算法设计与分析

班级：计科2206

姓名：韩伯尧

学号：202201200203

**目录**

[第二章 3](#_Toc182999976)

[算法实现题 3-1 3](#_Toc182999977)

[题目 3](#_Toc182999978)

[答案 3](#_Toc182999979)

[算法实践题 3-4 5](#_Toc182999980)

[题目 5](#_Toc182999981)

[答案 5](#_Toc182999982)

[算法实践题3-8 6](#_Toc182999983)

[题目 6](#_Toc182999984)

[答案 7](#_Toc182999985)

[算法实践题3-25 8](#_Toc182999986)

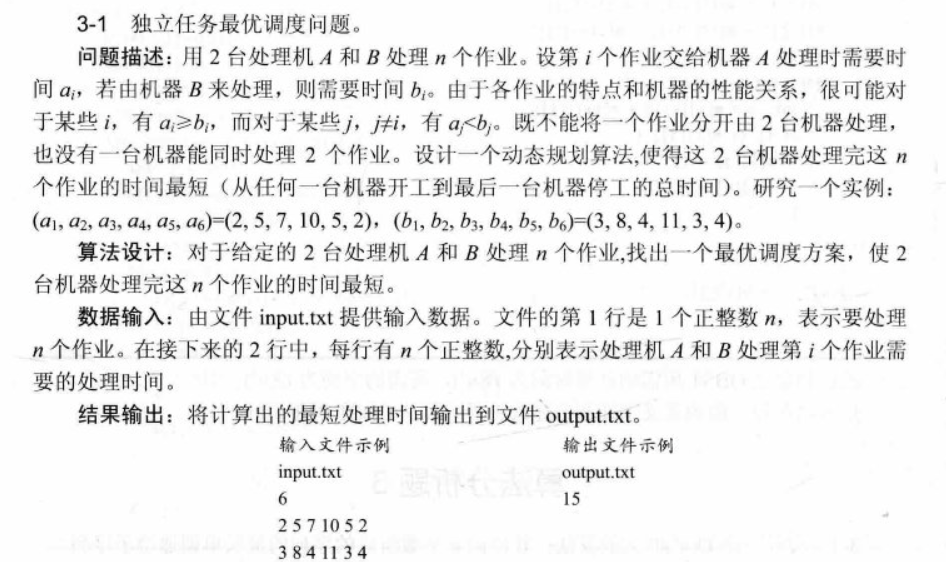
[题目 8](#_Toc182999987)

[答案 9](#_Toc182999988)

第二章

算法实现题 3-1

题目



答案

* 分析

定义状态：设表示前i个作业中，分配给机器A的作业总时间为j时的最小总时间。

状态转移方程：对第i个作业有两种选择：

• 分配给机器A，所需时间为:

• 分配给机器B，所需时间为:

综合这两种选择：

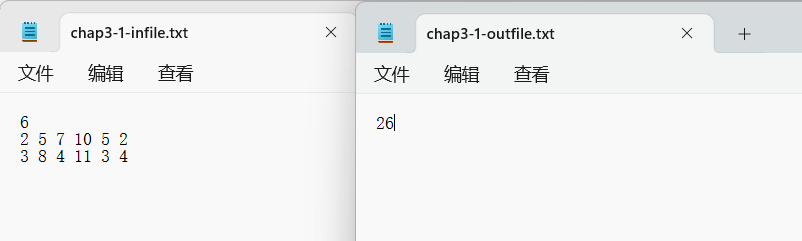
3. 边界条件：, 表示还未分配任何作业时总时间为0。

4. 目标：找出所有中的最小值，其中j是机器A的可能作业总时间。

* 实现

1. **int** main() {
2. freopen("input.txt", "r", stdin);
3. freopen("output.txt", "w", stdout);
4. **int** n;
5. cin >> n;
7. **int** maxn = 0; // 记录 a[i] 的总和
8. **for** (**int** i = 1; i <= n; i++) {
9. cin >> a[i];
10. maxn += a[i];
11. }
12. **for** (**int** i = 1; i <= n; i++) cin >> b[i];
14. // 初始化 dp 数组
15. fill(dp, dp + maxn + 1, INF);
16. dp[0] = 0;
18. // 动态规划求解
19. **for** (**int** i = 1; i <= n; i++) {
20. **for** (**int** j = maxn; j >= 0; j--) {  // 从后往前更新，避免状态覆盖
21. **if** (j >= a[i]) {
22. dp[j] = min(dp[j], dp[j - a[i]] + b[i]);
23. }
24. }
25. }
27. // 找到最优答案
28. **int** ans = INF;
29. **for** (**int** i = 0; i <= maxn; i++) {
30. ans = min(ans, max(i, dp[i]));
31. }
33. cout << ans << endl;
34. **return** 0;
35. }

