1. public static void FloodFill(int[][] w, int x, int y, int c) {
2. // write code here
3. int n = w.length, m = w[0].length;
4. if(w[x][y] == c) {
5. return;
6. }
7. helper(w, x, y, c, w[x][y], n, m);
8. }
9. public static void helper(int[][] w, int i, int j, int nc, int oc, int n, int m) {
10. if(i < 0 || j < 0 || i >= n || j >= m) {
11. return;
12. }
13. if(w[i][j] != oc) {
14. return;
15. }
16. w[i][j] = nc;
17. helper(w, i + 1, j, nc, oc, n, m);
18. helper(w, i - 1, j, nc, oc, n, m);
19. helper(w, i, j + 1, nc, oc, n, m);
20. helper(w, i, j - 1, nc, oc, n, m);
21. }

[download](https://pastebin.com/dl/7LMXk4r7) [clone](https://pastebin.com/clone/7LMXk4r7) [embed](https://pastebin.com/embed/7LMXk4r7) [print](https://pastebin.com/print/7LMXk4r7) [report](https://pastebin.com/report/7LMXk4r7)

1. public static ArrayList<ArrayList<Integer>> findSubsets(int[] arr, int tar) {
2. // write code here
3. ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
4. ArrayList<Integer> temp = new ArrayList<>();
5. helper(arr, 0, tar, ans, temp);
6. return ans;
7. }
8. static void helper(int[] arr, int idx, int tar, ArrayList<ArrayList<Integer>> ans, ArrayList<Integer> temp) {
9. if(idx == arr.length) {
10. if(tar == 0) {
11. ans.add(new ArrayList<>(temp));
12. }
13. return;
14. }
15. temp.add(arr[idx]);//added 10 in asf;
16. helper(arr, idx + 1, tar - arr[idx], ans, temp);
17. temp.remove(temp.size() - 1);//remove 10 from asf
18. helper(arr, idx + 1, tar, ans, temp);
19. }

22.11.2024

{

        //your code hereds

     int arr[][]={{2,2},{3,4},{3,5}};

     for(int i=0;i<arr.length;i++)

     {

    for(int j=0;j<arr[0].length;j++)

     {

     System.out.print(arr[i][j]);

    }

    }

22.11.2024

1. import java.util.\*;
3. class Solution {
4. public static void coinChange(int[] coins, int amt){
5. //Write your code here
6. String s = "abc";
7. boolean[] vis = new boolean[s.length()];
8. helper(s, 0, "", vis);
9. }
10. public static void helper(String s, int idx, String asf, boolean[] vis) {
12. if(idx == s.length()) {
13. if(asf.length() == s.length()) {
14. System.out.println(asf);
15. }
16. return;
17. }
18. if(vis[idx] == false) {
19. vis[idx] = true;
20. helper(s, 0, asf + s.charAt(idx), vis);
21. vis[idx] = false;
22. }
23. helper(s, idx + 1, asf, vis);
25. }
26. }
28. public class Main {
29. public static void main(String[] args) {
30. Scanner sc = new Scanner(System.in);
31. int n = sc.nextInt();
32. int[] coins = new int[n];
33. for (int i = 0; i < n; i++) {
34. coins[i] = sc.nextInt();
35. }
36. int amt = sc.nextInt();
38. Solution Obj = new Solution();
39. Obj.coinChange(coins, amt);
40. }
41. }
42. import java.util.\*;
43. class Solution {
45. public List<List<Integer>> combinationSum(int[] nums, int target){
46. //write your code here
47. Arrays.sort(nums);
48. List<List<Integer>> ans = new ArrayList<>();
49. helper(nums, target, 0, ans, new ArrayList<>());
50. return ans;
51. }
52. public void helper(int[] nums, int target, int idx, List<List<Integer>> ans, List<Integer> temp) {
53. if(target == 0) {
54. ans.add(new ArrayList<>(temp));
55. return;
56. }
57. if(target < 0 || idx == nums.length) {
58. return;
59. }
61. //place 1
62. //include
63. temp.add(nums[idx]);
64. helper(nums, target - nums[idx], idx, ans, temp);
65. temp.remove(temp.size() - 1);
67. //place 2
68. while(idx + 1 < nums.length && nums[idx] == nums[idx + 1]) {
69. idx++;
70. }//duplicate
71. //exclude
72. helper(nums, target, idx + 1, ans, temp);
74. //place 3
75. }
76. }
77. public class Main {
78. public static void main(String args[]) {
79. Scanner sc = new Scanner(System.in);
80. int n = sc.nextInt();
81. int target = sc.nextInt();
82. int []nums = new int[n];
83. for(int i = 0 ; i < n ; ++i){
84. nums[i] = sc.nextInt();
85. }
86. Solution ob = new Solution();
87. List<List<Integer>> ans = ob.combinationSum(nums,target);
88. for(int i = 0 ; i < ans.size() ; ++i){
89. Collections.sort(ans.get(i));
90. }
91. Collections.sort(ans, (o1, o2) -> {
92. int m = Math.min(o1.size(), o2.size());
93. for (int i = 0; i < m; i++) {
94. if (o1.get(i) == o2.get(i)){
95. continue;
96. }else{
97. return o1.get(i) - o2.get(i);
98. }
99. }
100. return 1;
101. });
102. for (int i = 0; i < ans.size (); i++)
103. {
104. for (int j = 0; j < ans.get(i).size (); j++)
105. {
106. System.out.print(ans.get(i).get(j)+" ");
107. }
108. System.out.println();
110. }

