TÀI LIỆU ĐƯỢC PHÉP SỬ DỤNG CHO MÔN THI TOÁN RỜI RẠC

```
m, n = 4, 3
counter = 0
for i in range(1, m+1):
    for j in range(1, n+1):
        counter += 1
        print(counter, i, j)
```

Mã 1: Ví dụ 1.7

```
n = 4
counter = 0
for i in range(1, n+1):
    for j in range(1, i+1):
        counter += 1

print(counter)
```

Mã 2: Ví dụ 1.8

```
from sympy import *

n, i = symbols('n i')
Sum(i, (i, 1, n)).doit().simplify()
```

Mã 3: Ví dụ 1.8

```
A = [[ 2,  0, -1],
     [ 1,  3, -2]]
B = [[ 0, -1,  1, 0],
     [ 2,  3, -1, 4],
     [-3,  0, -2, 1]]
```

Mã 4: Ví dụ 1.9

```
def matrix_mul(A, B):
    m, n = len(A), len(B)
    p = len(B[0])
    C = [[0 for j in range(p)] for i in range(m)]
    for i in range(m):
        for j in range(p):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

```
matrix_mul(A, B)
```

Mã 5: Ví dụ 1.9

```
import numpy as np

np.dot(A, B)
```

Mã 6: Ví dụ 1.9

```
def BubbleSort(x):
    n = len(x)
    for i in range(n-1):
        for j in range(n-1,i,-1):
            if x[j] < x[j-1]:
                x[j-1], x[j] = x[j], x[j-1]
    return x

BubbleSort([7, 9, 2, 5, 8])
```

Mã 7: Ví dụ 1.10

```
def factorial(n):
    p = 1
    for i in range(1, n+1):
        p *= i
    return p

factorial(8)
```

Mã 8: Ví dụ 1.13

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

factorial(8)
```

Mã 9: Ví dụ 1.13

```
from sympy import *

factorial(8)
```

Mã 10: Ví dụ 1.13

```python
def permutations(a):
    n = len(a)
    if n == 1:
        return [a]
    P = []
    for i in range(n):
        b = a.copy()
        x = b.pop(i)
        for p in permutations(b):
            p = [x] + p
            P.append(p)
    return P
```

Mã 11: Ví dụ 1.14

```python
import itertools

list( itertools.permutations([1, 2, 3]) )
```

Mã 12: Ví dụ 1.14

```python
def binary_strs(n):
    if n==1:
        return ['0', '1']
    A = []
    for s in binary_strs(n-1):
        A.append('0' + s)
    for s in binary_strs(n-1):
        A.append('1' + s)
    return A

binary_strs(3)
```

Mã 13: Ví dụ 1.15

```python
import itertools
list(itertools.product([0, 1], repeat=3))
```

Mã 14: Ví dụ 1.15

```python
def P(n, r):
    p = 1
    for i in range(r):
        p *= n - i
    return p
```

```
P(8, 5)
```

Mã 15: Ví dụ 1.16

```
from sympy import *

n, r = 8, 5
factorial(n) / factorial(n-r)
```

Mã 16: Ví dụ 1.16

```
def permutations(a, r):
    if r == 1:
        return [[i] for i in a]
    P = []
    n = len(a)
    for i in range(n):
        b = a.copy()
        x = b.pop(i)
        for p in permutations(b, r-1):
            p = [x] + p
            P.append(p)
    return P

permutations([1, 2, 3, 4], 3)
```

Mã 17: Ví dụ 1.17

```
import itertools

list( itertools.permutations([1, 2, 3, 4], 3) )
```

Mã 18: Ví dụ 1.17

```
from sympy import *

binomial(10, 4)
```

Mã 19: Ví dụ 1.18

```
def binomial(n, r):
    p = 1
    for i in range(r):
        p = p * (n-i) // (i+1)
    return p
```

```
binomial(10, 4)
```

