SHL.

Question

The current selected programming language is **Python3.7**. We emphasize the submission of a fully working code over partially correct but efficient code. Once **submitted**, you cannot review this problem again. You can use *print* to debug your code. The *print* may not work in case of syntax/runtime error. The version of **Python3.7** being used is **3.7.3**.

An employee in an organization has begun working on N projects (numbered 0 to N-1). Each week he/she can work on a single module of one of the projects. The modules that are chosen on any two successive weeks should come from different projects. A project i can have at most Ci modules. The modules of the projects are such that a module is completed in a week.

Write an algorithm to determine the number of weeks the employee can work on projects following the above-mentioned rules.

Input
The first line of the input consists of an integer - *num*, representing the number of projects (N).
The next line consists of N space-separated integers - *projC0, projC1,... projCN-1*, representing the number of modules of the projects.

Output
Print an integer representing the maximum number of weeks the employee can work on the projects.

Constraints

$1 \leq num \leq 5*10^4$

$1 \leq projCi \leq 10^7$

$\Sigma\ projCi \leq 10^5$

$0 \leq i < num$

Example
Input:
3
7 2 3

Output:
11

Explanation:
The first, second and third projects have 7, 2 and 3 modules respectively.
The modules of different projects are selected on successive weeks in a sequence: first, second, first, third, first, second, first, third, first, third, first.
So, the maximum number of weeks an employee can work on these projects is 11.

```python
"""

"""
def workingWeeks(projC):
    # Write your code here

    return

def main():
    #input for projC
    projC = []
    projC_size  = int(input())
    projC = list(map(int,input().split()))


    result = workingWeeks(projC)
    print(result)

if __name__ == "__main__":
    main()
```

The current selected programming language is **Python3.7**. We emphasize the submission of a fully working code over partially correct but efficient code. Once **submitted**, you cannot review this problem again. You can use *print* to debug your code. The *print* may not work in case of syntax/runtime error. The version of **Python3.7** being used is **3.7.3**.

A mouse is placed in a maze. There is a huge chunk of cheese somewhere in the maze. The maze is represented as an N x M grid of integers where 0 represents a wall, 1 represents the path where the mouse can move and 9 represents the chunk of cheese. The mouse starts at the top left corner at (0,0).
Write an algorithm to output 1 *if* the mouse can reach the chunk of cheese, else output 0.

Input
The first line of the input consists of two space-separated integers - *maze_row* and *maze_col* representing the number of rows (N) and the number of columns (M) in the maze, respectively.
The next N lines consist of M space-separated integers representing the maze.

Output
Print 1 *if* there is a path from the initial position of the mouse to the cheese, else print 0.

Note
The mouse is not allowed to leave the maze or climb the walls.

Example
Input:
8 8
1 0 1 1 1 0 0 1
1 0 0 0 1 1 1 1
1 0 0 0 0 0 0 0
1 0 1 0 9 0 1 1
1 1 1 0 1 0 0 1
1 0 1 0 1 1 0 1
1 0 0 0 0 1 0 1
1 1 1 1 1 1 1 1

Output:
1

Explanation:
The mouse can reach the chunk of cheese.
So, the output 1.