113. }
114. }
115. import java.util.\*;
117. public class Main {
118. public static int totalNQueens(int n) {
119. // write code here
120. return helper(0, 0, n, new boolean[n][n]);
121. }
122. public static int helper(int r, int c, int n, boolean[][] mat) {//r = 0, c = 0, n = 4
123. if(n == 0) {
124. return 1;
125. }
126. if(c == mat[0].length) {
127. return 0;
128. }
129. //place a queen
130. int count = 0;
131. if(safe(r, c, mat)) {
132. mat[r][c] = true;
133. count += helper(r + 1, 0, n - 1, mat);
134. mat[r][c] = false;
135. }
136. //skip the current place (queen not sitting at r, c)
137. count += helper(r, c + 1, n, mat);
138. return count;
139. }
140. public static boolean safe(int r, int c, boolean[][] mat) {
141. for(int i = r - 1, j = c; i >= 0; i--) {
142. if(mat[i][j] == true) {
143. return false;
144. }
145. }
146. for(int i = r, j = c - 1; j >= 0; j--) {
147. if(mat[i][j] == true) {
148. return false;
149. }
150. }
152. for(int i = r - 1, j = c - 1; i >= 0 && j >= 0; i--, j--) {
153. if(mat[i][j] == true) {
154. return false;
155. }
156. }
158. for(int i = r - 1, j = c + 1; i >= 0 && j < mat[0].length; i--, j++) {
159. if(mat[i][j] == true) {
160. return false;
161. }
162. }
163. return true;
165. }
167. public static void main(String[] args) throws Throwable {
168. Scanner sc = new Scanner(System.in);
169. int n = sc.nextInt();
170. System.out.println(totalNQueens(n));
171. }
172. }
173. /\*
174. 4 \* 4
175. - - - -
176. - - - -
177. - - - -
178. - - - -
180. place 4 queens in 4 x 4 matrix
181. n = 4
182. starting from (0, 0)
183. two options
184. 1. place the queen at (0, 0) and ask recursively total ways to place 3 queens from next rows and columns
185. 2. skip this place and ask how many ways are there to place 4 queens in remaining rows and columns
186. \*/

