ASSIGNMENT 5

1. **Maximum XOR of Two Non-Overlapping Sub-trees**

**PROGRAM:**

**class N:**

**def \_init\_(s, v):**

**s.v = v**

**s.c ,s.s= [],0**

**def b\_t(n, e, v):**

**t = [N(v[i]) for i in range(n)]**

**for x, y in e:**

**t[x].c.append(t[y])**

**t[y].c.append(t[x])**

**return t[0]**

**def s\_s(n, p):**

**s = n.v**

**for c in n.c:**

**if c == p:**

**continue**

**s += s\_s(c, n)**

**n.s = s**

**return s**

**def d(n, p, t, s):**

**m = 0**

**for c in n.c:**

**if c == p:**

**continue**

**m = max(m, d(c, n, t, s))**

**if n != p:**

**for x in s:**

**m = max(m, (t - n.s) ^ x)**

**s.add(n.s)**

**return m**

**def x\_t(n, e, v):**

**t = b\_t(n, e, v)**

**t\_s = s\_s(t, None)**

**return d(t, None, t\_s, set())**

**n = 6**

**e = [[0, 1], [0, 2], [1, 3], [1, 4], [2, 5]]**

**v = [2, 8, 3, 6, 2, 5]**

**print(x\_t(n, e, v))**

**OUTPUT:**

1. **Minimum Cuts to Divide a Circle**

**PROGRAM:**

**def m\_c(n):**

**if n == 1:**

**return 0**

**if n % 2 == 0:**

**return n // 2**

**else:**

**return n**

**print(m\_c(4))**

**OUTPUT:**

1. **Difference Between Ones and Zeros in Row and Column**

**PROGRAM:**

**def d\_b\_r\_c(grid):**

**rows = len(grid)**

**cols = len(grid[0])**

**row\_ones = [sum(row) for row in grid]**

**col\_ones = [sum(grid[i][j] for i in range(rows)) for j in range(cols)]**

**result = [[0] \* cols for \_ in range(rows)]**

**for i in range(rows):**

**for j in range(cols):**

**ones = row\_ones[i] + col\_ones[j] - grid[i][j]**

**zeros = (rows - row\_ones[i]) + (cols - col\_ones[j]) - (1 - grid[i][j])**

**result[i][j] = ones - zeros**

**return result**

**grid = [[0, 1, 1],[1, 0, 1],[0, 0, 1]]**

**print(d\_b\_r\_c(grid))**

**OUTPUT:**

1. **Minimum Penalty for a Shop**

**PROGRAM:**

**def m\_p(c):**

**y = c.count('Y')**

**n = len(c) - y**

**min\_p = n**

**p = 0**

**m\_t = 0**

**for i, ch in enumerate(c):**

**if ch == 'Y':**

**p -= 1**

**else:**

**p += 1**

**if p < min\_p:**

**min\_p = p**

**m\_t = i + 1**

**return m\_t**

**print(m\_p("YYNY"))**

**OUTPUT:**

1. **Count Palindromic Sub sequences**

**PROGRAM:**

**def d\_b\_r\_c(grid):**

**rows = len(grid)**

**cols = len(grid[0])**

**row\_ones = [sum(row) for row in grid]**

**col\_ones = [sum(grid[i][j] for i in range(rows)) for j in range(cols)]**

**result = [[0] \* cols for \_ in range(rows)]**

**for i in range(rows):**

**for j in range(cols):**

**ones = row\_ones[i] + col\_ones[j] - grid[i][j]**

**zeros = (rows - row\_ones[i]) + (cols - col\_ones[j]) - (1 - grid[i][j])**

**result[i][j] = ones - zeros**

**return result**

**grid = [[0, 1, 1],[1, 0, 1],[0, 0, 1]]**

**print(d\_b\_r\_c(grid))**

**OUTPUT:**

1. **Find the Pivot Integer**

**PROGRAM:**

**def p\_i(n):**

**t = (n \* (n + 1)) // 2**

**s = 0**

**for i in range(1, n + 1):**

**s += i**

**if s == t - s + i:**

**return i**

**return -1**

**print(p\_i(8))**

**OUTPUT:**

1. **Append Characters to String to Make Subsequent**

**PROGRAM:**

**def a\_c(s, t):**

**i = 0**

**for c in s:**

**if i < len(t) and c == t[i]:**

**i += 1**

**return len(t) - i**

**print(a\_c("coaching", "coding"))**

**OUTPUT:**

1. **Remove Nodes From Linked List**

**PROGRAM:**

**class L:**

**def \_init\_(s, v=0, n=None):**

**s.v = v**

**s.n = n**

**def r\_n(h):**

**d = L(0)**

**d.n = h**

**s = []**

**c = d**

**while c:**

**while s and s[-1].v < c.v:**

**s.pop()**

**s.append(c)**

**c = c.n**

**for i in range(len(s) - 1):**

**s[i].n = s[i + 1]**

**s[-1].n = None**

**return d.n**

**def l\_t\_a(l):**

**a = []**

**while l:**

**a.append(l.v)**

**l = l.n**

**return a**

**h = L(5, L(2, L(13, L(3, L(8)))))**

**r = r\_n(h)**

**print(l\_t\_a(r))**

**OUTPUT:**

1. **Count Sub arrays With Median K**

**PROGRAM:**

**def c\_m\_k(a, k):**

**n = len(a)**

**m\_i = a.index(k)**

**d = {0: 1}**

**b = 0**

**c = 0**

**for i in range(m\_i, n):**

**if a[i] < k:**

**b -= 1**

**elif a[i] > k:**

**b += 1**

**c += d.get(b, 0) + d.get(b - 1, 0)**

**d[b] = d.get(b, 0) + 1**

**d = {0: 1}**

**b = 0**

**for i in range(m\_i - 1, -1, -1):**

**if a[i] < k:**

**b -= 1**

**elif a[i] > k:**

**b += 1**

**c += d.get(-b, 0) + d.get(-b - 1, 0)**

**d[-b] = d.get(-b, 0) + 1**

**return c**

**print(c\_m\_k([3, 2, 1, 4, 5], 4))**

**OUTPUT:**