Mã 20: Ví dụ 1.18

```
def combinations(a, r):
    if r == 1:
        return [[i] for i in a]
    n = len(a)
    if r == n:
        return [a]
    C = []
    for c in combinations(a[1:], r-1):
        c = [a[0]] + c
        C.append(c)
    for c in combinations(a[1:], r):
        C.append(c)
    return C

combinations([1, 2, 3, 4, 5], 3)
```

Mã 21: Ví dụ 1.19

```
import itertools

list( itertools.combinations([1, 2, 3, 4, 5], 3) )
```

Mã 22: Ví dụ 1.19

```
from sympy import *

x, y = symbols('x y')
( (x + y)**2 ).expand()
```

Mã 23: Ví dụ 1.21

```
def binomial(n, r):
    if r == 0 or r == n:
        return 1
    return binomial(n-1, r-1) + binomial(n-1, r)

binomial(10, 4)
```

Mã 24: Định lý 1.2

```
def binomial(n, r):
    a = [1]
    for i in range(1, n+1):
```

```
        for j in range(i-1, 0, -1):
            a[j] += a[j-1]
        a.append(1)
    return a[r]

binomial(10, 4)
```

Mã 25: Định lý 1.2

```
def permutations_with_replacement(a, n):
    r = len(a)
    if sum(n) == 0:
        return [[]]
    P = []
    for i in range(r):
        if n[i] > 0:
            n_ = n.copy()
            n_[i] -= 1
            for p in permutations_with_replacement(a, n_):
                p = [a[i]] + p
                P.append(p)
    return P

permutations_with_replacement(['A', 'B', 'L'], [1, 1, 2])
```

Mã 26: Ví dụ 1.23

```
def walks(a, b, x, y):
    if a == x:
        return ['U' * (y-b)]
    if b == y:
        return ['R' * (x-a)]
    W = []
    for w in walks(a+1, b, x, y):
        w = 'R' + w
        W.append(w)
    for w in walks(a, b+1, x, y):
        w = 'U' + w
        W.append(w)
    return W
```

Mã 27: Ví dụ 1.24

```
from sympy import *
```

```
x, y, z = symbols('x y z')
( (x + y + z)**7 ).expand()

( (x + y + z)**7 ).expand().coeff(x * y**5 * z)

a, b, c = symbols('a b c')
expr = (a - 2*b + 3*c + 5)**10
expr.expand().coeff(a**4 * b * c**3)
```

Mã 28: Ví dụ 1.25

```
import itertools

list( itertools.combinations_with_replacement(['A', 'B', 'C'], 4) )
```

Mã 29: Ví dụ 1.26

```
def combinations_with_replacement(a, r):
    n = len(a)
    if n == 1:
        return [a * r]
    if r == 1:
        return [[i] for i in a]
    C = []
    for c in combinations_with_replacement(a, r-1):
        c = [a[0]] + c
        C.append(c)
    for c in combinations_with_replacement(a[1:], r):
        C.append(c)
    return C

combinations_with_replacement(['A', 'B', 'C'], 4)
```

Mã 30: Ví dụ 1.26

```
from sympy import *

r = symbols('r')
Sum(binomial(6+r-1, r), (r, 0, 9)).doit()
```

Mã 31: Ví dụ 1.29

```
def summands(n):
    if n == 1:
        return [[1]]
    S = []
```

```
    for i in range(1, n):
        for s in summands(n-i):
            s = [i] + s
            S.append(s)
    S.append([n])
    return S

summands(3)
```

Mã 32: Ví dụ 1.30

```
def compare(a, b):
    m, n = len(a), len(b)
    i = 0
    while i < m and i < n and a[i] == b[i]:
        i += 1
    if i == m == n:
        return('=')
    if i == m < n:
        return('<')
    if i == n < m:
        return('>')
    if i < m and i < n:
        if a[i] < b[i]:
            return('<')
        else:
            return('>')

compare([4, 1, 2], [4, 1, 2, 3])

compare([3, 1, 4], [3, 1, 2, 5])
```

