5*minutes - 30*hour))>180)
            return 360-double(abs(5.5*minutes - 30*hour));
        else
            return double(abs(5.5*minutes - 30*hour));
    }



// 797. All Paths From Source to Target
    int target;
    vector<vector<int>> res;
    vector<int> tmp;
```

```
  void dfs(vector<vector<int>>& graph, int currNode = 0) {
    tmp.push_back(currNode);

    if (currNode == target)
      res.push_back(tmp);
    else
      for (int node: graph[currNode]) {
      dfs(graph, node);
    }

tmp.pop_back();
  }
  vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
    target = graph.size() - 1;
    dfs(graph);
    return res;
  }


  //bfs
  bool vis(int v, vector<int> &path)
  {
    for(auto i: path)
      if(i == v)
        return true;
    return false;
  }

  vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
    int src = 0, des = graph.size()-1;
    vector<vector<int>> res;
    vector<int> path;
    path.push_back(src);
    queue<vector<int>> q;
    q.push(path);
    while(!q.empty())
    {
      path = q.front();
      q.pop();
      int last_val = path.back();
      if(last_val == des)
        res.push_back(path);
      for(auto v: graph[last_val])
      {
        if(!vis(v, path))
        {
          vector<int> temp(path);
          temp.push_back(v);
          q.push(temp);
        }
      }
    }
    return res;
  }
```

```
// 617. Merge Two Binary Trees
    TreeNode* mergeTrees(TreeNode* t1, TreeNode* t2) {
        if ( t1 && t2 ) {
            TreeNode * root = new TreeNode(t1->val + t2->val);
            root->left = mergeTrees(t1->left, t2->left);
            root->right = mergeTrees(t1->right, t2->right);
            return root;
        }
        else
            return t1 ? t1 : t2;

    }
```

```
// 257. Binary Tree Paths
    vector<string> ans;
    void preorder(TreeNode* root, string add)
    {
        if(!root)
            return;
        if(add.size() != 0)
            add += "->";
        add += to_string(root -> val);
        if(root -> left == nullptr and root -> right == nullptr)
            ans.push_back(add);
        preorder(root -> left, add);
        preorder(root -> right, add);
    }

    vector<string> binaryTreePaths(TreeNode* root) {
        preorder(root, "");
        return ans;
    }
```

```
    int peakIndexInMountainArray(vector<int>& arr) {
        // for(int  i = 0 ; i < arr.size()-1 ; i++){
        //     if(arr[i] >arr[i+1] ){
        //         cout<<1<<" ";
        //         return i;
        //     }
        // }
        // return 0;

       //Binary Approach
    int lo = 0, hi = arr.size() - 1;
        while (lo < hi) {
            int mi = lo + (hi - lo) / 2;
            if (arr[mi] < arr[mi + 1])
```

```
            lo = mi + 1;
        else
            hi = mi;
    }
    return lo;
}
```

```
// 797. All Paths From Source to Target
    int target;
    vector<vector<int>> res;
    vector<int> tmp;

    void dfs(vector<vector<int>>& graph, int currNode = 0) {
        tmp.push_back(currNode);

        if (currNode == target)
            res.push_back(tmp);
        else
            for (int node: graph[currNode]) {
            dfs(graph, node);
        }

  tmp.pop_back();
    }
    vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
        target = graph.size() - 1;
        dfs(graph);
        return res;
    }

    //bfs
    bool vis(int v, vector<int> &path) {
        for(auto i: path)
            if(i == v)
                return true;
        return false;
    }
    vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
        int src = 0, des = graph.size()-1;
        vector<vector<int>> res;
        vector<int> path;

        path.push_back(src);
        queue<vector<int>> q;
        q.push(path);

        while(!q.empty()){
            path = q.front();
            q.pop();
            int last_val = path.back();
            if(last_val == des)
                res.push_back(path);
```

```
        for(auto v: graph[last_val])
        {
            if(!vis(v, path))
            {
                vector<int> temp(path);
                temp.push_back(v);
                q.push(temp);
            }
        }
    }
    return res;
}
```

//114. Flatten Binary Tree to Linked List

```
void flatten(TreeNode* root) {
    if( root ){
        TreeNode* temp = root->right;
        root->right = root->left;
        root->left = nullptr;
        TreeNode* node = root;

        while( node->right )
            node = node->right;

        node->right = temp;
        flatten( root->right );
    }
    return;
}
```

// 968. Binary Tree Cameras

```
    // Apply a recusion function dfs.
// Return 0 if it's a leaf.
// Return 1 if it's a parent of a leaf, with a camera on this node.
// Return 2 if it's coverd, without a camera on this node.

