E3-F题题解

> 作者：余绍函

题目描述

「魈」也想长高高，请你帮帮他！

「魈」从起点出发，想要到达终点，沿途有若干的 `buff` 或是 `深坑`。用 a_i表示第 i 个遇到特殊事件为 `buff` 或 `深坑`。

若 $a_i \geq 0$，表示第 i 个遇到的为 `buff`，可以帮助「魈」的身高提高 a_i。若 $a_i < 0$，则表示第 i 个遇到的是深度为 $-a_i$ 的 `深坑`，若此时「魈」的身高 h 大于 $-a_i$（即大于坑的深度），则可直接越过这个坑，若 h 小于等于 $-a_i$，则「魈」会因为掉入坑中导致身高减半（即 $h' = \lfloor h/2 \rfloor$）。但「魈」有 k 次使用 「风轮两立」的机会，可以不用考虑身高直接跳过一个 `深坑`。

测试数据保证每组数据中 $a_i < 0$ 的个数不超过 2×10^3。

当「魈」的身高变为 0 时，将直接失败，无法到达终点。现给出「魈」的初始身高 h，和沿途会遇到的 n 个 `buff` 或 `深坑`，请你求出「魈」在到达终点时的最高身高，若无论如何都无法到达终点，则输出 -1。

输入

第一行为数据组数 $T(1 \leq T \leq 10)$。

接下来 T 组数据：

第一行三个整数 $n, k, h(1 \leq n \leq 10^6, 0 \leq k \leq 2 \times 10^3, 1 \leq h \leq 10^3)$，分别表示沿途会遇到的 n 个特殊事件，最多可使用「风轮两立」的次数和「魈」的初始身高。

