排序模板：

```
list_ = [(2,2),(1,2),(1,3)]

list_.sort(key=lambda x: (x[0],x[1]),reverse=True)

print(list_)

#list_ = [(2, 2), (1, 3), (1, 2)]
```

dfs 模板：

```
dx = [-1, 0, 1, 0]
dy = [ 0, 1, 0, -1]


def dfs(maze, x, y):
    global cnt


    for i in range(4):
        nx = x + dx[i]
        ny = y + dy[i]


        if maze[nx][ny] == 'e':
            cnt += 1
            continue


        if maze[nx][ny] == 0:
            maze[x][y] = 1
            dfs(maze, nx, ny)
            maze[x][y] = 0
    return


n, m = map(int, input().split())
maze = []
maze.append( [-1 for x in range(m+2)] )
for _ in range(n):
    maze.append([-1] + [int(_) for _ in input().split()] + [-1])
maze.append( [-1 for x in range(m+2)] )


maze[1][1] = 's'
maze[n][m] = 'e'


cnt = 0
dfs(maze, 1, 1)
print(cnt)
```

bfs 模板：

```
from collections import deque

def bfs(n):

    inq = set()

    inq.add(1)

    q = deque()

    q.append((0, 1))

    while q:
        step, front = q.popleft()

        if front == n:

            return step

        if front * 2 <= n and front * 2 not in inq:

            inq.add(front * 2)

            q.append((step + 1, front * 2))

        if front + 1 <= n and front + 1 not in inq:

            inq.add(front + 1)

            q.append((step + 1, front + 1))




n = int(input())

print(bfs(n))
```

爆栈使用：

```
import sys

sys.setrecursionlimit(20000)
```

二分查找模板：

非递归实现：

```
def binary_search(alist, item):
      first = 0
      last = len(alist)-1
      while first<=last:
          midpoint = (first + last)/2
          if alist[midpoint] == item:
              return True
          elif item < alist[midpoint]:
              last = midpoint-1
          else:
              first = midpoint+1
    return False
testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binary_search(testlist, 3))
print(binary_search(testlist, 13))
```

递归实现：

```
   def binary_search(alist, item):
       if len(alist) == 0:
           return False
       else:
           midpoint = len(alist)//2
           if alist[midpoint]==item:
             return True
           else:
             if item<alist[midpoint]:
               return binary_search(alist[:midpoint],item)
             else:
```

```
            return binary_search(alist[midpoint+1:],item)

testlist = [0, 1, 2, 8, 13, 17, 19, 32, 42,]
print(binary_search(testlist, 3))
print(binary_search(testlist, 13))
```

堆

在小顶堆中，父节点的值小于或等于其子节点的值。

代码实现：

```
import heapq


heap = []  # 创建一个空堆

list1 = list(map(int,input().split()))

heapq.heapify(list1)#将列表转化成堆


heapq.heappush(heap, 5)  # 插入元素 5

heapq.heappush(heap, 2)  # 插入元素 2

heapq.heappush(heap, 8)  # 插入元素 8

print(heap)  # 输出: [2, 5, 8]
```

```
min_value = heapq.heappop(heap)  # 删除并返回最小值

print(min_value)  # 输出: 2

print(heap)  # 输出: [5, 8]
```

保留小数

%：四舍五入，自动补零

```
a = float(input())

print('%.3f'%a)

print('%.6f'%a)

#输入：1.4345

#输出：1.435

       1.434500
```

Format(a -> float , ‘.4f’): 四舍五入，自动补零

```
a = float(input())

print(format(a, '.3f'))

print(format(a, '.6f'))

#输入：1.4345

#输出：1.435

       1.434500
```

round()：四舍五入，不会补零

```
a=float(input())

print(round(a,3))
print(round(a,6))

#输入：1.4345

#输出：1.435

       1.4345
```

enumerate 函数用法：其作用于列表给出 index,item

```
for index,item in enumerate(list1):
```

字典：

对字典里面的 key 取最小值只需要 min(dict)即可

对字典里面的 value 取最值要用 min(dict.values())

输入多组数据，并且买有终止条件时使用：

```
While True:

   try:

   except EOFError:

      break
```

pow(a，b)函数：a 代表要处理的数，b 代表 a 的倍数。

math.log2(num)可以直接开根