题目描述

给一个空格分开的字符串，表示人线上每个位置 $n(1<n<=2^31-1)$ 的值，例如：1 2 5 7 9 1 2 2，n=8，表示n2+n1+n0 的可能取值，1<=k<=1000，求小于等于k的最大，最接近k的数字 $n0+n1+n2$。

如：现有四个数字 0，1，2，3，当存在 0，1，2 时，他能组成的最小的数字 0，1，2；当存在 1，2，3 时，3-2=1，2，3，1 也能组成。要求人线位置 n0<n1<n2，此时每个单位要长：

现在 0-0，到二维平面中 0 坐标处结束，得点 (0，0)，最终每个行新一个长度 12。

输入描述

输入第一行包括两行以空格隔开的字符串，如"5 6 7 8"，为此时数字有几个线，输入的正整数个数为 n，1<=n<=1000。

输出描述

最终每个数字之间的空格，只要最终所印数字即可，如"5 6 7 8"，所对数字应为这些，5 1 2 3

用例

输入	3 10 20 30 40 1
输出	1 1 1 0
说明	3+10+20+30+40，输入 10,40，-3，10，20，30，40 分别应 10，3 应 10%，40%的 30+40的比例为 1 0.1%。

输入	6 7 8 12 9
输出	9 10 8 7 6
说明	无

输入	4 2 5 7 9 1 2 2
输出	4 1 3 9 4 1
说明	无

题目解析

我的解题思路如下：

我们可以先求出最大数值n9的所对应的数值k0，比如遍历一个序列

arr	1	2	5	7	9	1	2	2
dp	1	3	8	15	24	25	27	29

被下来我们第一个数时 1，从第 1 行开始计算

索引	0	1	2	3	4	5	6	7
arr	1	2	5	7	9	1	2	2
dp	1	3	8	15	24	25	27	29

此时我们需要关注arr位和dp位关系：

- 如果 arr位 i == dp位-1，那么说明第一个要求范围的arr元素之和等于arr位，这此可以组成一个所有元素之和等于k,arr位的，那dp位 arr位i-1 范围所有元素都能重复k，那此时dp的第一个范围所有元素都能重复k，那此时dp的k-1 所有元素都能重复k。
- 如果arr位 i < dp位-1，那说明那第arr位索引第一个 i 的元素范围重复k一遍，那这样arr位k，此时不适合并。
- 如果arr位 i > dp位-1，

1. 如果我们可以找到一个k索引位置，记为k(== dp位-1)，arr位k，那此k+1 一直在范围的arr元素之和等于arr位，此时可以k+1-1 范围arr元素之和于arr位，那arr位 i == k(* 2，那此arr的k+1-1范围所有元素都能重复k，那此时dp的k+1-1 所有元素都能重复k。
2. 如果找不到这样的k索引位置，此时无法合并。

最后一个例的，按照上面逻辑，那此过程如下：

索引	0	1	2	3	4	5	6	7
arr	1	2	5	7	9	1	2	2
dp	1	3	8	15	24	25	27	29

arr位 i > dp位-1，的此无法合并，那直接入下一步

索引	0	1	2	3	4	5	6	7
arr	1	2	5	7	9	1	2	2
dp	1	3	8	15	24	25	27	29

同上

索引	0	1	2	3	4	5	6	7
arr	1	2	5	7	9	1	2	2
dp	1	3	8	15	24	25	27	29

arr位 < dp位-1
说明找到了k = 8满足
dp位 i == dp位-1) - arr位k,
的此可以找
k+1-i-1范围内arr元素开入到arr分中

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	2	2
dp	1	1	1	15	24	25	27	29

k+1-i-1范围内arr元素被开入arr分红，则：
k+1-i-1范围的arr元素被重置为0.
dp元素也要重置为dp[k]

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	2	2
dp	1	1	1	15	24	25	27	29

arr位 < dp位-1
但此时k已经到一个k = 5，满足
dp[k] == dp[i-1] - arr[k],
的此无法合并，直接跳过

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	2	2
dp	1	1	1	15	24	25	27	29

同上

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	2	2
dp	1	1	1	15	24	25	27	29

同上

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	2	2
dp	1	1	1	15	24	25	27	29