第二行 n 个整数 $a_i(-10^3 \leq ai \leq 10^3)$，表示沿途遇到的 `buff` 或 `深坑`。

输出

对于每组数据，输出一行一个正整数，表示到达终点时能达到的最高身高，若无法到达输出 -1。

输入样例

```
3
4 1 10
-10 -5 1 2
3 0 1
-1 1 2
5 2 1
-1 2 -3 5 -10
```

输出样例

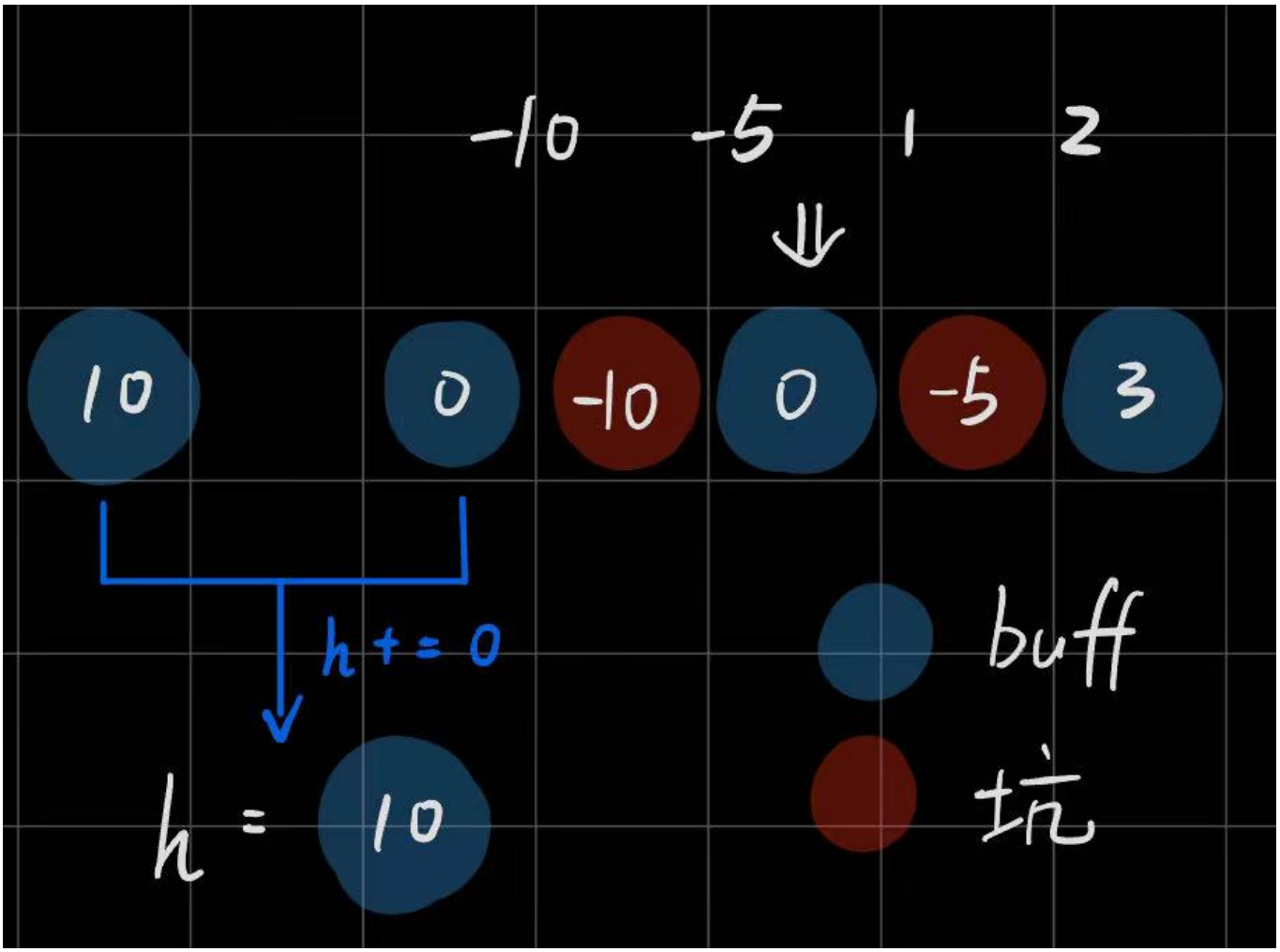
```
13
-1
6
```

解题思路

优化数据结构

理解题意后，不难发现当「魈」向终点前进的过程中，导致最终身高差异的仅仅与他在**遇到坑的时候跳或不跳有关**，故经过坑后的身高和经过坑前的身高和跳/不跳有关。因此，我们只需要保证每次过坑前的身高是最优的，那么算出来经过坑后的身高肯定也是最优的，这就符合**动态规划**的思路了。

但是在真正做题开始前，我们可以对题目进行一些优化：1. 连续的多个 `buff` 可以被叠加到一起，不用分开计算 2. 没遇到坑前的 `buff` 可以直接和初始身高加到一起，以简便计算 3. 如果两个坑之间没有 `buff` ，我们可以视为其中存在一个增益为 0 的 `buff` 。由于题目规定了，遇到的坑最多有 2×10^3 个，我们这样就可以更简单的处理数据。以第一个样例为例：



通过这样的优化，我们可以把问题转变为：*「魈」每经过一个坑就一定可以得到一个buff*。

设计状态转移方程

上文中已经描述了设计状态转移方程的基本思路，根据「魈」经过坑的身高变化规律，我们不难提出状态转移方程:

$$dp[i][j] = \begin{cases} dp[i][j-1] + buff[i] & \text{dp[i][j-1]+hole[i]} \geq 0 \\ max(dp[i-1][j-1] + buff[i], dp[i][j-1]/2 + buff[i]) & \text{dp[i][j-1]+hole[i]} < 0 \end{cases}$$

其中，i 代表「魈」已经使用技能跳过 i 个 `hole`，j 代表「魈」已经走过 j 个 `buff` 和 `hole`。

代码实现

```
#include<bits/stdc++.h>
using namespace std;
int t, n, k, dp[2003][2003], hole[2003], buff[2003], route[1000003], rp, hbp,
```

```
maxHeight;
int main() {
    scanf("%d", &t);
    while(t--) {
        scanf("%d%d%d", &n, &k, &(dp[0][0]));
        for(int i = 0; i < n; i++)
            scanf("%d", &route[i]);
        rp = hbp = 0;
        while(route[rp] >= 0 && rp < n)
            dp[0][0] += route[rp++];
        while(rp < n) {
            hole[hbp] = route[rp++];
            buff[hbp] = 0;
            while(rp < n && route[rp] >= 0)
                buff[hbp] += route[rp++];
            hbp++;
        }
        for(int i = 1; i <= hbp; i++) {
            dp[0][i] = (dp[0][i - 1] + hole[i - 1] > 0) ? (dp[0][i - 1] +
buff[i - 1]) : (dp[0][i - 1] > 1 ? (dp[0][i - 1] / 2 + buff[i - 1]) : 0);
            dp[i][i] = dp[i - 1][i - 1] + buff[i - 1];
        } for(int i = 1; i <= hbp; i++) {
            for(int j = i + 1; j <= hbp; j++) {
                if(dp[i - 1][j - 1] > 0)
                    dp[i][j] = max(dp[i - 1][j - 1] + buff[j - 1], (dp[i][j -
1] + hole[j - 1] > 0) ? (dp[i][j - 1] + buff[j - 1]) : (dp[i][j - 1] > 1 ?
(dp[i][j - 1] / 2 + buff[j - 1]) : 0));
                else
                    dp[i][j] = (dp[i][j - 1] + hole[j - 1] > 0) ? (dp[i][j -
1] + buff[j - 1]) : (dp[i][j - 1] > 1 ? (dp[i][j - 1] / 2 + buff[j - 1]) : 0);
            }
        } maxHeight = dp[0][hbp];
        for(int i = 1; i <= k; i++)
            maxHeight = max(maxHeight, dp[i][hbp]);
        printf("%d\n", maxHeight > 0 ? maxHeight : -1);
    } return 0;
}
```

再优化思路

本题是可以使用 滚动数组 对空间复杂度进一步优化的，但由于课程中没涉及、这道题没有卡空间，故不作太多介绍。欢迎感兴趣的同学自行研究~