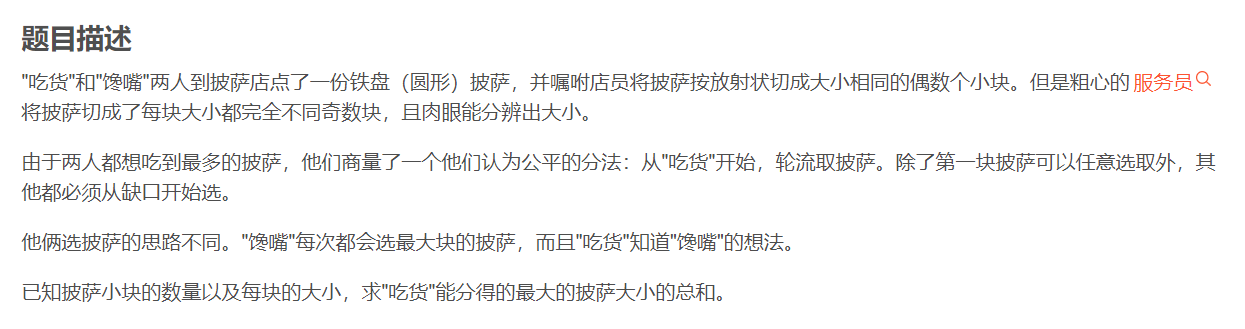
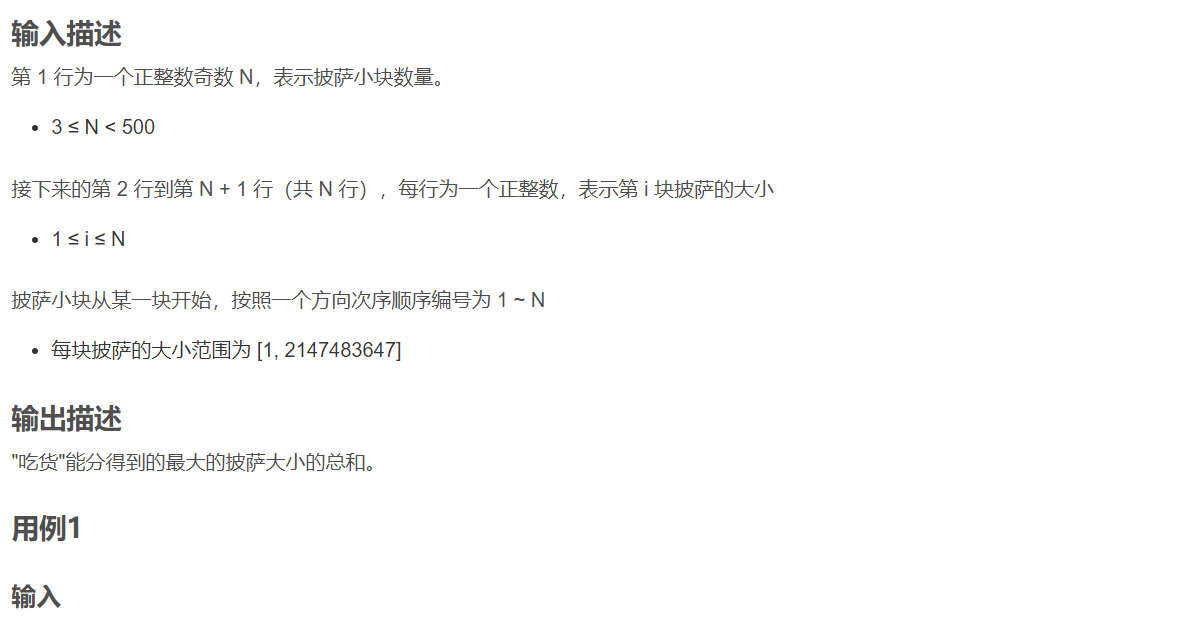
# **E卷-分批萨[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**