Mã 33: Ví dụ 1.31

```
def next_permutations(a):
    n = len(a)
    k = n - 1
    while k >= 1 and a[k-1] > a[k]:
        k -= 1

    if k == 0:
        return None

    i = n - 1
```

```
    while a[i] < a[k-1]:
        i -= 1
    a[k-1], a[i] = a[i], a[k-1]

    b = a[k:]
    b.reverse()
    return a[:k] + b

next_permutations([3, 6, 2, 5, 4, 1])
```

Mã 34: Ví dụ 1.32

```
def next_combinations(n, a):
    r = len(a)
    i = r - 1
    while i >= 0 and a[i] == n - r + (i + 1):
        i -= 1

    if i == -1:
        return None

    return a[:i] + [a[i] + j for j in range(1, r-i+1)]

next_combinations(6, [1, 2, 5, 6])
```

Mã 35: Ví dụ 1.33

```
def next_bin_str(a):
    n = len(a)
    i = n - 1
    while i >=0 and a[i] == 1:
        i -= 1

    if i == -1:
        return None

    for j in range(i, n):
        a[j] = 1 - a[j]
    return a

a = [1, 0, 0, 0, 1, 0, 0, 1, 1, 1]
next_bin_str(a)
```

Mã 36: Ví dụ 1.34

```
def catalan_walks(a, b, n):
    if a == n:
        return ['U' * (n-b)]
    W = []
    if a == b:
        for w in catalan_walks(a+1, b, n):
            w = 'R' + w
            W.append(w)
    if a > b:
        for w in catalan_walks(a+1, b, n):
            w = 'R' + w
            W.append(w)
        for w in catalan_walks(a, b+1, n):
            w = 'U' + w
            W.append(w)
    return W

catalan_walks(0, 0, 3)
```

Mã 37: Ví dụ 1.35

```
from sympy import *

[binomial(2*n, n) / (n+1) for n in range(11)]
```

Mã 38: Ví dụ 1.35

```
def binary_arrays(n):
    if n == 1:
        return [[True], [False]]
    A = []
    for a in binary_arrays(n-1):
        a = [True] + a
        A.append(a)
    for a in binary_arrays(n-1):
        a = [False] + a
        A.append(a)
    return A

binary_arrays(2)

from sympy import *

p, q, r = symbols('p q r')
```

```
P = p >> (~q & r) | False

for p_, q_, r_ in binary_arrays(3):
    print(p_, q_, r_, P.subs({p: p_, q: q_, r: r_}))
```

Mã 39: Ví dụ 2.4

```
import ttg

ttg.Truths(
    ['p', 'q', 'r'],
    ['p => (~q and r) or False']
).as_pandas()
```

Mã 40: Ví dụ 2.4

```
from sympy import *

p, q, r, s, t, u = symbols('p q r s t u')
P = (p >> q) & (q >> r & s) & (~r | ~t | u) & (p & t)
P.simplify()
```

Mã 41: Ví dụ 2.10

```
def is_prime(n):
    for k in range(2, n):
        if n % k == 0:
            return False
    return True

is_prime(7)
```

Mã 42: Ví dụ 2.14

```
def is_composite(n):
    for k in range(2, n):
        if n % k == 0:
            return True
    return False

is_composite(7)
```

Mã 43: Ví dụ 2.15

```
def power_set(a: list):
    if len(a) == 0:
        return [[]]
```

```
    P = [[]]
    for s in power_set(a[1:]):
        s = [a[0]] + s
        P.append(s)
    for s in power_set(a[1:])[1:]:
        P.append(s)
    return P

power_set([1, 2, 3])
```

Mã 44: Ví dụ 3.4

```
from sympy import *

n, i = symbols('n i')

Sum(i**2, (i, 1, n)).doit().factor()
(n*(n+1)*(2*n+1)/6 + (n+1)**2).factor()
```

