1. import java.util.\*;
2. import java.math.\*;
3. import java.text.\*;
4. import java.io.\*;
6. public class Main{
7. //Recursion Problems Solution:
9. public static int Sum(int x){
10. if(x == 1)return 1;
11. return x + Sum(x - 1);
12. }
14. public static boolean isSymmetric(int left, int right, int[] arr){
15. if(left >= right)return true;
16. if(arr[left] != arr[right])return false;
18. return isSymmetric(left + 1, right - 1, arr);
19. }
21. public static int factorial(int n){
22. if(n <= 1) return 1;
23. return n \* factorial(n-1);
24. }
26. public static void threeNplusOne(int n){
27. System.out.print(n + " ");
28. if(n == 1) return;
29. if(n % 2 == 1)
30. threeNplusOne(3\*n + 1);
31. else
32. threeNplusOne(n/2);
33. }
35. //Flood-Fill Code:
37. public static void FloodFill(int x, int y, int m, int n, int[][] grid){
38. if(x < 0 || y < 0 || x >= n || y >= m || grid[y][x] == 0)
39. return;
41. grid[y][x] = 0;
43. FloodFill(x - 1, y, m, n, grid);
44. FloodFill(x, y - 1, m, n, grid);
45. FloodFill(x, y + 1, m, n, grid);
46. FloodFill(x + 1, y, m, n, grid);
48. }
50. //Binary-Search Code:
51. public static int binarySearch(int left, int right, int[] Array, int value){
53. if(left > right)return -1;
55. int mid = left + (right - left)/2;
57. if(Array[mid] == value)return mid;
59. if(Array[mid] < value)
60. return binarySearch(mid + 1, right, Array, value);
61. else
62. return binarySearch(left, mid - 1, Array, value);
63. }
65. //Fibonacci Series- naive:
66. public static int fibonacci(int n){
67. if(n <= 1) return n;
68. return fibonacci(n - 1) + fibonacci(n - 2);
69. }

72. //Fibonacci Series-memoization
73. public static int fibonacci(int n, int[] Memo){
74. if(n <= 1)return n;
76. if(Memo[n] != -1)return Memo[n];
78. return Memo[n] = fibonacci(n - 1) + fibonacci(n - 2);
79. }
81. //Tribonacci Naive
82. public static int Tribonacci(int n){
83. if(n <= 2)return n;
85. return Tribonacci(n - 1) - Math.max()
86. }
88. //Merge-Sort
89. public static void MergeSort(int left, int right, int[] Array){
90. if(left >= right)return;
92. int mid = left + (right - left)/2;
94. MergeSort(left, mid, Array);
95. MergeSort(mid + 1, right, Array);
96. Merge(left, mid, right, Array);
97. }
99. static void Merge(int left, int mid, int right, int[] Array){
100. int ln = mid - left;
101. int rn = right - mid + 1;
103. int[] LArr = new int[ln];
104. int[] RArr = new int[rn];
106. for(int i = 0; i < ln; i++)
107. LArr[i] = Array[i + left];
109. for(int i = 0; i < rn; i++)
110. RArr[i] = Array[i + mid + 1];
112. int i = 0, j = 0, k = left;
114. while(i < ln && j < rn){
115. if(LArr[i] <= RArr[j]){
116. Array[k] = LArr[i];
117. i++;
118. }else{
119. Array[k] = RArr[j];
120. j++;
121. }
123. k++;
124. }
126. while(i < ln){
127. Array[k] = LArr[i];
128. k++; i++;
129. }
131. while(j < rn){
132. Array[k] = RArr[j];
133. k++; j++;
134. }
135. }
137. //Binary Strings
138. public static void BString(int i, int n, int Array[]){
139. if(i == n){
140. for(int l: Array)
141. System.out.print(l + " ");
142. System.out.println();
143. return;
144. }
145. Array[i] = 0;
146. BString(i + 1, n, Array);
147. //Backtracking
149. Array[i] = 1;
150. BString(i + 1, n, Array);
152. }
154. //k-Ary Strings
155. public static void kAry(int i, int n, int k, int Array[], int Basket[]){
156. if(i == n){
157. for(int l: Array)
158. System.out.print(l + " ");
159. System.out.println();
160. return;
161. }
163. for(int l = 0; l < k; l++){
164. Array[i] = Basket[l];
165. kAry(i + 1, n, k, Array, Basket);
166. }
167. }
169. //Magnet-Chain Naive:
170. public static int Chain(int i, int j, int gSize){
171. if(j < 0|| j>= gSize)return 0;
173. if(i == gSize)return 1;
175. i++;
177. return Chain(i, j - 1, gSize) +
178. Chain(i, j, gSize) +
179. Chain(i, j + 1, gSize);
180. }
182. //Magnet-Chain Memoization:
183. public static int Chain(int i, int j, int gSize, int[][]Memo){
184. if(j < 0|| j>= gSize)return 0;
186. if(i == gSize)return 1;
188. // -1 signifies "Unvisited"
189. if(Memo[i][j] != -1)return Memo[i][j];
191. i++;
193. return Memo[i][j] = Chain(i, j - 1, gSize) +
194. Chain(i, j, gSize) +
195. Chain(i, j + 1, gSize);
196. }
198. //Maze-Problem Simplified
200. //Naive
201. public static int MazeRat(int i, int j, int m, int n, int[][]Maze){
202. if(i < 0 || j < 0 || i == m || j == n)return 0;
204. if(Maze[i][j] == -1)return 0;
206. if(i == m - 1 && j == n - 1)return 1;
208. return MazeRat(i + 1, j, m, n, Maze) +
209. MazeRat(i, j + 1, m, n, Maze);
210. }
212. //Memoization
213. public static int MazeRat(int i, int j, int m, int n, int[][]Maze, int[][]Memo){
214. if(i < 0 || j < 0 || i == m || j == n)return 0;
216. if(Maze[i][j] == -1)return 0;
218. if(i == m - 1 && j == n - 1)return 1;
220. if(Maze[i][j] != -1)return Memo[i][j];
222. return Memo[i][j] = MazeRat(i + 1, j, m, n, Maze) +
223. MazeRat(i, j + 1, m, n, Maze);
224. }