5

8

2

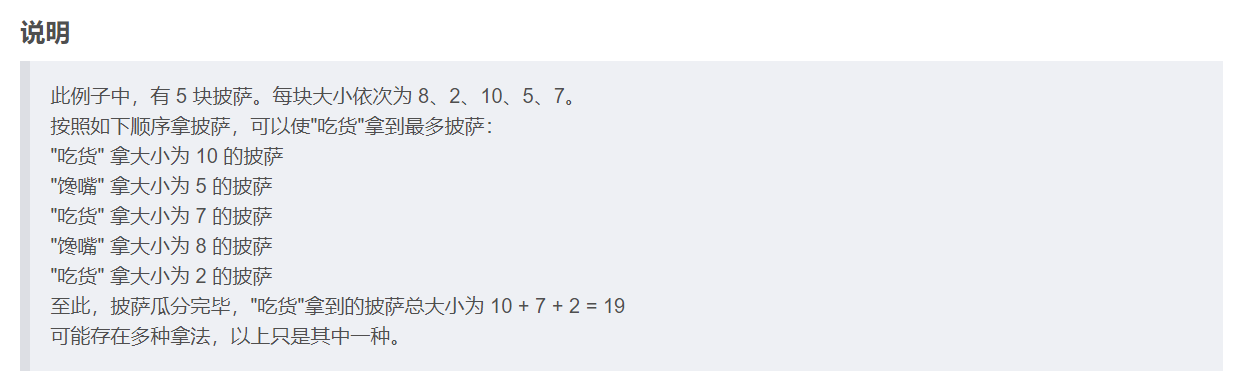
10

5

7



19





import java.util.Scanner;

public class Main {

// 定义一个函数来计算吃货能获得的最大披萨大小总和

public static int maxPizzaSum(int[] pizzaSizes) {

int n = pizzaSizes.length; // 获取披萨块的数量

int[][] dp = new int[n][n]; // 创建一个二维数组用于动态规划

// 初始化 dp 数组，每块披萨单独考虑时的值

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

dp[i][i] = pizzaSizes[i];

}

// 计算 dp 数组

for (int length = 2; length <= n; length++) { // 从长度 2 开始计算

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

int j = (i + length - 1) % n; // 计算环形数组中的结束位置

// 选择开始或结束位置，计算最大值

dp[i][j] = Math.max(

pizzaSizes[i] + Math.min(

dp[(i + 2) % n][j], // 吃货选择 i 后，馋嘴选择 (i+1)

dp[(i + 1) % n][(j - 1 + n) % n] // 吃货选择 i 后，馋嘴选择 j

),

pizzaSizes[j] + Math.min(

dp[i][(j - 2 + n) % n], // 吃货选择 j 后，馋嘴选择 (j-1)

dp[(i + 1) % n][(j - 1 + n) % n] // 吃货选择 j 后，馋嘴选择 i

)

);

}

}

// 找到能得到的最大值

int maxSum = 0;

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

maxSum = Math.max(maxSum, dp[i][(i - 1 + n) % n]);

}

return maxSum; // 返回最大值

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in); // 创建扫描器对象

int n = scanner.nextInt(); // 读取披萨块数量

int[] pizzaSizes = new int[n]; // 创建数组存储披萨块大小

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

pizzaSizes[i] = scanner.nextInt(); // 读取每块披萨的大小

}

System.out.println(maxPizzaSum(pizzaSizes)); // 输出结果

}

}



def max\_pizza\_sum(pizza\_sizes):

n = len(pizza\_sizes)

dp = [[0] \* n for \_ in range(n)]

# 初始化 dp 数组

for i in range(n):

dp[i][i] = pizza\_sizes[i]

# 计算 dp 数组

for length in range(2, n + 1): # 从长度 2 开始

for i in range(n):

j = (i + length - 1) % n

# 吃货可以选择从 i 开始或者从 j 开始

# 馋嘴会总是选择最大块

dp[i][j] = max(

pizza\_sizes[i] + min(

dp[(i + 2) % n][j], # 吃货选择 i 后，馋嘴选择 (i+1)

dp[(i + 1) % n][(j - 1 + n) % n] # 吃货选择 i 后，馋嘴选择 j

),

pizza\_sizes[j] + min(

dp[i][(j - 2 + n) % n], # 吃货选择 j 后，馋嘴选择 (j-1)

dp[(i + 1) % n][(j - 1 + n) % n] # 吃货选择 j 后，馋嘴选择 i

)

)

