```python
#□□□
def print_values(a, b, c):
    if a > b:
        if b > c:
            # □□ c, b, a
            print(f"{c}, {b}, {a}")
            print(c + b - 10 * a)
        else:
            if a > c:
                # □□ b, c, a
                print(f"{b}, {c}, {a}")
                print(b + c - 10 * a)
            else:
                # □□ b, a, c
                print(f"{b}, {a}, {c}")
                print(b + a - 10 * c)
    else:
        if b > c:
            if a > c:
                # □□ c, a, b
                print(f"{c}, {a}, {b}")
                print(c + a - 10 * b)
            else:
                # □□ a, c, b
                print(f"{a}, {c}, {b}")
                print(a + c - 10 * b)
        else:
            # □□ a, b, c
            print(f"{a}, {b}, {c}")
            print(a + b - 10 * c)

# □□ a=5, b=15, c=10
print("□ a=5, b=15, c=10 □□")
print_values(5, 15, 10)

# □□□□□□□□□□
print("\n□□□□□□□")
print_values(15, 10, 5)   # a>b, b>c
print_values(15, 5, 10)   # a>b, b<=c, a>c
print_values(10, 5, 15)   # a>b, b<=c, a<=c
print_values(5, 10, 15)   # a<=b, b<=c
```

```
42
43 #□□□ □□□DeepSeek □□□□□□□□□ □□□□□□□□
44 import math
45 from functools import lru_cache
46
47
48 @lru_cache(maxsize=None) #□□□□□□□□□□□□
   □□□□debug□□□□□□□□□□□□
49 def continuous_ceiling(x):
50     """
51     □□□□□□□□□ F(x) = F(ceil(x/3)) + 2x, □□ F(1) = 1
52     """
53     if x == 1:
54         return 1
55     return continuous_ceiling(math.ceil(x / 3)) + 2
    * x
56
57
58 def calculate_continuous_ceiling(numbers):
59     """
60     □□□□□□□□□□□□□□□□□□□□
61     """
62     results = []
63     for num in numbers:
64         result = continuous_ceiling(num)
65         results.append((num, result))
66     return results
67
68
69 # □□□□
70 def test_function():
71     # □□□□□□□
72     test_numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15
   , 20]
73
74     print("□□□□□□□ F(x) = F(ceil(x/3)) + 2x, F(1) = 1
   ")
75     print("=" * 50)
76
77     results = calculate_continuous_ceiling(
   test_numbers)
```

```
 78
 79      for num, result in results:
 80          print(f"F({num}) = {result}")
 81
 82      # □□□□□□□□
 83      print("\n□□□□□□:")
 84      print("F(1) = 1")
 85      print("F(2) = F(ceil(2/3)) + 2*2 = F(1) + 4 = 1
     + 4 = 5")
 86      print("F(3) = F(ceil(3/3)) + 2*3 = F(1) + 6 = 1
     + 6 = 7")
 87      print("F(4) = F(ceil(4/3)) + 2*4 = F(2) + 8 = 5
     + 8 = 13")
 88
 89
 90 #□□□
 91
 92 #3.1 □□□□□□□□□□□□□□
 93 def find_number_of_ways(x, n=10, faces=6):
 94      """
 95      □□n□□□□□□□□□x□□□□
 96
 97      □□:
 98      x: □□□□□
 99      n: □□□□ (□□10)
100      faces: □□□□□□□ (□□6)
101
102      □□:
103      □□□
104      """
105      # □□□□□□
106      # dp[i][j] □□□□i□□□□□□□□j□□□□
107      dp = [[0] * (x + 1) for _ in range(n + 1)]
108
109      # □□□□1□□□□□□
110      for j in range(1, min(faces, x) + 1):
111          dp[1][j] = 1
112
113      # □□DP□
114      for i in range(2, n + 1):  # □□□□□2□n
115          for j in range(i, min(i * faces, x) + 1):
```

```
115 # ????i?i*faces
116             for k in range(1, min(faces, j) + 1):
    # ???????
117                 if j - k >= 1:  # ?????
118                     dp[i][j] += dp[i - 1][j - k]
119
120     return dp[n][x]
121
122
123 # ??????????????????
124 def find_number_of_ways_optimized(x, n=10, faces=6):
125     """
126     ??????????????????
127     """
128     if x < n or x > n * faces:
129         return 0
130
131     # ????????
132     dp_prev = [0] * (x + 1)
133     dp_curr = [0] * (x + 1)
134
135     # ???1???
136     for j in range(1, min(faces + 1, x + 1)):
137         dp_prev[j] = 1
138
139     # ??DP?
140     for i in range(2, n + 1):
141         dp_curr = [0] * (x + 1)
142         for j in range(i, min(i * faces, x) + 1):
143             for k in range(1, min(faces + 1, j + 1
    )):
144                 if j - k >= i - 1:  # ??????????????
145                     dp_curr[j] += dp_prev[j - k]
146         dp_prev = dp_curr
147
148     return dp_curr[x] if n > 1 else dp_prev[x]
149
150
151 # ????
152 def test_find_ways():
153     """?????????"""
