**Solution ASDE Algorithm Test**

**Algorithm canAbhimanyuCross**

Input: initialPower, size = 11, currentPower, enemy[], noOfSkips, noOfRecharges, currentCircle

Output: boolean value indicating if Abhimanyu can cross

Used dynamic programming recursively

1. If currentCircle >= n then

Return true

2. If currentPower < 0 then

Return false

3. Create a key for map using currentPower, noOfSkips, noOfRecharges, and currentCircle

4. If key exists in the map then

Return the value associated with the key

5. Initialize option1, option2, and option3 as false

6. If currentPower >= enemy[i] then

Set option1 to canAbhimanyuCross(initialPower, size, currentPower - enemy[i], enemy, noOfSkips, noOfRecharges, currentCircle + 1)

7. If currentPower < enemy[i] and initialPower >= enemy[i] and noOfRecharges > 0 then

Set option2 to canAbhimanyuCross(initialPower, size, initialPower - enemy[i], enemy, noOfSkips, noOfRecharges - 1, currentCircle + 1)

8. If currentCircle is 2 or 6 and noOfSkips >= 2 then

Set option3 to canAbhimanyuCross(initialPower, size, currentPower, enemy, noOfSkips - 2, noOfRecharges, currentCircle + 2)

9. Else if currentCircle is not 2 or 6 and noOfSkips > 0 then

Set option3 to canAbhimanyuCross(initialPower, size, currentPower, enemy, noOfSkips - 1, noOfRecharges, currentCircle + 1)

10. Set ans to option1 OR option2 OR option3

11. Store ans in the map with the key

12. Return ans

Main Algorithm:

1. Create an instance of Chakraviyoh

2. Initialize map as a new HashMap

3. Test Case 1:

Set power1 = 10

Set enemy1 = [2, 3, 4, 5, 5, 4, 3, 7, 10, 11, 1]

Set noOfSkips1 = 2

Set noOfRecharges1 = 3

enemy1[2] = enemy[2]\*1.5

enemy1[6] = enemy[6]\*1.5

Print result of canAbhimanyuCross(power1, enemy1.length, power1, enemy1, noOfSkips1, noOfRecharges1, 0)

4. Test Case 2:

Set power2 = 10

Set enemy2 = [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]

Set noOfSkips2 = 2

Set noOfRecharges2 = 3

Print result of canAbhimanyuCross(power2, enemy2.length , power2, enemy2, noOfSkips2, noOfRecharges2, 0)

**Java Code**

import java.util.HashMap;

class Chakraviyoh {

HashMap<String, Boolean> map;

public boolean canAbhimanyuCross(double initialPower, int n, double p, double[] k, int a, int b, int i) {

if(i>=n) return true;

if(p<0) return false;

String key = p+" "+a+" "+b+" "+i;

if(map.containsKey(key)) return map.get(key);

boolean option1 = false, option2 = false, option3 = false;

if(p>=k[i]) {

option1 = canAbhimanyuCross(initialPower, n, p-k[i], k, a, b, i+1);

}

if(p<k[i] && initialPower >= k[i] && b>0){

option2 = canAbhimanyuCross(initialPower, n, initialPower-k[i], k, a, b-1, i+1);

}

if((i== 2 || i == 6) && a>=2){

option3 = canAbhimanyuCross(initialPower, n, p, k, a-2, b, i+2);

}

else if (a>0){

option3 = canAbhimanyuCross(initialPower, n, p, k, a-1, b, i+1);

}

boolean ans = option1 || option2 || option3;

map.put(key, ans);

return ans;

}

public static void main(String[] args) {

Chakraviyoh c1 = new Chakraviyoh();

c1.map = new HashMap<>();

int p1 = 10;

double[] k1 = {2, 3, 2, 5, 5, 4, 3, 7, 10, 11, 1};

int a1 = 2;

int b1 = 3;

k1[2] = k1[2]\*1.5;

k1[6] = k1[6]\*1.5;

System.out.println(c1.canAbhimanyuCross(p1, k1.length, p1, k1, a1, b1, 0)); // Expected Output: true

Chakraviyoh c2 = new Chakraviyoh();

c2.map = new HashMap<>();

// Test Case 2

int p2 = 10;

double[] k2 = {5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5};

int a2 = 2;

int b2 = 3;

k2[2] = k2[2]\*1.5;

k2[6] = k2[6]\*1.5;

System.out.println(c2.canAbhimanyuCross(p2, k2.length, p2, k2, a2, b2, 0)); // Expected Output: false

}

}