同上

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	2	2
dp	1	1	1	15	24	25	27	29

arr位 < dp位-1
但此时k已经到一个k = 5，以
dp[k] == dp[i-1] - arr[k]
的的，但此时k特别
k+1-i-1范围内arr元素开入arr分

索引	0	1	2	3	4	5	6	7
arr	1	0	0	14	9	1	0	0
dp	1	1	1	15	24	25	25	27

k+1-i-1范围内arr元素被开入arr分红，则：
k+1-i-1范围的arr元素被重置为0.
dp元素也要重置为dp[k]

最后arr数组中去除掉 0 值得元素，得到 [1, 14, 9, 1, 4] ，刚好了三组为最解。

Java算法源码

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(" ").map(Integer::parseInt).toArray(Integer[]::new);

    System.out.print(getResult(arr));
  }

  public static String getResult(Integer[] arr) {
    int n = arr.length;

    int[] dp = new int[n];
    dp[0] = arr[0];
    for (int i = 1; i < n; i++) {
      dp[i] = dp[i - 1] + arr[i];
    }

    for (int i = 1; i < n; i++) {
      if (dp[i - 1] == arr[i]) {
        arr[i] *= 2;
        Arrays.fill(arr, 0, i, 0);
        Arrays.fill(dp, 0, i, 0);
        continue;
      }

      if (dp[i - 1] > arr[i]) {
        boolean[] dp2 = new boolean[n];
        for (int k = 0; k < i; k++) {
          if (dp[k] == prevSum) {
            arr[i] *= 2;
            Arrays.fill(arr, j + 1, i, 0);
            Arrays.fill(dp, j + 1, i, dp[j]);
            break;
          }

          if (dp[k] > prevSum) {
            break;
          }
          dp2[k] = true;
        }
      }
    }

    StringJoiner sj = new StringJoiner(" ");
    for (int i = n - 1; i >= 0; i--) {
      if (arr[i] != 0) {
        sj.add(arr[i] + "");
      }
    }
    return sj.toString();
  }
}
```

JS算法源码

```
const fs = require('fs').readFileSync(0, 'utf-8').trim();

const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const arr = line.split(" ").map(Number);

  console.log(getResult(arr));
});

function getResult(arr) {
  const n = arr.length;

  const dp = new Array(n).fill(0);
  dp[0] = arr[0];
  for (let i = 1; i < n; i++) dp[i] = dp[i - 1] + arr[i];

  for (let i = 1; i < n; i++) {
    if (dp[i - 1] == arr[i]) {
      arr[i] *= 2;
      arr.fill(0, 0, i);
      dp.fill(0, 0, i);
      continue;
    }

    if (dp[i - 1] > arr[i]) {
      const prevSum = dp[i - 1] - arr[i];
      for (let j = 0; j < i - 1; j++) {
        if (dp[j] == prevSum) {
          arr[i] *= 2;
          arr.fill(0, j + 1, i);
          dp.fill(dp[j], j + 1, i);
          break;
        }

        if (dp[j] > prevSum) {
          break;
        }
      }
    }
  }

  return arr
    .filter(x => x != 0)
    .reverse()
    .join(" ");
}
```

Python算法源码

```
# 输入处理
arr = list(map(int, input().split()))

# 核心逻辑
def getResult():
    n = len(arr)

    dp = [0] * n
    dp[0] = arr[0]
    for i in range(1, n):
        dp[i] = dp[i - 1] + arr[i]

    for i in range(1, n):
        if dp[i - 1] == arr[i]:
            arr[i] *= 2
            for j in range(0, i):
                arr[j] = 0
                dp[j] = 0
            continue

        if dp[i - 1] > arr[i]:
            prevSum = dp[i - 1] - arr[i]
            for j in range(i - 1):
                if dp[j] == prevSum:
                    arr[i] *= 2
                    for k in range(j + 1, i):
                        arr[k] = 0
                        dp[k] = dp[j]
                    break

                if dp[j] > prevSum:
                    break

    ans = list(filter(lambda x: x != 0, arr))
    ans.reverse()
    return " ".join(map(str, ans))

# 输出结果
print(getResult())
```