# 找到能得到的最大值

max\_sum = 0

for i in range(n):

max\_sum = max(max\_sum, dp[i][(i - 1 + n) % n])

return max\_sum

# 输入处理

n = int(input())

pizza\_sizes = [int(input()) for \_ in range(n)]

# 计算结果

result = max\_pizza\_sum(pizza\_sizes)

print(result)



#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// 定义一个函数来计算吃货能获得的最大披萨大小总和

int maxPizzaSum(vector<int>& pizzaSizes) {

int n = pizzaSizes.size(); // 获取披萨块的数量

vector<vector<int>> dp(n, vector<int>(n, 0)); // 创建一个二维数组用于动态规划

// 初始化 dp 数组，每块披萨单独考虑时的值

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

dp[i][i] = pizzaSizes[i];

}

// 计算 dp 数组

for (int length = 2; length <= n; length++) { // 从长度 2 开始计算

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

int j = (i + length - 1) % n; // 计算环形数组中的结束位置

// 选择开始或结束位置，计算最大值

dp[i][j] = max(

pizzaSizes[i] + min(

dp[(i + 2) % n][j], // 吃货选择 i 后，馋嘴选择 (i+1)

dp[(i + 1) % n][(j - 1 + n) % n] // 吃货选择 i 后，馋嘴选择 j

),

pizzaSizes[j] + min(

dp[i][(j - 2 + n) % n], // 吃货选择 j 后，馋嘴选择 (j-1)

dp[(i + 1) % n][(j - 1 + n) % n] // 吃货选择 j 后，馋嘴选择 i

)

);

}

}

// 找到能得到的最大值

int maxSum = 0;

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

maxSum = max(maxSum, dp[i][(i - 1 + n) % n]);

}

return maxSum; // 返回最大值

}

int main() {

int n;

cin >> n; // 读取披萨块数量

vector<int> pizzaSizes(n); // 创建数组存储披萨块大小

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

cin >> pizzaSizes[i]; // 读取每块披萨的大小

}

cout << maxPizzaSum(pizzaSizes) << endl; // 输出结果

return 0;

}



#include <stdio.h>

#include <stdlib.h>

// 定义一个函数来计算吃货能获得的最大披萨大小总和

int maxPizzaSum(int\* pizzaSizes, int n) {

int\*\* dp = (int\*\*)malloc(n \* sizeof(

int\*)); // 创建二维数组用于动态规划

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

dp[i] = (int\*)malloc(n \* sizeof(int));

dp[i][i] = pizzaSizes[i]; // 初始化 dp 数组

}

// 计算 dp 数组

for (int length = 2; length <= n; length++) { // 从长度 2 开始计算

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

int j = (i + length - 1) % n; // 计算环形数组中的结束位置

// 选择开始或结束位置，计算最大值

dp[i][j] = (pizzaSizes[i] + (dp[(i + 2) % n][j] < dp[(i + 1) % n][(j - 1 + n) %

n] ? dp[(i + 2) % n][j] : dp[(i + 1) % n][(j - 1 + n) % n])) >

(pizzaSizes[j] + (dp[i][(j - 2 + n) % n] < dp[(i + 1) % n][(j - 1 + n) % n] ?

dp[i][(j - 2 + n) % n] : dp[(i + 1) % n][(j - 1 + n) % n])) ?

(pizzaSizes[i] + (dp[(i + 2) % n][j] < dp[(i + 1) % n][(j - 1 + n) % n] ? dp[(i

+ 2) % n][j] : dp[(i + 1) % n][(j - 1 + n) % n])) :

(pizzaSizes[j] + (dp[i][(j - 2 + n) % n] < dp[(i + 1) % n][(j - 1 + n) % n] ?

dp[i][(j - 2 + n) % n] : dp[(i + 1) % n][(j - 1 + n) % n]));

}

}

// 找到能得到的最大值

int maxSum = 0;

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

if (dp[i][(i - 1 + n) % n] > maxSum) {

maxSum = dp[i][(i - 1 + n) % n];

}

}

// 释放内存

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

free(dp[i]);