24.11.2024

1. class Employee {
2. private String name;
3. private int age;
5. // Constructor with parameter names same as instance variables
6. public Employee(String cName, int cAge) {
7. this.name = cName; // Refers to the instance variable
8. this.age = cAge; // Refers to the instance variable
9. }
11. public void display() {
12. System.out.println("Name: " + this.name + ", Age: " + this.age);
13. }
14. }
16. public class ThisDemo {
17. public static void main(String[] args) {
18. Employee emp = new Employee("John", 30);
19. emp.display(); // Output: Name: John, Age: 30
20. }
21. }
23. class Student {
24. private String name;
25. private int age;
27. // Constructor 1
28. public Student(String name) {
29. this(name, 18); // Calls Constructor 2
30. }
32. // Constructor 2
33. public Student(String cName, int cAge) {
34. this.name = cName;
35. this.age = cAge;
36. }
38. public void display() {
39. System.out.println("Name: " + this.name + ", Age: " + this.age);
40. }
41. }
43. public class ThisConstructorDemo {
44. public static void main(String[] args) {
45. Student student = new Student("Alice");
46. student.display(); // Output: Name: Alice, Age: 18
47. }
48. }
49. class Student {
50. private String name;
51. private int age;
53. // Constructor 1
54. public Student(String name) {
55. this(name, 18); // Calls Constructor 2
56. }
58. // Constructor 2
59. public Student(String cName, int cAge) {
60. this.name = cName;
61. this.age = cAge;
62. }
64. public void display() {
65. System.out.println("Name: " + this.name + ", Age: " + this.age);
66. }
67. }
69. public class ThisConstructorDemo {
70. public static void main(String[] args) {
71. Student student = new Student("Alice");
72. student.display(); // Output: Name: Alice, Age: 18
73. }
74. }

[download](https://pastebin.com/dl/MZQDLM6M) [clone](https://pastebin.com/clone/MZQDLM6M) [embed](https://pastebin.com/embed/MZQDLM6M) [print](https://pastebin.com/print/MZQDLM6M) [report](https://pastebin.com/report/MZQDLM6M)