// For each node,
// if it has a child, which is leaf (node 0), then it needs camera.
// if it has a child, which is the parent of a leaf (node 1), then it's covered.

// If it needs camera, then res++ and we return 1.
// If it's covered, we return 2.
// Otherwise, we return 0.
    int res = 0;
    int minCameraCover(TreeNode* root) {
        return (dfs(root) < 1 ? 1 : 0) + res;
    }

    int dfs(TreeNode* root) {
        if (!root) return 2;
        int left = dfs(root->left), right = dfs(root->right);
        if (left == 0 || right == 0) {
```

```
        res++;
        return 1;
    }
    return left == 1 || right == 1 ? 2 : 0;
}
```

```
int countDays(vector<int>& ws, int tot_cap, int cur_cap = 0, int res = 1) {
  for (auto w : ws) {
    cur_cap += w;
    if (cur_cap > tot_cap) ++res, cur_cap = w;
  }
  return res;
}
int shipWithinDays(vector<int>& ws, int D) {
  auto r = accumulate(begin(ws), end(ws), 0);
  auto l = max(r / D, *max_element(begin(ws), end(ws)));
  while (l < r) {
    auto m = (l + r) / 2;
    if (countDays(ws, m) <= D) r = m;
    else l = m + 1;
  }
  return l;
}
```

```
int snakesAndLadders(vector<vector<int>>& board) {
    int n=board.size();
    vector<vector<bool>> visited(n , vector<bool>(n,false));

    queue<int> q;
    q.push(1);
    visited[n-1][0]=true;
    int steps=0;
    while(!q.empty())
    {
        int size=q.size();
        while(size--)
        {
            int currpos = q.front();
            if(currpos==n*n)
                    return steps;
            q.pop();
            for(int i=1;i<=6;i++)
            {
                int nextpos=currpos+i;
                if(nextpos>n*n)
                    break;
                int r = n - (nextpos-1)/n -1;
                int c = (nextpos-1)%n;
```

```
            if(r%2 == n%2)
                c = n-c-1;

            if(!visited[r][c])
            {
                visited[r][c]=true;
                if(board[r][c]!=-1)
                    q.push(board[r][c]);
                else
                    q.push(nextpos);
            }
        }
    }
    steps++;
    }
    return -1;
}
```

```
//matrix multiply

void mulMat(int mat1[][C1], int mat2[][C2]) {
    int rslt[R1][C2];

    cout << "Multiplication of given two matrices is:\n" << endl;

    for (int i = 0; i < R1; i++) {
        for (int j = 0; j < C2; j++) {
            rslt[i][j] = 0;

            for (int k = 0; k < R2; k++) {
                rslt[i][j] += mat1[i][k] * mat2[k][j];
            }

            cout << rslt[i][j] << "\t";
        }

        cout << endl;
    }
}


//print permu iterative;y


//tower of hanoi
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 0)  return;
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
```

```
    cout << "Move disk " << n << " from rod " << from_rod << " to rod " << to_rod << endl;
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}
```

```
//knight tour
//function to display the 2-d array
void display(vector<vector<int>>& chess) {
  for (int i = 0; i < chess.size(); i++) {
    for (int j = 0; j < chess.size(); j++) {
      cout << chess[i][j] << " ";
    }
    cout << endl;
  }
  cout << endl;
}

void printKnightsTour(vector<vector<int>>& chess, int n, int r, int c, int upcomingMove) {
  //base case
  if (r < 0 || c < 0 || r >= n || c >= n || chess[r][c] != 0) {
    return;
  }
  if (upcomingMove == n * n) {
    chess[r][c] = upcomingMove;
    display(chess);
    chess[r][c] = 0;
    return;
  }
  chess[r][c] = upcomingMove;
  printKnightsTour(chess, n, r - 2, c + 1, upcomingMove + 1);
  printKnightsTour(chess, n, r - 1, c + 2, upcomingMove + 1);
  printKnightsTour(chess, n, r + 1, c + 2, upcomingMove + 1);
  printKnightsTour(chess, n, r + 2, c + 1, upcomingMove + 1);
  printKnightsTour(chess, n, r + 2, c - 1, upcomingMove + 1);
  printKnightsTour(chess, n, r + 1, c - 2, upcomingMove + 1);
  printKnightsTour(chess, n, r - 1, c - 2, upcomingMove + 1);
  printKnightsTour(chess, n, r - 2, c - 1, upcomingMove + 1);
  chess[r][c] = 0;
}
```