Mã 45: Ví dụ 4.4

```
for n in range(1, 6):
    a = [2*i - 1 for i in range(1, n+1)]
    print(a, sum(a))
```

Mã 46: Ví dụ 4.6

```
count = 0
n = 1
while count <= 3:
    print(n, 4*n, n**2 - 7)
    n += 1
    if 4*n < n**2 - 7:
        count += 1

count
```

Mã 47: Ví dụ 4.7

```
def summands(n):
    if n == 1:
        return [[1]]
    L = []
    for x in summands(n - 1):
        y = x.copy()
        x.append(1)
```

```
        L.append(x)
        y[-1] += 1
        L.append(y)
    return L

summands(4)
```

Mã 48: Ví dụ 4.8

```
from sympy import *

gcd(287, 91)
```

Mã 49: Ví dụ 4.25

```
def gcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a

gcd(287, 91)
```

Mã 50: Ví dụ 4.25

```
from sympy import *

gcdex(287, 291)
```

Mã 51: Ví dụ 4.27

```
def gcdex(a, b):
    x0, y0 = 1, 0
    x1, y1 = 0, 1
    while b != 0:
        q = a // b
        a, b = b, a % b
        x = x0 - x1 * q
        y = y0 - y1 * q
        x0, y0 = x1, y1
        x1, y1 = x, y
    return x0, y0, a

gcdex(287, 291)
```

Mã 52: Ví dụ 1.

```
from sympy import *

lcm(456, 168)
```

Mã 53: Ví dụ 4.29

```
from sympy import *

factorint(980220)
```

Mã 54: Ví dụ 4.33

```
def factorint(n):
    i = 2
    f = {}
    while n > 1:
        while n % i != 0:
            i += 1
        e = 0
        while n % i == 0:
            n //= i
            e += 1
        f[i] = e
    return f

factorint(980220)
```

Mã 55: Ví dụ 4.33

```
a = [1, 7, 0, 0, 3]
b = 8

n = 0
k = len(a)
for i in range(k-1, -1, -1):
    n = n * b + a[i]

n
```

Mã 56: Ví dụ 4.38

```
n, b = 6137, 8

a = []
while n != 0:
    r = n % b
```

```
    a.append(r)
    n //= b

a
```

Mã 57: Ví dụ 4.39

```
a = [6, 4, 2, 7]
b = [5, 3, 7, 4]
base = 8

k = len(a)
r = 0
s = [0] * (k+1)
for i in range(k):
    t = a[i] + b[i] + r
    s[i] = t % base
    r = t // base
s[k] = r

s
```

Mã 58: Ví dụ 4.40

```
a = [2, 4, 3]
b = [3, 2, 1, 4]
base = 5

k = len(a)
l = len(b)

s = [0] * (k+l)
for i in range(l):
    r = R = 0
    for j in range(k):
        t = a[j] * b[i] + r
        p = t % base
        r = t // base
        t = s[i+j] + p + R
        s[i+j] = t % base
        R = t // base
    s[k+i] = r + R
```

```
s
```

Mã 59: Ví dụ 4.41

```
from sympy import *

floor(3.8)
ceiling(3.7)
```

Mã 60: Hàm sàn, trần

```
def luy_thua(a, n):
    x = 1
    while n != 0:
        if n % 2 == 1:
            x = x * a
        a = a * a
        n = n // 2
    return x
```

Mã 61: Ví dụ 5.69

```
import numpy as np
M1 = np.array([[0, 1, 0, 0],
               [0, 1, 0, 0],
               [0, 0, 1, 1],
               [0, 0, 0, 0]], dtype=bool)
M2 = np.array([[1, 0, 0],
               [0, 1, 0],
               [0, 0, 0],
               [0, 0, 0]], dtype=bool)
M = M1.dot(M2)
np.array(M, dtype=int)
```

