1、小多想在美化一下自己的庄园。他的庄园毗邻一条小河，他希望在河边种一排树，共 M 棵。小多采购了 N 个品种的树，每个品种的数量是 Ai (树的总数量恰好为 M)。但是他希望任意两棵相邻的树不是同一品种的。小多请你帮忙设计一种满足要求的种树方案。

解析：使用搜索来做，但纯粹使用搜索的话通过率为 90%，有一个点会超时，所以需要剪枝，一个简单的剪枝思路是比较当前未种的树和坑的大小关系！

具体的剪枝思路是每次搜索之前判断当前剩余的坑位 left 和任意品种的树之间的关系:

1) 如果 left 为偶数，那么只要 tree[i] > left / 2，就表示肯定种不了

2) 如果 left 为奇数，那么只要 tree[i] > (left + 1) / 2，就表示肯定种不了

这里有一个小技巧：left 为偶数时，left/2 和(left + 1)/2 的值是相等的，所以可以统一使用 tree[i] > (left+1)/2 的关系来做剪枝优化！

```
import java.util.*;

public class Main {

    static int n, m;
    static int[] tree;
    static List<String> ans;

    static boolean check(int left) {
        for (int i = 1; i <= n; i++) {
            if (tree[i] > (left + 1) / 2) return false;
        }
        return true;
    }

    static boolean dfs(int idx) {
        if (!check(m - idx)) return false;
        if (idx == m) {
            return true;
        } else {
            for (int i = 1; i <= n; i++) {
                if (idx == 0 || (tree[i] != 0 && i != Integer.valueOf(ans.get(idx
- 1)))) {
                    tree[i]--;
                    ans.add(i + "");
                    if (dfs(idx + 1)) return true;
                    ans.remove(ans.size() - 1);
                    tree[i]++;
                }
            }
        }
        return false;
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            n = sc.nextInt();
            tree = new int[n + 1];
            for (int i = 1; i <= n; i++) {
                tree[i] = sc.nextInt();
                m += tree[i];
            }
            ans = new ArrayList<>();
            if (dfs(0)) {
                System.out.println(String.join(" ", ans));
            } else {
                System.out.println("-");
            }
        }
    }
}
```

2 给定一个长度为偶数的数组 arr，将该数组中的数字两两配对并求和，在这些和中选出最大和最小值，请问该如何两两配对，才能让最大值和最小值的差值最小？

解析：

```
import java.util.*;
public class Main{
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int[] a=new int[n];
        for(int i=0;i<n;i++){
            a[i]=sc.nextInt();
        }
        Arrays.sort(a);

        int max=0,min=Integer.MAX_VALUE;
        for(int i=0,j=n-1;i<n/2;i++,j--){
            max=Math.max(max,a[i]+a[j]);
            min=Math.min(min,a[i]+a[j]);
        }
        System.out.println(max-min);
    }
}
```

2、你在玩一个回合制角色扮演的游戏。现在你在准备一个策略，以便在最短的回合内击败敌方角色。在战斗开始时，敌人拥有 HP 格血量。当血量小于等于 0 时，敌人死去。一个缺乏经验的玩家可能简单地尝试每个回合都攻击。但是你知道辅助技能的重要性。

在你的每个回合开始时你可以选择以下两个动作之一：聚力或者攻击。

聚力会提高你下个回合攻击的伤害。

攻击会对敌人造成一定量的伤害。如果你上个回合使用了聚力，那这次攻击会对敌人造成 buffedAttack 点伤害。否则，会造成 normalAttack 点伤害。
给出血量 HP 和不同攻击的伤害，buffedAttack 和 normalAttack，返回你能杀死敌人的最小回合数。
解析:用一个 int 变量 buffed,当 buffed=1 表示 buffedAttack,buffed=0 表示 normalAttack，然后每次进入递归方法都有两种决策方式。具体 Code 如下所示：

```
import java.util.HashMap;
import java.util.Scanner;

public class Main {

    public static HashMap<String, Integer> map = new HashMap<>(); //用于记忆化搜索

    /**
     * @param hp 当前 hp
     * @param normalAttack 普通攻击
     * @param buffedAttack 辅助攻击
     * @param isBuffed 本回合是否使用辅助攻击
     * @param cur 当前回合数
     * <a href="/profile/547241" data-card-uid="547241" class="js-nc-card" target="_blank">@return
     */
    public static int process(int hp, int normalAttack, int buffedAttack, int isBuffed, int cur) {
        String str = isBuffed + "_" + hp; //用 isBuffed 和 hp 作为记忆化搜索的 key

        //每次进入递归之前先搜索是否有缓存结果
        if (map.containsKey(str)) {
            return map.get(str);
        }
        //如果 hp < 1,返回需要的回合
        if (hp < 1) {
            map.put(str, cur); //返回结果前加缓存
            return cur;
        }

        int res = 0;
        if (isBuffed == 1) {
            //如果上回合用 buffedAttack，则这回合直接减去敌人生命值
            res = process(hp - buffedAttack, normalAttack, buffedAttack, 0, cur + 1);
        } else {
            //如果上回合使用 normalAttack，则这回合有两种决策方式
```

```
            res = Math.min(process(hp - normalAttack, normalAttack, buffedAttack,
0, cur + 1),
                    process(hp, normalAttack, buffedAttack, 1, cur + 1));
        }
        map.put(str, res); //返回结果前加缓存
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int hp = sc.nextInt();
        int normal = sc.nextInt();
        int buffed = sc.nextInt();
        //一开始有两种选择
        int a = process(hp, normal, buffed, 1, 1); //蓄力
        int b = process(hp - normal, normal, buffed, 0, 1); //普通攻击
        System.out.println(Math.min(a, b));
    }
}
</a>
```