一、编程题

ACM： 最多购买宝石数目

题目描述：

橱窗里有一排宝石，不同的宝石对应不同的价格，宝石的价格标记为gems[i],0<=i<n, n = gems.length
宝石可同时出售0个或多个，如果同时出售多个，则要求出售的宝石编号连续；例如客户最大购买宝石个数为m，购买的宝石编号必须为gems[i],gems[i+1]...gems[i+m-1](0<=i<n,m<=n)
假设你当前拥有总面值为value的钱，请问最多能购买到多少个宝石,如无法购买宝石，则返回0.

输入描述：第一行输入n，参数类型为int，取值范围：[0,10^6]，表示橱窗中宝石的总数量。
之后n行分别表示从第0个到第n-1个宝石的价格，即gems[0]到gems[n-1]的价格，类型为int，取值范围：(0,1000]。
之后一行输入v，类型为int，取值范围：[0,10^9]表示你拥有的钱。

输出描述：输出int类型的返回值，表示最大可购买的宝石数量。

补充说明：

示例1

输入：7
8
4
6
3
1
6
7
10

输出：3

说明：

```
gems = [8,4,6,3,1,6,7], value = 10
```

最多购买的宝石为gems[2]至gems[4]或者gems[3]至gems[5]

示例2

输入：0
1

输出：0

说明：

```
gems = [], value = 1
```

因为没有宝石，所以返回0

示例3
输入：9
6
1
3
1
8
9
3
2
4
15
输出：4
说明：

gems = [6, 1, 3, 1, 8, 9, 3, 2, 4], value = 15

最多购买的宝石为gems[0]至gems[3]

示例4
输入：9
1
1
1
1
1
1
1
1
1
10
输出：9
说明：

gems = [1, 1, 1, 1, 1, 1, 1, 1, 1], value = 10

最多购买的宝石为gems[0]至gems[8]，即全部购买

代码：

```
#include <cstddef>
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, a, sum;
    cin >> n;
    vector<int> vec;
    for (int i = 0 ; i<n ;i++)
    {
```

```
        cin >> a;
        vec.push_back(a);
    }
    cin >> sum;

    auto ca = vector<vector<int>>(vec.size(), vector<int>()); //  保存以最后购买 i 下标宝石时价格总数
    vector<int> ca2;
    for (auto i = 0; i< vec.size(); i++)
    {
        if (vec[i] > sum)
            ca[i].push_back(0);
        else
            ca[i].push_back(vec[i]);
    }
    int maxc = 0;
    for (int i = 1; i < vec.size(); i++)
    {
        if (ca[i-1][0] == 0)
            continue;

        for (int j = 0; j < ca[i-1].size(); j++)
        {
            if (ca[i-1][j] + vec[i] <= sum)
            {
                ca[i].push_back(ca[i-1][j] + vec[i]);
            }
            else
            {
                break;
            }
        }

        // vector<int> ca3;
        // if (!ca2.empty())
        // {
        //      swap(ca2, ca3);
        // }

        // if (vec[i] > sum)
        //      continue;
        // else
        //      ca2.push_back(vec[i]);
```

```
            // for (int j = 0; j < ca3.size(); j++)
            // {
            //         if (ca3[j] + vec[i] <= sum)
            //         {
            //                 ca2.push_back(ca3[j] + vec[i]);
            //         }
            //         else
            //         {
            //                 break;
            //         }
            // }

            // int s = ca2.size();
            // maxc = max(maxc, s);
        }


        for (auto it : ca)
        {
            if (it[0] != 0)
            {
                int s = it.size();
                maxc = max(maxc, s);
            }
        }
        cout << maxc;
        return 0;
}
```