Mã 62: Ví dụ 6.16

```
import numpy as np
M = np.array([[0, 1, 1, 0],
               [0, 0, 0, 1],
               [0, 1, 0, 0],
               [0, 0, 0, 0]], dtype=bool)

n = len(M)
L = M.copy()
for k in range(2, n+1):
    L = M.dot(L)
```

```
10      print(k, np.array(L, dtype=int))
```

Mã 63: Ví dụ 6.17

```
1  import numpy as np
2  M = np.array([[0, 1, 0, 1],
3                [0, 1, 0, 0],
4                [1, 0, 0, 0],
5                [0, 0, 1, 0]], dtype=bool)
6  n = len(M)
7  A = M.copy()
8  L = M.copy()
9  for k in range(2, n+1):
10     L = M.dot(L)
11     A += L
12     print(k, np.array(L, dtype=int))
13 np.array(A, dtype=int)
```

Mã 64: Ví dụ 6.42

```
1  import numpy as np
2  W = np.array([[0, 1, 0, 1],
3                [0, 1, 0, 0],
4                [1, 0, 0, 0],
5                [0, 0, 1, 0]], dtype=bool)
6  n = len(W)
7  for k in range(n):
8      for i in range(n):
9          for j in range(n):
10             W[i, j] += W[i, k] * W[k, j]
11     print(k+1, np.array(W, dtype=int))
```

Mã 65: Ví dụ 6.43

```
1  from sympy import *
2  n = symbols('n')
3  F = symbols('F', cls=Function)
4  rsolve(
5      -F(n) + F(n-1) + F(n-2),
6      F(n),
7      {F(0): 0, F(1): 1}
8  )
```

Mã 66: Ví dụ 9.2

```
a = 100
for _ in range(3):
    a = a + a * 5 / 100
    print(a)
```

Mã 67: Ví dụ 9.3

```
from sympy import *

n = symbols('n')
a = symbols('a', cls=Function)

sol = rsolve(
    a(n) - 1.05*a(n-1),
    a(n),
    {a(0): 100}
)
sol
```

Mã 68: Ví dụ 9.3

```
from sympy import *

n, P, r, S = symbols('n P r S')
a = symbols('a', cls=Function)
sol = rsolve(
    -a(n) + (1+r) * a(n-1) - P,
    a(n),
    {a(0): S}
)
sol

sol.simplify()

sol_P = solve(sol, P)
sol_P

sol_P[0]

sol_P[0].subs({S: 100, r: 0.01, n: 12})
```

Mã 69: Ví dụ 9.4

```
def summands(n):
    if n == 1:
```

```
        return [[1]]
    S = []
    for s in summands(n-1):
        s[0] += 1
        S.append(s)
    for s in summands(n-1):
        s = [1] + s
        S.append(s)
    return S

summands(3)
```

Mã 70: Ví dụ 9.5

```
def BubleSort(x):
    n = len(x)
    if n == 1:
        return x
    for i in range(n-1, 0, -1):
        if x[i] < x[i-1]:
            x[i], x[i-1] = x[i-1], x[i]
    return [x[0]] + BubleSort(x[1:])

BubleSort([7, 9, 2, 5, 8])
```

Mã 71: Ví dụ 9.6

```
def hanoi_tower(n, A, B, C):
    if n == 1:
        return [[1, A, B]]
    return hanoi_tower(n-1, A, C, B) + [[n, A, B]] + hanoi_tower(n
   -1, C, B, A)

hanoi_tower(3, 'A', 'B', 'C')
```

Mã 72: Ví dụ 9.8

```
def quaternary_strs(n):
    if n == 1:
        return [[i] for i in range(4)]
    S = []
    for i in range(4):
        for s in quaternary_strs(n-1):
            s = [i] + s
            S.append(s)
```

```
    return S

quaternary_strs(2)

def quaternary_strs_1s_even(n):
    if n == 1:
        return [[0], [2], [3]]
    S = []
    for s in quaternary_strs(n-1):
        if s not in quaternary_strs_1s_even(n-1):
            s = [1] + s
            S.append(s)
    for i in [0, 2, 3]:
        for s in quaternary_strs_1s_even(n-1):
            s = [i] + s
            S.append(s)
    return S

quaternary_strs_1s_even(2)
```