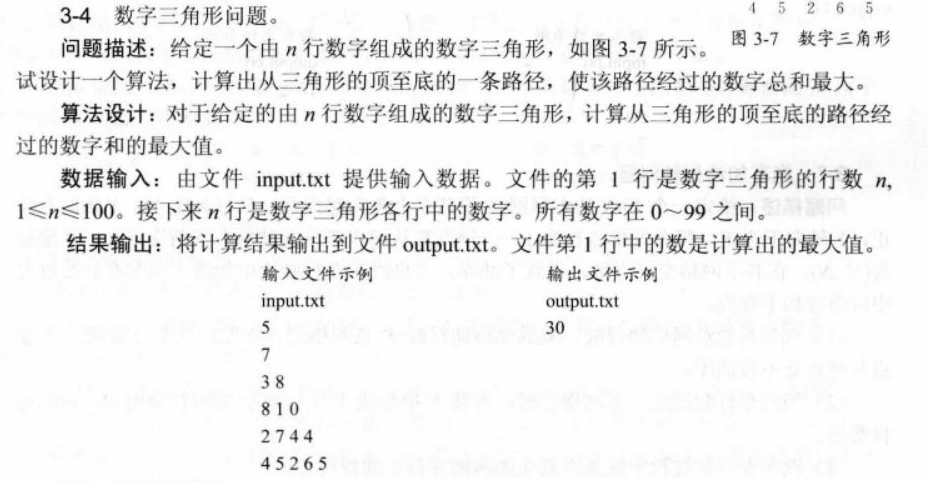
* 结果



时间复杂度的瓶颈是状态总数，优化前后无法降低时间复杂度，因为所有状态必须被遍历。通过滚动数组，空间复杂度从减少到。

算法实践题 3-4

题目



答案

* 分析

递归式定义

对于数字三角形中的任意位置，其最大路径和的定义如下:

如果在最后一行,则：即该位置的数字本身。如果在非最后一行,则：即当前数字加上下一行可能的最大路径和。

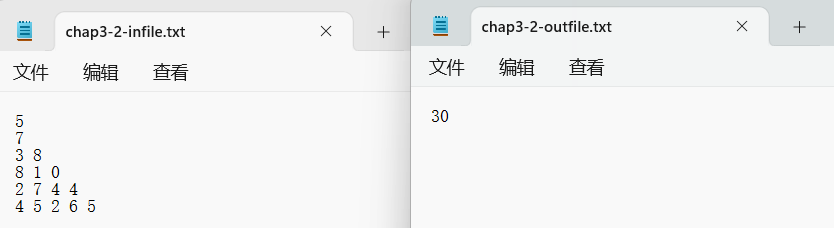
计算方法

从最底层开始计算，将每一行的数字和其下面两格中较大的一个数字相加。最终，顶点(1,1)的值就是从顶到底的最大路径和。

实现

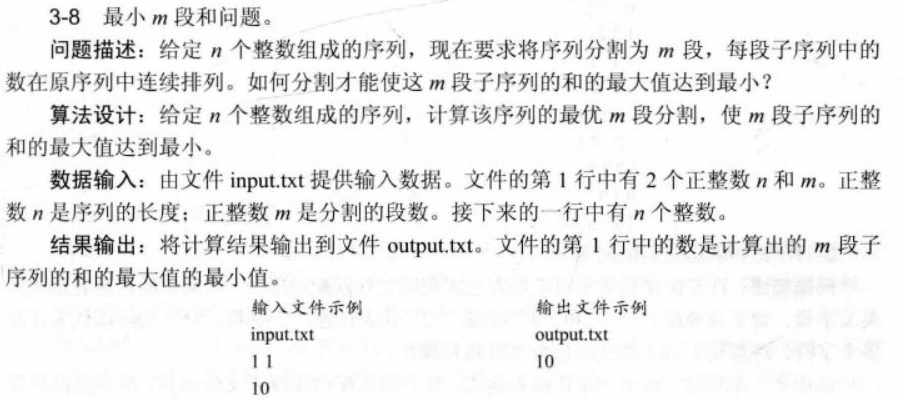
1. **int** Solution() {
2. vector<vector<**int**>> dp = triangle; // 创建副本，保留原始数据
3. **for** (**int** i = n - 1; i >= 1; --i) {
4. **for** (**int** j = 1; j <= i; ++j) {
5. dp[i][j] += max(dp[i + 1][j], dp[i + 1][j + 1]);
6. }
7. }
8. **return** dp[1][1]; // 最大路径和位于顶端
9. }
11. // 回溯求解路径
12. **void** TrackBack() {
13. path.resize(n + 1, 0);
14. **int** i = 1, j = 1; // 从顶端开始
15. **for** (**int** level = 1; level < n; ++level) {
16. **if** (triangle[level + 1][j] > triangle[level + 1][j + 1]) {
17. path[level] = triangle[level][j] - triangle[level + 1][j];
18. ++i;
19. } **else** {
20. path[level] = triangle[level][j] - triangle[level + 1][j + 1];
21. ++i; ++j;
22. }
23. }
24. path[n] = triangle[n][j];
25. }

* 结果



算法实践题3-8

题目



答案

* 分析

定义状态:

表示将长度为i的数组分成j段后,各段子数组和的最大值的最小值

表示数组从第i个元素到第j个元素的子数组和2. 状态转移方程:

表示前k个元素分成j−1段的最优解

:表示从第k+1到第i的子数组和(作为第j段)。在所有可能的划分点k中，取最大值最小的情况。

边界条件:

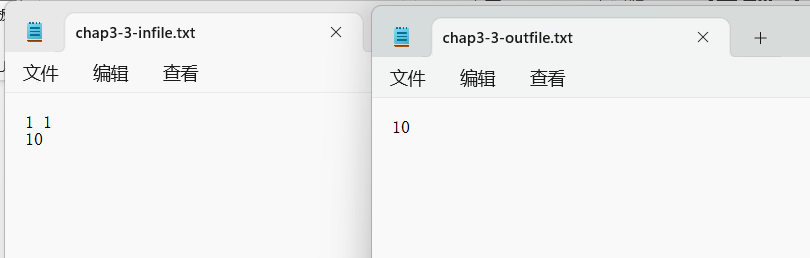
dp[i][1] = sum[1][i]: 分成1段时，结果是整个子数组的和。

目标:求出dp[n][m],即将长度为n的数组分成m段的最优解。

* 实现

1. **int** main() {
2. **int** n, m;
3. cin >> n >> m;
4. vector<**int**> a(n + 1, 0);
5. vector<**int**> prefix\_sum(n + 1, 0);
6. vector<vector<**int**>> dp(2, vector<**int**>(n + 1, INT\_MAX)); // 滚动数组
8. **for** (**int** i = 1; i <= n; ++i) {
9. cin >> a[i];
10. prefix\_sum[i] = prefix\_sum[i - 1] + a[i];
11. }
13. // 初始化 dp[i][1]
14. **for** (**int** i = 1; i <= n; ++i) {
15. dp[1][i] = prefix\_sum[i];
16. }
18. **for** (**int** j = 2; j <= m; ++j) { // 分段数
19. **for** (**int** i = j; i <= n; ++i) { // 长度
20. dp[j % 2][i] = INT\_MAX; // 重置当前状态
21. **for** (**int** k = j - 1; k < i; ++k) {
22. dp[j % 2][i] = min(dp[j % 2][i], max(dp[(j - 1) % 2][k], prefix\_sum[i] - prefix\_sum[k]));
23. }
24. }
25. }
27. cout << dp[m % 2][n] << endl; // 最终答案
28. **return** 0;
29. }