```
//Coin Change Combination
// You are required to calculate and print the number of combinations of the n coins using which the amount "amt" c
an be paid.
{
        int[] dp = new int[amt + 1];
        dp[0] = 1;

        for(int coin: coins){
            for(int i = 1; i < dp.length; i++){
                if(i >= coin){
                    dp[i] += dp[i - coin];
                }
            }
```

```
    }
}
```

// You are required to calculate and print the number of permutations of the n coins using which the amount "amt" can be paid.2 + 2 + 3 = 7 and 2 + 3 + 2 = 7 and 3 + 2 + 2 = 7 are different permutations of same combination. You should treat them as 3 and not 1.

```
{
        int[] dp = new int[amt + 1];
    dp[0] = 1;

    for (int i = 1; i < dp.length; i++) {
        for (int coin : coins) {
            if (i >= coin) {
                dp[i] += dp[i - coin];
            }
        }
    }
}
```

//linked list to stack adapter
//linked list to queue adapter
//iterative dfs

//Smallest Substring Of A String Containing All Characters Of Another String
//Count Of Substrings With Exactly K Unique Characters - equivalent subarrays same
//Binary String With Substrings Representing Numbers From 1 To N
// Find All Anagrams In A String

// Construct Binarytree From Preorder And Inorder Traversal Easy
// Construct Binarytree From Postorder And Inorder Traversal Medium
// Construct Binary Tree From Inorder And Levelorder Traversal Medium
// Construct Binary Tree From Preorder And Postorder Traversal Easy
// Construct Bst From Inorder Traversal Easy
// Construct Bst From Preorder Traversal Easy
// Construct Bst From Postorder Traversal Easy
// Construct Bst From Levelorder Traversal

```
// Input:  A = "EACBD", B = "EABCD"
// Output: 3
// Explanation: Pick B and insert at front, EACBD => BEACD
//              Pick A and insert at front, BEACD => ABECD
//              Pick E and insert at front, ABECD => EABCD
int minOps(string &A, string &B)
{
  int m = A.length(), n = B.length();
```

```
    // This parts checks whether conversion is possible or not
    if (n != m)
        return -1;
    int count[256];
    memset(count, 0, sizeof(count));
    // count characters in A
    for (int i = 0; i < n; i++)
        count[A[i]]++;
    // subtract count for every character in B
    for (int i = 0; i < n; i++)
        count[B[i]]--;
    // Check if all counts become 0
    for (int i = 0; i < 256; i++)
        if (count[i])
            return -1;

    // This part calculates the number of operations
    // required
    int res = 0;
    for (int i = n - 1, j = n - 1; i >= 0;)
    {
        // If there is a mismatch, then keep incrementing
        // result 'res' until B[j] is not found in A[0..i]
        while (i >= 0 && A[i] != B[j])
        {
            i--;
            res++;
        }
        // If A[i] and B[j] match
        if (i >= 0)
        {
            i--;
            j--;
        }
    }
    return res;
}
void leftViewUtil(struct Node *root,
                  int level, int *max_level)
{
    if (root == NULL)
        return;
    if (*max_level < level)
    {
        cout << root->data << " ";
        *max_level = level;
    }
    leftViewUtil(root->left, level + 1, max_level);
    leftViewUtil(root->right, level + 1, max_level);
}
void leftView(struct Node *root)
{
    int max_level = 0;
    leftViewUtil(root, 1, &max_level);
}
```

