问题描述

小R正在计划一次从地点A到地点B的徒步旅行，总路程需要 N 天。为了在旅途中保持充足的能量，小R每天必须消耗1份食物。幸运的是，小R在路途中每天都会经过一个补给站，可以先购买完食物后再消耗今天的1份食物。然而，每个补给站的食物每份的价格可能不同，并且小R在购买完食物后最多只能同时携带 K 份食物。

现在，小R希望在保证每天食物消耗的前提下，以最小的花费完成这次徒步旅行。你能帮助小R计算出最低的花费是多少吗？

\*\*输入 \*\*

n 总路程需要的天数

k 小R最多能同时携带食物的份数

data[i] 第i天补给站每份食物的价格

\*\*输出 \*\*

返回完成这次徒步旅行的最小花费

\*\*约束条件 \*\*

1 < n,k < 1000

1 < data[i] < 10000

测试样例

样例1：

输入：n = 5 ,k = 2 ,data = [1, 2, 3, 3, 2]

输出：9

样例2：

输入：n = 6 ,k = 3 ,data = [4, 1, 5, 2, 1, 3]

输出：9

样例3：

输入：n = 4 ,k = 1 ,data = [3, 2, 4, 1]

输出：10

Issue in the Original Code

State Transition Issue: The logic inside the second loop is not correctly calculating the minimum cost. Instead of considering dp[i-1], you need to track costs when purchasing from previous stations within the k day window.

Dynamic Programming Table: dp[i] represents the minimum cost to reach the ith day, but you need to ensure it incorporates previous days' costs and handle the purchase logic properly.

DP:

public class Main {

public static int solution(int n, int k, int[] data) {

// Length of the array data

int length = data.length;

// Create dp array to store the minimum cost at each day

int[] dp = new int[length];

// Base case: On day 0, you must buy food for that day

dp[0] = data[0];

// Iterate over the days

for (int i = 1; i < length; i++) {

// Start with buying food from the current station on the i-th day

dp[i] = dp[i - 1] + data[i];

// Try purchasing food from previous days, considering carrying food from the previous days

for (int j = i - 1; j >= i - k && j >= 0; j--) {

dp[i] = Math.min(dp[i], dp[j] + data[i]);

}

}

// Return the minimum cost to finish the trip on the last day

return dp[length - 1];

}

public static void main(String[] args) {

// Test cases to check the solution

System.out.println(solution(5, 2, new int[]{1, 2, 3, 3, 2}) == 9);

System.out.println(solution(6, 3, new int[]{4, 1, 5, 2, 1, 3}) == 9);

System.out.println(solution(4, 1, new int[]{3, 2, 4, 1}) == 10);

}

}

Greedy:

import java.util.ArrayList;

import java.util.List;

public class Main {

public static int solution(int n, int k, int[] data) {

// Initialize the length of the data array and the result variable

int length = data.length;

int res = 0;

// This is my backpack (a list of food items)

List<Food> bag = new ArrayList<>();

// Start buying food

for (int i = 0; i < length; i++) {

// If the backpack is empty, buy food until it's full

if (bag.isEmpty()) {

while (bag.size() != k) {

Food food = new Food();

food.index = i;

res += data[i];

bag.add(food);

}

} else {

// Temporary backpack to organize the food

List<Food> temp = new ArrayList<>();

for (int j = 0; j < bag.size(); j++) {

if (data[bag.get(j).index] > data[i]) {

// Refund the food that is more expensive than the current day's food

res -= data[bag.get(j).index];

} else {

// Keep the cheaper food

temp.add(bag.get(j));

}

}

// Fill the backpack until it's full

while (temp.size() != k) {

Food food = new Food();

food.index = i;

res += data[i];

temp.add(food);

}

bag = temp;

}

// At the end of the day, eat the cheapest food; expensive food can wait for a refund

Food min\_val\_food = bag.get(0);

for (int j = 1; j < bag.size(); j++) {

if (data[min\_val\_food.index] > data[bag.get(j).index]) {

min\_val\_food = bag.get(j);

}

}

bag.remove(min\_val\_food);

}

// At the end of the journey, sell all excess food

while (!bag.isEmpty()) {

Food remove = bag.remove(0);

res -= data[remove.index];

}

return res;

}

public static void main(String[] args) {

// Test case

System.out.println(solution(5, 2, new int[]{1, 2, 3, 3, 2}) == 9);

}

}

// Helper class to represent food items

class Food {

int index;

}

Attetntion:食物是可以退的，然后先吃便宜的，贵的最后吃或退款