Mã 73: Ví dụ 9.9

```
from sympy import *

n = symbols('n')
a = symbols('a', cls=Function)
rsolve(
    -a(n) + 3*a(n-2) - 2*a(n-3),
    a(n)
)

rsolve(
    -a(n) + 3*a(n-2) - 2*a(n-3),
    a(n),
    {a(0): 5, a(1): -1, a(2): 2}
)
```

Mã 74: Ví dụ 9.12

```
from sympy import *

x = symbols('x')
P = x**3 - 3*x + 2
P.factor()
```

```
C1, C2, C3 = symbols('C1 C2 C3')
ans = C1 + C2*n + (-2)**n * C3

[ans.subs(n, i) for i in [0, 1, 2]]

eqns = [ans.subs(n, i) - a for i, a in zip([0, 1, 2], [5, -1, 2])]
solve(eqns)
```

Mã 75: Ví dụ 9.12

```
from sympy import *

n = symbols('n', integer=True)
a = symbols('a', cls=Function)
ans = rsolve(
    a(n+1) - 2*a(n) + 2*a(n-1),
    a(n),
    {a(0): 1, a(1): 2}
)
ans

polar = lambda z: abs(z) * E**(I*arg(z))

print(ans)

ans = (Rational(1, 2) + I/2) * polar(1 - I)**n + (Rational(1, 2) -
   I/2) * polar(1 + I)**n
ans

re(ans)
```

Mã 76: Ví dụ 9.13

```
from sympy import *

x = symbols('x')
P = x**2 - 2*x + 2
solve(P)

x = 1 + I
r = abs(x)
phi = arg(x)
```

```
C1, C2 = symbols('C1 C2')
ans = r**n * (C1 * cos(n*phi) + C2 * sin(n*phi))

[ans.subs(n, i).simplify() for i in [0, 1]]

solve([ans.subs(n, i).simplify() - a for i, a in zip([0, 1], [1, 2])])
```

Mã 77: Ví dụ 9.13

```
def subsets_with_condition(n):
    if n == 1:
        return [[], [1]]
    if n == 2:
        return [[], [1], [2]]
    S = []
    for s in subsets_with_condition(n-1):
        S.append(s)
    for s in subsets_with_condition(n-2):
        s.append(n)
        S.append(s)
    return S

subsets_with_condition(3)
```

Mã 78: Ví dụ 9.14

```
def binary_strs(n):
    if n == 1:
        return ['0', '1']
    if n == 2:
        return ['01', '10', '11']
    S = []
    for s in binary_strs(n-2):
        s = '01' + s
        S.append(s)
    for s in binary_strs(n-1):
        s = '1' + s
        S.append(s)
    return S

binary_strs(3)
```

Mã 79: Ví dụ 9.16

```
from sympy import *

n = symbols('n')
a = symbols('a', cls=Function)
ans = rsolve(
    a(n) - 2*a(n-2),
    a(n),
    {a(1): 1, a(2): 2}
)
ans

k = symbols('k', integer=True)

ans.subs(n, 2*k).simplify()
ans.subs(n, 2*k+1).simplify()
```

Mã 80: Ví dụ 9.18

```
def symmetric_summands(n):
    if n == 1:
        return [[1]]
    if n == 2:
        return [[2], [1, 1]]
    S = []
    for s in symmetric_summands(n-2):
        s[0] += 1
        s[-1] += 1
        S.append(s)
    for s in symmetric_summands(n-2):
        s = [1] + s + [1]
        S.append(s)
    return S

symmetric_summands(4)
```

Mã 81: Ví dụ 9.18