```
vector<int> topView(Node *root)
{
    map<int, int> mp;
    queue<pair<Node *, int>> q;
    q.push({root, 0});
    while (!q.empty())
    {
        auto p = q.front();
        q.pop();
        Node *node = p.first;
        int line = p.second;

        if (mp.find(line) == mp.end())
            mp[line] = node->data;

        if (node->left)
            q.push({node->left, line - 1});
        if (node->right)
            q.push({node->right, line + 1});
    }
    vector<int> ans;
    for (auto it : mp)
    {
        ans.push_back(it.second);
    }
    return ans;
}

void printLeaves(Node *root)
{
    if (root == nullptr)
        return;
    printLeaves(root->left);
    if (!(root->left) && !(root->right))
        cout << root->data << " ";
    printLeaves(root->right);
}
void printBoundaryLeft(Node *root)
{
    if (root == nullptr)
        return;
    if (root->left)
    {
        cout << root->data << " ";
        printBoundaryLeft(root->left);
    }
    else if (root->right)
    {
        cout << root->data << " ";
        printBoundaryLeft(root->right);
    }
}
void printBoundaryRight(Node *root)
{
```

```
    if (root == nullptr)
        return;
    if (root->right)
    {
        printBoundaryRight(root->right);
        cout << root->data << " ";
    }
    else if (root->left)
    {
        printBoundaryRight(root->left);
        cout << root->data << " ";
    }
}
void printBoundary(Node *root)
{
    if (root == nullptr)
        return;
    cout << root->data << " ";
    printBoundaryLeft(root->left);
    printLeaves(root->left);
    printLeaves(root->right);
    printBoundaryRight(root->right);
}

int minimumSwaps(vector<int> &arr, int n)
{
    vector<int> vi;

    vector<pair<int, int>> v;
    for (int i = 0; i < n; i++)
        v.push_back({arr[i], i});
    sort(v.begin(), v.end());

    int vis[n + 10];
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        if (vis[i] || v[i].second == i)
            continue;
        int j = i;
        int cycle_size = 0;
        while (!vis[j])
        {
            vis[j] = 1;
            j = v[j].second;
            cycle_size++;
        }
        if (cycle_size)
            ans += (cycle_size - 1);
    }
    return ans;
}

map<string, int> m;
string formSubtree(Node *root)
```

```
{
    if (root == NULL)
    {
        return "$";
    }
    string s = "";
    if (root->right == NULL && root->left == NULL)
    {
        s = to_string(root->data);
        return s;
    }
    s = s + to_string(root->data);
    s = s + formSubtree(root->left);
    s = s + formSubtree(root->right);
    m[s]++;
    return s;
}
int dupSub(Node *root)
{
    // code here
    formSubtree(root);
    for (auto x : m)
    {
        if (x.second >= 2)
        {
            return true;
        }
    }
    return false;
}

int ma = 0;
int func(Node *root)
{
    if (!root)
        return 0;
    int l = func(root->left);
    int r = func(root->right);
    ma = max(ma, l + r + root->data);
    return l + r + root->data;
}

Node *lca(Node *root, int n1, int n2)
{
    if (!root)
        return NULL;
    if (root->data == n1 || root->data == n2)
        return root;
    Node *l1 = lca(root->left, n1, n2);
    Node *l2 = lca(root->right, n1, n2);
    if (l1 && l2)
        return root;
    if (l1)
        return l1;
    else
```

```
        return l2;
}

int solve(Node *root, int val)
{
    if (!root)
        return 0;
    if (root->data == val)
        return 1;
    int a = solve(root->left, val);
    int b = solve(root->right, val);
    if (!a and !b)
        return 0;
    else
        return a + b + 1;
}
int findDist(Node *root, int a, int b)
{
    Node *LCA = lca(root, a, b);
    int x = solve(LCA, a);
    int y = solve(LCA, b);
    return x + y - 2;
}

Node *kthAncestorDFS(Node *root, int node, int &k)
{
    if (!root)
        return NULL;
    if (root->data == node ||
        (temp = kthAncestorDFS(root->left, node, k)) ||
        (temp =
             kthAncestorDFS(root->right, node, k)))
    {
        if (k > 0)
            k--;
        else if (k == 0)
        {
            cout << "Kth ancestor is: " << root->data;
            return NULL;
        }
        return root;
    }
}

Node *inpre(Node *root)
{
    Node *p = root->left;
    while (p->right)
        p = p->right;
    return p;
}
Node *insuc(Node *root)
{
    Node *p = root->right;
    while (p->left)
```

