36、信道分配，考点 or 实现——逻辑分析

题目描述

算法工程师小明面对着这样一个问题，需要将通信用的信道分配给尽量多的用户:

信道的条件及分配规则如下:

1. 所有信道都有属性:"阶"。阶为 r的信道的容量为 2^r比特;
2. 所有用户需要传输的数据量都一样:D比特;
3. 一个用户可以分配多个信道，但每个信道只能分配给一个用户;
4. 只有当分配给一个用户的所有信道的容量和>=D，用户才能传输数据;

给出一组信道资源，最多可以为多少用户传输数据?

输入描述

第一行，一个数字 R。R为最大阶数。

0<=R<20

第二行，R+1个数字，用空格隔开。代表每种信道的数量 Ni。按照阶的值从小到大排列。

0<=i<=R,0<=Ni<1000.

第三行，一个数字 D。D为单个用户需要传输的数据量。

0<D<1000000

输出描述

一个数字,代表最多可以供多少用户传输数据。

用例

输入	5 10 5 0 1 3 2 30
输出	4
说明	无

题目解析

这题是真的难，可能是我没想到点子上吧，解法想了一晚上，第二天带着两个黑眼圈，灵光一闪，有了下面的解法。

首先，题目的意思，我一开始就没看懂，后面琢磨来琢磨去，分析题目的意思应该是:

第一行输入的r表示第二行的最大阶数，第二行又是r+1个数，也就是说:

比如第一行输入的r=5，则第二行会输入r+1=6个数，如下

- 第一个数10，的阶是0，即容量是2^0的信道有10个
- 第二个数 5，的阶是1，即容量是2^1的信道有5个
- 第三个数 0，的阶是2，即容量是2^2的信道有0个
- 第四个数 1，的阶是3，即容量是2^3的信道有1个
- 第五个数 3，的阶是4，即容量是2^4的信道有3个
- 第六个数 2，的阶是5，即容量是2^5的信道有2个

题目要求从上面给定的多种信道中，任意挑选几个（未使用的）进行组合，让组合之和大于等于第三行输入的D值，问这种组合最多能有多少个?

这题，我一开始想要dfs求得所有无重复使用信道的组合，但是发现只能求出一类，而无法求出多类。大家可以试试，看看dfs行不行。

之后我又想，想要最多的组合情况，那么信道就要省着用，比如每个用户需要至少D容量的信道组合，那么我们就尽可能地构造出容量准确为D的信道组合，

比如 16 + 8 + 4 + 2 = 30，因此我们可以选择：

> 一个阶4的信道，一个阶3的信道，一个阶2的信道，一个阶1的信道。

但是由于没有阶2的信道，因此我们可以将对于阶2的需求降级，变为两个阶1的信道，也就是说最终选择是：

> 一个阶4的信道，一个阶3的信道，三个阶1的信道。

即 16 + 8 + 2 * 3 = 30

另外，如果单个信道容量就能满足D，比如阶5的单个信道容量是32，虽然此时浪费了一些，但是一个信道只能给一个用户使用，因此为了避免更大的浪费，32的信道就可以单独组合，而不需要组合其他信道。

那么如何能准确的构造出容量为30的信道组合呢?

题目中信道容量是2^n，因此我有了如下思路：

首先，我们将D值转为二进制，然后转为字符数组，再反序，让D和N的阶数方向保持一致，

比如D=30，可以转为[0, 1, 1, 1, 1] 的反序二进制值的个数数组，该数组的含义是

- 2^0有0个
- 2^1有1个
- 2^2有1个
- 2^3有1个
- 2^4有1个

```
> let D = 30
< undefined
> Number(D).toString(2).split('').reverse()
< ▶ (5) ['0', '1', '1', '1', '1']
>
```


而输入的第二行，也就是N也可以看成反序二进制数的个数数组： [10, 5, 0, 1, 3, 2]