```

```
154     test_cases = [
155         (10, 1),  # □□□□□□1
156         (11, 10),  # 9□1□1□2
157         (60, 1),  # □□□□□□6
158         (35, 0),  # □□□□□□
159     ]
160 
161     print("□□□□□□□:")
162     print("=" * 30)
163     for x, expected in test_cases:
164         result = find_number_of_ways_optimized(x)
165         print(f"□□□ {x}: {result} □□□")
166 #3.2□□□□□□□□□□□□□□□□
167 # □□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□DeepSeek
168 def analyze_dice_distribution(n=10, faces=6):
169     """
170     □□□□□□□□□□□□□□□□
171 
172     □□:
173     number_of_ways: □□□□□□□□□□□□□□□□□□
174     max_x: □□□□□□□□□□□□
175     max_ways: □□□□□
176     """
177     min_sum = n * 1  # □□□□□
178     max_sum = n * faces  # □□□□□
179 
180     # □□□DP□
181     dp = [[0] * (max_sum + 1) for _ in range(n + 1)]
182 
183     # □□□1□□□
184     for j in range(1, faces + 1):
185         dp[1][j] = 1
186 
187     # □□DP□
188     for i in range(2, n + 1):
189         for j in range(i, i * faces + 1):
190             for k in range(1, min(faces + 1, j + 1
    )):
191                 if j - k >= i - 1:
192                     dp[i][j] += dp[i - 1][j - k]
193 
```

```
194     # □□10□60□□□□
195     number_of_ways = []
196     max_ways = 0
197     max_x = 0
198 
199     for x in range(min_sum, max_sum + 1):
200         ways = dp[n][x]
201         number_of_ways.append(ways)
202 
203         if ways > max_ways:
204             max_ways = ways
205             max_x = x
206 
207     return number_of_ways, max_x, max_ways
208 
209 
210 # □□□
211 if __name__ == "__main__":
212     # 3.1 □□
213     test_find_ways()
214 
215     print("\n" + "=" * 50)
216     print("3.2 □□□□□□□")
217     print("=" * 50)
218 
219     # 3.2 □□□□
220     number_of_ways, max_x, max_ways =
    analyze_dice_distribution()
221 
222     print(f"□□□□: 10 - 60")
223     print(f"□□□□□□□□□□: {max_x}")
224     print(f"□□□□: {max_ways:,}")
225 
226     # □□□□□□
227     print("\n□10□□□□□□□□□:")
228     for i in range(10):
229         x = i + 10
230         print(f"□□ {x}: {number_of_ways[i]:,} □□")
231 
232     print("\n□□10□□□□□□□□□:")
233     for i in range(20, 30):
```

```
234         x = i + 10
235         print(f"□□□ {x}: {number_of_ways[i]:,} □□□")
236 
237     print("\n□□10□□□□□□□□□□:")
238     for i in range(41, 51):
239         x = i + 10
240         print(f"□□□ {x}: {number_of_ways[i]:,} □□□")
241 
242     # □□□□
243     total_ways = sum(number_of_ways)
244     expected_total = 6 ** 10
245     print(f"\n□□:")
246     print(f"□□□□□□□: {total_ways:,}")
247     print(f"6^10 = {expected_total:,}")
248     print(f"□□{'□□' if total_ways == expected_total
    else '□□'}")
249 
250 #□□□□□□□□□□□□□□□□□□□□□□□□□□□4.3□□□□□□□□□□□□□DeepSeek
251 
252 import random
253 
254 import matplotlib.pyplot as plt
255 
256 
257 # 4.1 □□□□□□
258 def random_integer(N):
259     return [random.randint(0, 10) for _ in range(N)]
260 
261 
262 # 4.2 □□□□□□□□□□□
263 def sum_averages(arr):
264     n = len(arr)
265     total = 0
266 
267     # □□□□□□□□
268     for i in range(1, 1 << n):  # □1□□□□□□□
269         subset = []
270         for j in range(n):
271             if i & (1 << j):  # □□□j□□□□1
272                 subset.append(arr[j])
273         total += sum(subset) / len(subset)
```

```
274
275     return total
276
277
278 # 4.3 □□□□
279 def analyze_trend():
280     total_sum_averages = []
281
282     for N in range(1, 101):
283         arr = random_integer(N)
284         result = sum_averages(arr)
285         total_sum_averages.append(result)
286
287         if N % 20 == 0:  # □20□□□□□
288             print(f"N={N}: {result:.2f}")
289
290     return total_sum_averages
291
292
293 # □□□
294 if __name__ == "__main__":
295     # □□□□□
296     test_arr = [1, 2, 3]
297     print(f"□□□□: {test_arr}")
298     print(f"Sum_averages: {sum_averages(test_arr)}")
299
300     print("\n□□N□1□100...")
301     results = analyze_trend()
302
303     # □□
304     plt.figure(figsize=(10, 6))
305     plt.plot(range(1, 101), results, 'b-')
306     plt.xlabel('□□□□ N')
307     plt.ylabel('Sum_averages')
308     plt.title('Sum_averages □N□□□□□')
309     plt.grid(True)
310     plt.show()
311
312     print(f"\n□□□: {min(results):.2f}")
313     print(f"□□□: {max(results):.2f}")
314 #□□□
```

```python
import random


# 5.1 □□□□
def create_matrix(N, M):
    matrix = [[random.randint(0, 1) for _ in range(M)] for _ in range(N)]
    matrix[0][0] = 1
    matrix[N - 1][M - 1] = 1
    return matrix


# 5.2 □□□□□
def count_path(matrix):
    N, M = len(matrix), len(matrix[0])
    dp = [[0] * M for _ in range(N)]
    dp[0][0] = 1

    for i in range(N):
        for j in range(M):
            if matrix[i][j] == 0:
                continue
            if i > 0:
                dp[i][j] += dp[i - 1][j]
            if j > 0:
                dp[i][j] += dp[i][j - 1]

    return dp[N - 1][M - 1]


# 5.3 □□1000□
N, M = 10, 8
total = 0

for _ in range(1000):
    mat = create_matrix(N, M)
    total += count_path(mat)

print(f"□□□□□: {total / 1000:.2f}")
```