* 结果

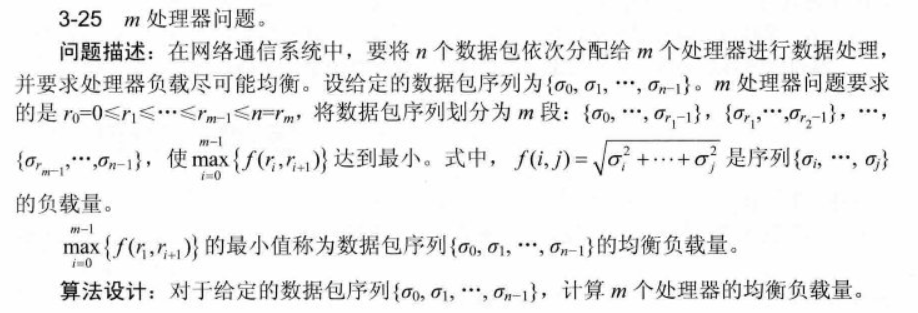


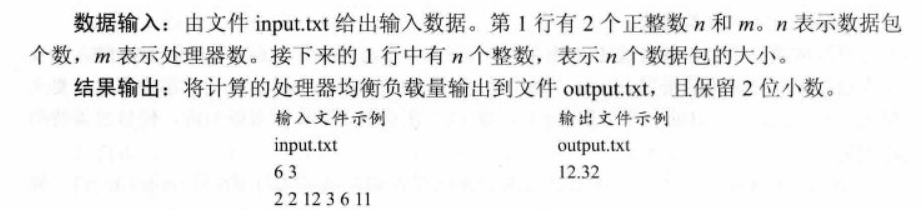
时间复杂度：O(n2 \* m)，其中 n 是数组的长度，m是分成的非空的连续子序列的个数。总状态数为 O(n \* m)，状态转移时间复杂度 O(n)，所以总时间复杂度为 O(n2 \*m)

空间复杂度：O(n\*m)，为动态规划数组的开销。

算法实践题3-25

题目





答案

* 分析

定义状态:

表示将序列从第i个元素开始，分成k段后，各段平方和的平方根的最大值的最小值。

子问题:对于序列从第i到第j个元素,平方和的平方根由函数计算。

如果第i到第j段作为一部分,则剩下的问题转化为从 开始分成段。

状态转移方程:

当前段的平方和平方根。剩下的段的最优解。

4. 边界条件:

分成1段时,结果为从第i到最后一个元素的平方和的平方根。

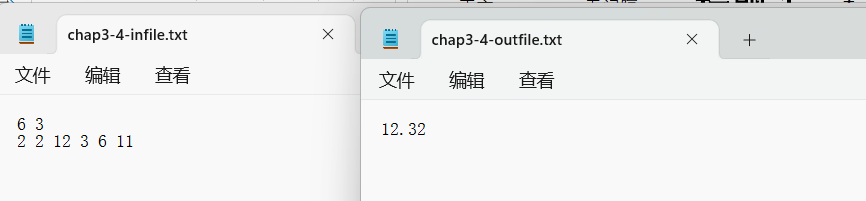
5. 目标:

求将整个序列分成m段的最优解。

* 实现

1. // 预处理平方和
2. **double** f(**int** i, **int** j) {
3. **return** sqrt(prefix\_square[j + 1] - prefix\_square[i]);
4. }
6. **double** solve() {
7. **for** (**int** i = n - 1; i >= 0; i--) {
8. g[i][1] = f(i, n - 1); // 初始化 g[i][1]
9. }
11. **for** (**int** k = 2; k <= m; k++) { // 分成 k 段
12. **for** (**int** i = n - 1; i >= 0; i--) { // 起点 i
13. **double** tmp = INT\_MAX;
14. **for** (**int** j = i; j <= n - k; j++) {
15. **double** maxt = max(f(i, j), g[j + 1][k - 1]);
16. tmp = min(tmp, maxt);
17. }
18. g[i][k] = tmp;
19. }
20. }
22. **return** g[0][m];
23. }

* 结果



时间复杂度：

通过前缀平方和优化f(i,j)，计算f(i,j)的时间降低到O(1)。总时间复杂度降低到。

空间复杂度：为。