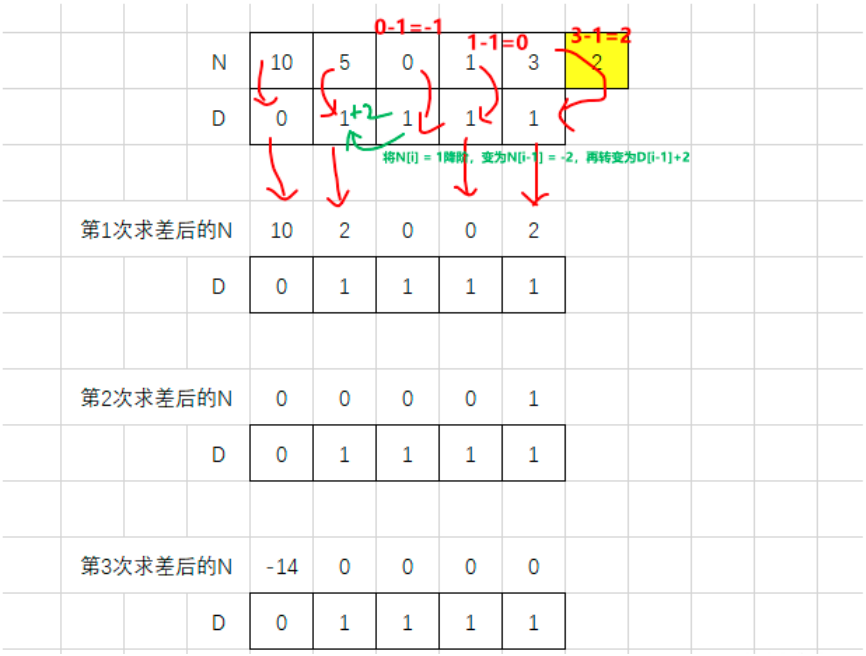
该数组的含义是：

- 2^0有10个
- 2^1有5个
- 2^2有0个
- 2^3有1个
- 2^4有3个
- 2^5有2个

如果想从N中选几个信道组成和为D的组合，那么不就是N和D的二进制值个数求差吗？

N	10	5	0	1	3	2
D	0	1	1	1	1	

我们只需要比较D.length范围的内，而超出D.length范围的N[i]其实就是单个信道就足以满足一个用户通信的。



如上图所示，如果每次求差后，N[0] >= 0，则表示可以构造出一个和为30的信道组合。

但是N[0] < 0 只能表示无法构造出一个和准确为30的的信道组合，但是却还是有可能构造出一个和大于30的信道组合

程序输入

```
5
10 5 0 1 3 2
47
```

对应的N和D如下

N	10	5	0	1	3	2
D	1	1	1	1	0	1
第1次求差后的N	9	2	0	0	3	1
D	1	1	1	1	0	1
第2次求差后的N	-2	0	0	0	3	0
D	1	1	1	1	0	1

此时N[0] < 0，但是还是有可能构造出一个和大于30的组合的

我们可以将N[0]的负值不断升阶，以求得高阶N[i]来抵消掉负值，需要注意的是升阶过程中，N[i]的值最少为-1

N	-2 -2 >> 1 = -1	0	0	0	3	0
N	0	-1 -1+0 = -1 -1 >> 1 = -1	0	0	3	0
N	0	0	-1 -1 + 0 = -1 -1 >> 1 = -1	0	3	0
N	0	0	0	-1 -1+0=-1 -1>>1=-1	3	0
N	0	0	0	0	2 -1+3=2	0

直到N[i]完全抵消了负值结束，然后count++。也可能N[i]到最后也没有抵消完负值，此时说明无法构造出一个和大于30的信道组合，整个程序结束。

以下代码比较难以理解，建议大家debug模式帮助理解，可以监听几个关键值

- N：输入的第二行转化来信道个数数组
- D2：输入的第三行D转化来的构造出准确D容量的信道个数数组
- i：循环变量
- minus：N[i]和D2[i]的差值
- count：满足要求的信道组合个数