227. //Gold Collector
229. //Memoization
230. public static int Collect(int i, int n, int Array[], int Memo[]){
231. if(i >= n)return 0;
232. if(Memo[i] != -1) return Memo[i];
234. return Memo[i] = Math.max(Collect(i + 1, n, Array, Memo),
235. Collect(i + 2, n, Array, Memo) + Array[i]);
236. }
238. //Tabulation
239. public static int Collect(int n, int Array){
240. int Table[] = new int[n];
242. Table[0] = Array[0];
243. Table[1] = Math.max(Array[1], Table[0]);
245. for(int i = 2; i < n; i++)
246. Table[i] = Math.max(Table[i - 1], Table[i -2] + Array[i]);
248. return Table[n - 1];
249. }
251. //Permutations
253. //Naive
254. public static void Arrange(int i, int n, int[] Array, int[] Basket){
255. if(i == n){
256. int temp[] = new int[n];
257. for(int l = 0; l < n; l++)
258. temp[i] = Array[i];
260. Arrays.sort(temp);
262. boolean key = true;
264. for(int l = 1; l < n; l++){
265. if(temp[l] == temp[l - 1]){
266. key = false;
267. break;
268. }
269. }
271. if(key){
272. for(int l = 0; l < n; l++)
273. System.out.print(Array[i] + " ");
274. System.out.println();
275. }
277. return;
278. }
280. for(int l = 0; l < n; l++){
281. Array[i] = Basket[l];
282. Arrange(i + 1, n, Array, Basket);
283. }
285. }
287. //Optimized
288. public static void Arrange(int i, int n, int[]Array, int[]Basket){
289. if(i == n){
290. for(int l = 0; l < n; l++)
291. System.out.print(Array[l] + " ");
292. System.out.println();
294. return;
295. }
297. for(int l = 0; l < n; l++){
298. if(Basket[l] == -1)continue;
299. Array[i] = Basket[l];
300. Basket[l] = -1;
301. Arrange(i + 1, n, Array, Basket);
302. Basket[l] = Array[i];
303. }
304. }
306. //More optimized
307. static void swap(int i, int j, int[] arr){
308. int temp = arr[i];
309. arr[i] = arr[j];
310. arr[j] = temp;
311. }
313. public static void Arrange(int i, int n, int[]Basket){
314. if(i == n){
315. for(int l = 0; l < n; l++)
316. System.out.print(Basket[l] + " ");
317. System.out.println();
319. return;
320. }
322. for(int l = i; l < n; l++){
323. swap(l, i, Basket);
324. Arrange(i + 1, n, Basket);
325. swap(l, i, Basket);
326. }
327. }
329. //Maze Problem Revised
330. public static int MazeRat(int i, int j, int m, int n, int Maze[]){
331. if(i < 0 || j < 0 || i == m || j == n)return 0;
333. if(Maze[i][j] < 0)return 0;
335. if(Maze[i][j] == 2) return 1;
337. Maze[i][j] -= 5;
339. int k = 0;
340. k += MazeRat(i - 1, j, m, n, Maze);
341. k += MazeRat(i, j - 1, m, n, Maze);
342. k += MazeRat(i + 1, j, m, n, Maze);
343. k += MazeRat(i, j + 1, m, n, Maze);
345. Maze[i][j] += 5;
346. return k;
347. }
349. //Knight Problem
350. public static int Knight(int i, int j, int m, int n, int Board[], int Moves[][]){
351. if(i < 0 || j < 0 || i >= m || j >= n)return 0;
353. if(Board[i][j] < 0)return 0;
355. if(Board[i][j] == 2) return 1;
357. Board[i][j] -= 5;
359. int k = 0;
360. for(int l = 0; l < 8; l++){
361. int x = Moves[l][0];
362. int y = Moves[l][1];
364. k += Knight(i + y, j + x, m, n, Board, Moves);
365. }
367. Board[i][j] += 5;
369. return k;
370. }
372. //N-queen Problem
373. static void swap(int i, int j, int[] arr){
374. int temp = arr[i];
375. arr[i] = arr[j];
376. arr[j] = temp;
377. }
379. public static void Arrange(int i, int n, int[]Basket){
380. if(i == n){
381. if(isValid(Basket, n)){
382. print(Basket, n);
383. }
385. return;
386. }
388. for(int l = i; l < n; l++){
389. swap(l, i, Basket);
390. Arrange(i + 1, n, Basket);
391. swap(l, i, Basket);
392. }
393. }
395. static boolean isValid(int[] Basket, int n){
397. for(int i = 0; i < n; i++)
398. for(int j = i + 1; j < n; j++)
399. if(Math.abs(i - j) == Math.abs(Basket[i] - Basket[j]))
400. return false;
402. return true;
403. }
405. static void print(int[] Basket, int n){
406. int Board[][] = new int[n][n];
408. for(int[] i: Board)
409. Arrays.fill(i, 0);
411. for(int l = 0; l < n; l++)
412. Board[l][Basket[l]] = 1;
414. for(int i = 0; i < n; i++){
415. for(int j = 0; j < n; j++){
416. System.out.print(Board[i][j] + " ");
417. }
419. System.out.println();
420. }
421. }

424. }