}

free(dp);

return maxSum; // 返回最大值

}

int main() {

int n;

scanf("%d", &n); // 读取披萨块数量

int\* pizzaSizes = (int\*)malloc(n \* sizeof(

int)); // 创建数组存储披萨块大小

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

scanf("%d", &pizzaSizes[i]); // 读取每块披萨的大小

}

printf("%d\n", maxPizzaSum(pizzaSizes, n)); // 输出结果

free(pizzaSizes); // 释放内存

return 0;

}



function maxPizzaSum(pizzaSizes) {

const n = pizzaSizes.length; // 获取披萨块的数量

const dp = Array.from({ length: n }, () => Array(n).fill(0)); // 创建二维数组用于动态规划

// 初始化 dp 数组，每块披萨单独考虑时的值

for (let i = 0; i < n; i++) {

dp[i][i] = pizzaSizes[i];

}

// 计算 dp 数组

for (let length = 2; length <= n; length++) { // 从长度 2 开始计算

for (let i = 0; i < n; i++) {

const j = (i + length - 1) % n; // 计算环形数组中的结束位置

// 选择开始或结束位置，计算最大值

dp[i][j] = Math.max(

pizzaSizes[i] + Math.min(

dp[(i + 2) % n][j], // 吃货选择 i 后，馋嘴选择 (i+1)

dp[(i + 1) % n][(j - 1 + n) % n] // 吃货选择 i 后，馋嘴选择 j

),

pizzaSizes[j] + Math.min(

dp[i][(j - 2 + n) % n], // 吃货选择 j 后，馋嘴选择 (j-1)

dp[(i + 1) % n][(j - 1 + n) % n] // 吃货选择 j 后，馋嘴选择 i

)

);

}

}

// 找到能得到的最大值

let maxSum = 0;

for (let i = 0; i < n; i++) {

maxSum = Math.max(maxSum, dp[i][(i - 1 + n) % n]);

}

return maxSum; // 返回最大值

}

// 输入处理

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

let input = [];

rl.on('line', (line) => {

input.push(parseInt(line.trim())); // 读取输入并存入数组

}).on('close', () => {

const n = input[0]; // 披萨块数量

const pizzaSizes = input.slice(1); // 每块披萨的大小

console.log(maxPizzaSum(pizzaSizes)); // 输出结果

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

)

// 定义一个函数来计算吃货能获得的最大披萨大小总和

func maxPizzaSum(pizzaSizes []int) int {

n := len(pizzaSizes) // 获取披萨块的数量

dp := make([][]int, n)

for i := range dp {

dp[i] = make([]int, n)

dp[i][i] = pizzaSizes[i] // 初始化 dp 数组

}

// 计算 dp 数组

for length := 2; length <= n; length++ { // 从长度 2 开始计算

for i := 0; i < n; i++ {

j := (i + length - 1) % n // 计算环形数组中的结束位置

// 选择开始或结束位置，计算最大值

dp[i][j] = max(

pizzaSizes[i]+min(

dp[(i+2)%n][j], // 吃货选择 i 后，馋嘴选择 (i+1)

dp[(i+1)%n][(j-1+n)%n], // 吃货选择 i 后，馋嘴选择 j

),

pizzaSizes[j]+min(

dp[i][(j-2+n)%n], // 吃货选择 j 后，馋嘴选择 (j-1)

dp[(i+1)%n][(j-1+n)%n], // 吃货选择 j 后，馋嘴选择 i

),

)

}

}

// 找到能得到的最大值

maxSum := 0

for i := 0; i < n; i++ {

maxSum = max(maxSum, dp[i][(i-1+n)%n])

}

return maxSum // 返回最大值

}

// 辅助函数：返回两个整数中的最小值

func min(a, b int) int {

if a < b {

return a

}

return b

}

// 辅助函数：返回两个整数中的最大值

func max(a, b int) int {

if a > b {

return a

}

return b

}

func main() {

scanner := bufio.NewScanner(os.Stdin) // 创建扫描器对象

scanner.Scan()

n, \_ := strconv.Atoi(scanner.Text()) // 读取披萨块数量

pizzaSizes := make([]int, n) // 创建数组存储披萨块大小

for i := 0; i < n; i++ {

scanner.Scan()

pizzaSizes[i], \_ = strconv.Atoi(scanner.Text()) // 读取每块披萨的大小

}

fmt.Println(maxPizzaSum(pizzaSizes)) // 输出结果

}