Python算法源码

```python
# 输入获取
r = int(input())
N = list(map(int, input().split()))
d = int(input())


# 算法入口
def getResult():
    D = list(map(int, str(bin(d))[2:]))
    D.reverse()

    count = 0

    if len(N) > len(D):
        for i in range(len(D), len(N)):
            count += N[i]

    while True:
        D2 = D[:]
        for i in range(len(D)-1, 0, -1):
            if N[i] > 0:
                diff = N[i] - D2[i]
                if diff >= 0:
                    N[i] = diff
                    D2[i] = 0
                else:
                    D2[i] = 0
```

```
                    D2[i-1] += abs(diff) * 2
                    N[i] = 0
            else:
                D2[i-1] += D2[i] * 2
                D2[i] = 0

        flag = False

        if N[0] >= D2[0]:
            N[0] -= D2[0]
            count += 1
        else:
            N[0] -= D2[0]
            D2[0] = 0
            for i in range(len(D)):
                if N[i] < 0:
                    if i != len(D) - 1:
                        N[i+1] += N[i] >> 1
                        N[i] = 0
                    else:
                        flag = True
                else:
                    count += 1
                    break

        if flag:
            break

    return count


# 算法调用
print(getResult())
```

Java算法源码

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

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

    int r = sc.nextInt();

    int[] N = new int[r + 1];
    for (int i = 0; i <= r; i++) {
      N[i] = sc.nextInt();
    }

    int d = sc.nextInt();

    System.out.println(getResult(r, N, d));
  }

  public static int getResult(int r, int[] N, int d) {
    Integer[] D =
        Arrays.stream(new StringBuilder(Integer.toString(d, 2)).reverse().toString().split(""))
            .map(Integer::parseInt)
            .toArray(Integer[]::new);

    int count = 0;

```

```
    int Nlen = N.length;
    int Dlen = D.length;

    if (Nlen > Dlen) {
      for (int i = Dlen; i < Nlen; i++) {
        count += N[i];
      }
    }

    while (true) {
      int[] D2 = new int[Dlen];
      for (int i = 0; i < Dlen; i++) D2[i] = D[i];

      for (int i = Dlen - 1; i >= 1; i--) {
        if (N[i] > 0) {
          int diff = N[i] - D2[i];
          if (diff >= 0) {
            N[i] = diff;
            D2[i] = 0;
          } else {
            D2[i] = 0;
            D2[i - 1] += Math.abs(diff) * 2;
            N[i] = 0;
          }
        } else {
          D2[i - 1] += D2[i] * 2;
          D2[i] = 0;
        }
      }

      boolean flag = false;
      if (N[0] >= D2[0]) {
        N[0] -= D2[0];
        count++;
      } else {
        N[0] -= D2[0];
        D2[0] = 0;
        for (int i = 0; i < Dlen; i++) {
          if (N[i] < 0) {
            if (i != Dlen - 1) {
              N[i + 1] += N[i] >> 1;
              N[i] = 0;
            } else {
              flag = true;
            }
```

```
            } else {
              count++;
              break;
            }
          }
        }

        if (flag) break;
      }

      return count;
    }
}
```

JavaScript算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

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

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 3) {
    let R = parseInt(lines[0]);
    let N = lines[1]
      .split(" ")
      .slice(0, R + 1)
      .map(Number);
    let D = parseInt(lines[2]);

    console.log(getMaxCount(N, D));
    lines.length = 0;
  }
});

function getMaxCount(N, d) {
  const D = Number(d).toString(2).split("").map(Number).reverse();

```

```
  let count = 0;

  let Nlen = N.length;
  let Dlen = D.length;

  if (Nlen > Dlen) {
    for (let i = Dlen; i < Nlen; i++) {
      count += N[i];
    }
  }

  while (true) {
    const D2 = D.slice();
    for (let i = Dlen - 1; i >= 1; i--) {
      if (N[i]) {
        let minus = N[i] - D2[i];
        if (minus >= 0) {
          N[i] = minus;
          D2[i] = 0;
        } else {
          D2[i] = 0;
          D2[i - 1] += Math.abs(minus) * 2;
          N[i] = 0;
        }
      } else {
        D2[i - 1] += D2[i] * 2;
        D2[i] = 0;
      }
    }

    let flag = false;
    if (N[0] >= D2[0]) {
      N[0] -= D2[0];
      count++;
    } else {
      N[0] -= D2[0];
      D2[0] = 0;
      for (let i = 0; i < Dlen; i++) {
        if (N[i] < 0) {
          if (i !== D.length - 1) {
            N[i + 1] += N[i] >> 1;
            N[i] = 0;
```

```
71            } else {
72              flag = true;
73            }
74          } else {
75            count++;
76            break;
77          }
78        }
79      }
80
81      if (flag) {
82        break;
83      }
84    }
85
86    return count;
87  }
```