1. class Calculator {
2. private String x = "Hi";
3. public void calculate(Display dp) {
4. dp.show(this, x); // Passing the current object/string
5. }
6. }
8. class Display {
9. public void show(Calculator calc, String str) {
10. System.out.println("Address -> " + calc);
11. System.out.println("Displaying resulted string for address -> " + str);
12. }
13. }
15. public class ThisObjectDemo {
16. public static void main(String[] args) {
17. Calculator calc = new Calculator();
18. Display dp = new Display();
19. calc.calculate(dp);
20. }
21. }
23. class Calculator {
24. private String x = "Hi";
25. public void calculate(Display dp) {
26. dp.show(this, x); // Passing the current object/string
27. }
28. }
30. class Display {
31. public void show(Calculator calc, String str) {
32. System.out.println("Address -> " + calc);
33. System.out.println("Displaying resulted string for address -> " + str);
34. }
35. }
37. public class ThisObjectDemo {
38. public static void main(String[] args) {
39. Calculator calc = new Calculator();
40. Display dp = new Display();
41. calc.calculate(dp);
42. }
43. }
44. class Calculator {
45. private String x = "Hi";
46. public void calculate(Display dp) {
47. dp.show(this, x); // Passing the current object/string
48. }
49. }
51. class Display {
52. public void show(Calculator calc, String str) {
53. System.out.println("Address -> " + calc);
54. System.out.println("Displaying resulted string for address -> " + str);
55. }
56. }
58. public class ThisObjectDemo {
59. public static void main(String[] args) {
60. Calculator calc = new Calculator();
61. Display dp = new Display();
62. calc.calculate(dp);
63. }
64. }
66. class Counter {
67. private static int count = 3;
69. public Counter() {
70. count++;
71. }
73. public static int getCount() {
74. return count;
75. }
76. }
78. public class StaticVariableDemo {
79. public static void main(String[] args) {
80. Counter c1 = new Counter();
81. Counter c2 = new Counter();
82. Counter c3 = new Counter();
84. System.out.println("Number of objects created: " + Counter.getCount());
85. // Output: Number of objects created: 3
86. }
88. }
90. class MathUtils {
91. public static int square(int number) {
92. return number \* number;
93. }
94. }
96. public class StaticMethodDemo {
97. public static void main(String[] args) {
98. System.out.println("Square of 5: " + MathUtils.square(5));
99. // Output: Square of 5: 25
100. }
101. }
102. class Config {
103. static String appName;
104. static {
105. appName = "My Application"; // Static initialization block
106. System.out.println("Static block executed");
108. }
109. }
111. public class StaticBlockDemo {
112. public static void main(String[] args) {
113. System.out.println("App Name: " + Config.appName);
114. // Output:
115. // Static block executed
116. // App Name: My Application
117. }
118. }
120. class Person {
121. private String name;
122. private int age;
124. // Constructor using 'this' to differentiate between instance variables and parameters
125. public Person(String name, int age) {
126. this.name = name; // this.name refers to the instance variable
127. this.age = age; // this.age refers to the instance variable
128. }
130. // Method to display the values of name and age
131. public void display() {
132. System.out.println("Name: " + this.name + ", Age: " + this.age);
133. }
134. }
136. public class ThisExample {
137. public static void main(String[] args) {
138. // Creating an object of Person
139. Person person1 = new Person("John", 30);
140. person1.display(); // Output: Name: John, Age: 30
142. // Creating another object of Person
143. Person person2 = new Person("Alice", 25);
144. person2.display(); // Output: Name: Alice, Age: 25
145. }
146. }
147. class Person {
148. private String name;
149. private int age;
151. // Constructor using 'this' to differentiate between instance variables and parameters
152. public Person(String name, int age) {
153. this.name = name; // this.name refers to the instance variable
154. this.age = age; // this.age refers to the instance variable
155. }
157. // Method to display the values of name and age
158. public void display() {
159. System.out.println("Name: " + this.name + ", Age: " + this.age);
160. }
161. }
163. public class ThisExample {
164. public static void main(String[] args) {
165. // Creating an object of Person
166. Person person1 = new Person("John", 30);
167. person1.display(); // Output: Name: John, Age: 30
169. // Creating another object of Person
170. Person person2 = new Person("Alice", 25);
171. person2.display(); // Output: Name: Alice, Age: 25
172. }
173. }
174. class Library {
175. // Static field to keep track of the total number of books
176. public static int totalBooks = 0;
178. // Non-static method to increase the number of books for each instance
179. public void addBooks(int books) {
180. totalBooks += books; // Increase the totalBooks by the number of books added
181. }
183. // Static method to display the total number of books
184. public static void displayTotalBooks() {
185. System.out.println("Total books in library: " + totalBooks);
186. }
187. }
189. public class StaticExample {
190. public static void main(String[] args) {
191. // Creating instances of Library
192. Library lib1 = new Library();
193. Library lib2 = new Library();
195. // Adding books using non-static method
196. lib1.addBooks(10); // Library lib1 added 10 books
197. lib2.addBooks(20); // Library lib2 added 20 books
199. // Displaying the total books in the library using static method
200. Library.displayTotalBooks(); // Output: Total books in library: 30
202. // Creating another instance and adding more books
203. Library lib3 = new Library();
204. lib3.addBooks(5); // Library lib3 added 5 books
206. // Displaying the updated total books
207. Library.displayTotalBooks(); // Output: Total books in library: 35
208. }
209. }
211. // Parent class
212. class Animal {
213. String name;
215. public void eat() {
216. System.out.println(name + " is eating.");
217. }
218. }
220. // Child class
221. class Dog extends Animal {
222. public void bark() {
223. System.out.println(name + " is barking.");
224. }
225. }
227. public class SingleInheritanceDemo {
228. public static void main(String[] args) {
229. Dog dog = new Dog();
230. dog.name = "Buddy"; // Inherited field
231. dog.eat(); // Inherited method
232. dog.bark(); // Child class method
233. }
234. }

**// Grandparent class**

**class Vehicle {**

**public void start() {**

**System.out.println("Vehicle is starting...");**

**}**

**}**

**// Parent class**

**class Car extends Vehicle {**

**public void drive() {**

**System.out.println("Car is driving...");**

**}**

**}**

**// Child class**

**class SportsCar extends Car {**

**public void turbo() {**

**System.out.println("SportsCar is using turbo...");**

**}**

**}**

**public class MultilevelInheritanceDemo {**

**public static void main(String[] args) {**

**SportsCar sportsCar = new SportsCar();**

**sportsCar.start(); // Method from Vehicle**

**sportsCar.drive(); // Method from Car**

**sportsCar.turbo(); // Method from SportsCar**

**}**

**}**