```
        p = p->left;
    return p;
}
void findPreSuc(Node *root, Node *&pre, Node *&suc, int key)
{
    if (!root)
        return;
    if (root->key == key)
    {
        if (root->left)
            pre = inpre(root);
        if (root->right)
            suc = insuc(root);
        return;
    }
    if (key > root->key)
    {
        pre = root;
        findPreSuc(root->right, pre, suc, key);
    }
    else if (key < root->key)
    {
        suc = root;
        findPreSuc(root->left, pre, suc, key);
    }
}

void func(Node *root, Node *&prev, int &f)
{
    if (!root)
        return;
    func(root->left, prev, f);
    if (prev != NULL and root->data <= prev->data)
    {
        f = 0;
        return;
    }
    prev = root;
    func(root->right, prev, f);
}
bool isBST(Node *root)
{
    int f = 1;
    Node *prev = NULL;
    func(root, prev, f);
    return f;
}
way2 bool isValidBSTHelper(TreeNode *root, long min, long max)
{
    if (root == NULL)
    {
        return true;
    }
    if (root->val > min && root->val < max)
    {
```

```
        return isValidBSTHelper(root->left, min, root->val) &&
               isValidBSTHelper(root->right, root->val, max);
    }
    return false;
}
bool isValidBST(TreeNode *root)
{
    return isValidBSTHelper(root,
                            LONG_MIN, LONG_MAX);
}
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 (isValid(board, i, j, c))
                    {
                        board[i][j] = c;
                        if (solve(board) == true)
                            return true;
                        else
                            board[i][j] = '.';
                    }
                }
                return false;
            }
        }
    }
    return true;
}
bool isValid(vector<vector<char>> &board, int row, int col,
             char c)
{
    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;
}

// mcoloring
const int N = 20;
int color[N];

bool check(int u, int n, int c, bool graph[101][101])
```

```
{
    for (int v = 0; v < n; v++)
    {
        if (u != v && graph[u][v] && color[v] == c)
            return true;
    }
    return false;
}

bool help(int u, int n, int m, bool graph[101][101])
{
    if (u == n)
        return true;
    for (int c = 0; c < m; c++)
    {
        if (check(u, n, c, graph))
            continue;
        color[u] = c;
        if (help(u + 1, n, m, graph))
            return true;
        color[u] = -1;
    }
    return false;
}

bool graphColoring(bool graph[101][101], int m, int n)
{
    memset(color, -1, sizeof(color));
    return help(0, n, m, graph);
}

int evaluatePostfix(string S)
{
    // Your code here
    // Your code here
    stack<int> st;
    for (int i = 0; i < S.length(); i++)
    {
        if (S[i] >= '0' && S[i] <= '9')
            st.push(S[i] - '0');
        else
        {
            int op2 = st.top();
            st.pop();
            int op1 = st.top();
            st.pop();

            switch (S[i])
            {
            case '+':
            {
                st.push(op1 + op2);
                break;
            }
            case '-':
```

```
            {
                st.push(op1 - op2);
                break;
            }
            case '*':
            {
                st.push(op1 * op2);
                break;
            }
            case '/':
            {
                st.push(op1 / op2);
                break;
            }
            }
        }
    }
    return st.top();
}

vector<int> dij()
{
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
    vector<int> d(V, INT_MAX);

    d[S] = 0;
    pq.push({0, S});

    while (!pq.empty())
    {
        pair<int, int> pr = pq.top();
        pq.pop();
        for (auto j : adj[pr.second])
        {
            if (pr.first + j[1] < d[j[0]])
            {
                d[j[0]] = pr.first + j[1];
                pq.push({d[j[0]], j[0]});
            }
        }
    }
    return d;
}

class Solution
{
public:
    int par[1001], sz[1001];
    void make(int i)
    {
        sz[i] = 1;
        par[i] = i;
    }
    int find(int v)
    {
```

```
        if (par[v] == v)
            return v;
        return par[v] = find(par[v]);
    }
    void Union(int a, int b)
    {
        a = find(a);
        b = find(b);
        if (a != b)
        {
            if (sz[a] < sz[b])
                swap(a, b);
            sz[a] += sz[b];
            par[b] = a;
        }
    }
    int spanningTree(int V, vector<vector<int>> adj[])
    {
        int mst = 0;
        for (int i = 0; i < V; i++)
            make(i);
        vector<pair<int, pair<int, int>>> e; //{wt,{u,v}}
        for (int i = 0; i < V; i++)
        {

            for (auto it : adj[i])
            {
                e.push_back(make_pair(it[1], make_pair(i, it[0])));
            }
        }
        sort(e.begin(), e.end());
        for (auto it : e)
        {
            if (find(it.second.first) != find(it.second.second))
            {
                Union(it.second.first, it.second.second);
                mst += it.first;
            }
        }
        return mst;
    }
};
// Prim's Algo

int spanningTree(int V, vector<vector<int>> adj[])
{
    int mst = 0;
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> q; // wt,v
    q.push({0, 0});

    vector<int> d(V, 1e9), vis(V, 0);
    d[0] = 0;
    while (!q.empty())
    {
        int u = q.top().second;
```

```
        int wt = q.top().first;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        mst += wt;
        for (auto it : adj[u])
        {
            int v = it[0];
            int w = it[1];
            if (!vis[v] && d[v] > w)
            {
                d[v] = w;
                q.push({w, v});
            }
        }
    }
    return mst;
}

vector<int> solve(Node *root)
{
    if (!root)
        return {1, 0, INT_MAX, INT_MIN};
    if (!root->left and !root->right)
        return {1, 1, root->data,
                root->data};
    vector<int> l = solve(root->left);
    vector<int> r = solve(root->right);
    if (l[0] and r[0])
    {
        if (root->data > l[3] and root->data < r[2])
        {
            int x = l[2];
            int y = r[3];
            if (x == INT_MAX)
                x = root->data;
            if (y == INT_MIN)
                y = root->data;
            return {l, l[1] + r[1] + 1, x, y};
        }
    }
    return {0, max(l[1], r[1]), 0, 0};
}
int largestBST(Node *root)
{
    vector<int> ans = solve(root);
    return ans[1];
}

// mtrixchain
int f(int i, int j, int arr[], vector<vector<int>> &dp)
{
    if (i == j)
        return 0;
```

```
    if (dp[i][j] != -1)
        return dp[i][j];
    int mini = 1e9;
    for (int k = i; k < j; k++)
    {
        int steps = arr[i - 1] * arr[k] * arr[j] + f(i, k, arr, dp) + f(k + 1, j, arr, dp);
        mini = min(mini, steps);
    }
    return dp[i][j] = mini;
}

int matrixMultiplication(int N, int arr[])
{
    // code here
    vector<vector<int>> dp(N, vector<int>(N, -1));
    return f(1, N - 1, arr, dp);
}

    int dp[201][201];
    int helper(int e,int f)
    {
        if(e<=1 || f<=1)
            return f;

        if(dp[e][f]!=-1)
            return dp[e][f];

        int temp;
        int res(INT_MAX);

        for(int k=1;k<=f;k++)
        {
            temp=max(helper(e-1,k-1),helper(e,f-k))+1;
            res=min(res,temp);
        }
        return dp[e][f]=res;
    }
    int eggDrop(int e, int f)
    {
        memset(dp,-1,sizeof(dp));

        return helper(e,f);
    }


    int maxSquare(int n, int m, vector<vector<int>> mat){
        // code here'
for(int i=1;i<n;i++){
        for(int j=1;j<m;j++)
            if(mat[i][j]) mat[i][j]+=min(mat[i-1][j],min(mat[i][j-1],mat[i-1][j-1]));
      }
      int h=0;
      for(auto i:mat)
        for(auto j:i) h=max(h,j);
      return h;
```

```
    }
```

//word break dp

```cpp
    int dp[1105];
    bool isMatch(string s1,string s2,int i)
    {
        if(s1.substr(i,s2.size())==s2) return true;
        return false;
    }
    int solve(int i,int n,string A,vector<string> &B)
    {
        if(i==n) return 1;
        if(dp[i]!=-1) return dp[i];
        for(int j=0;j<B.size();j++)
        {
            if(isMatch(A,B[j],i))
            {
                if(solve(i+B[j].size(),n,A,B))
                {
                    return dp[i]=1;
                }
            }
        }
        return dp[i]=0;
    }
    int wordBreak(string A, vector<string> &B) {
        //code here
        for(int i=0;i<=A.size();i++)
        dp[i]=-1;
        return solve(0,A.size(),A,B);
    }
```