作业2

202208010512 计科2205 刘志垚

算法实现题 3-1

3-1　独立任务最优调度问题。

问题描述： 用 2 台处理机 A 和 B 处理 n 个作业。设第 i 个作业交给机器 A 处理时需要时间 a_i，若由机器 B 来处理，则需要时间 b_i。由于各作业的特点和机器的性能关系，很可能对于某些 i，有 $a_i \geqslant b_i$，而对于某些 j，$j \neq i$，有 $a_j < b_j$。既不能将一个作业分开由 2 台机器处理，也没有一台机器能同时处理 2 个作业。设计一个动态规划算法,使得这 2 台机器处理完这 n 个作业的时间最短（从任何一台机器开工到最后一台机器停工的总时间）。研究一个实例：$(a_1, a_2, a_3, a_4, a_5, a_6)=(2, 5, 7, 10, 5, 2)$，$(b_1, b_2, b_3, b_4, b_5, b_6)=(3, 8, 4, 11, 3, 4)$。

算法设计： 对于给定的 2 台处理机 A 和 B 处理 n 个作业,找出一个最优调度方案，使 2 台机器处理完这 n 个作业的时间最短。

数据输入： 由文件 input.txt 提供输入数据。文件的第 1 行是 1 个正整数 n，表示要处理 n 个作业。在接下来的 2 行中，每行有 n 个正整数,分别表示处理机 A 和 B 处理第 i 个作业需要的处理时间。

结果输出： 将计算出的最短处理时间输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
6	15
2 5 7 10 5 2	
3 8 4 11 3 4	

1.思路

1. 定义状态：设 $dp[i][j]$ 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。
2. `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 a_i：

$$dp[i][j] = min(dp[i-1][j-a_i]), j \geq a_i$$

- 分配给机器B，所需时间为 b_i:

$$dp[i][j] = min(dp[i-1][j] + b_i)$$

综合这两种选择：

$$dp[i][j] = min(dp[i-1][j] + b_i, dp[i-1][j-a_i])$$

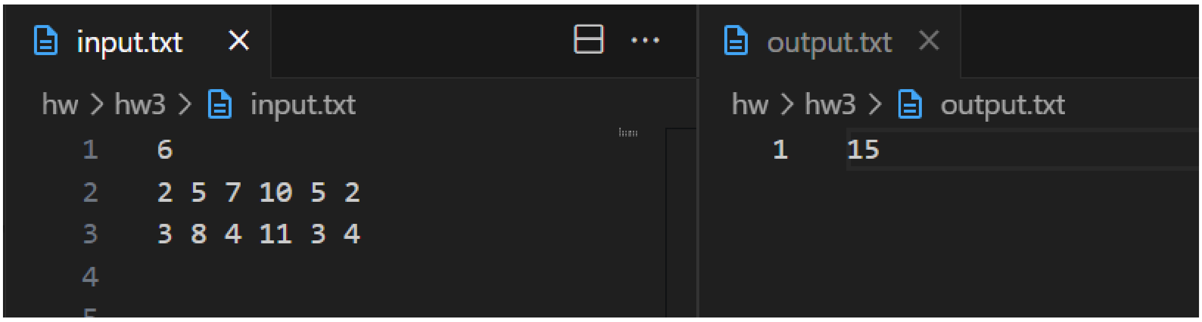
3. 边界条件：$dp[0][0] = 0$，表示还未分配任何作业时总时间为0。
4. 目标：找出所有 $dp[n][j]$ 中的最小值，其中 j 是机器A的可能作业总时间。

2.代码

```
int main() {
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int n;
    cin >> n;

    int m = 0; // 记录 a[i] 的总和
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        m += a[i];
    }
    for (int i = 1; i <= n; i++) cin >> b[i];

    // 初始化 dp 数组
    fill(dp, dp + m + 1, INF);
    dp[0] = 0;

    // 动态规划求解
    for (int i = 1; i <= n; i++) {
        for (int j = m; j >= 0; j--) {  // 从后往前更新，避免状态覆盖
            if (j >= a[i]) {
                dp[j] = min(dp[j], dp[j - a[i]] + b[i]);
            }
        }
    }

    // 找到最优答案
    int ans = INF;
    for (int i = 0; i <= m; i++) {
        ans = min(ans, max(i, dp[i]));
    }

    cout << ans << endl;
    return 0;
```

```
}
```

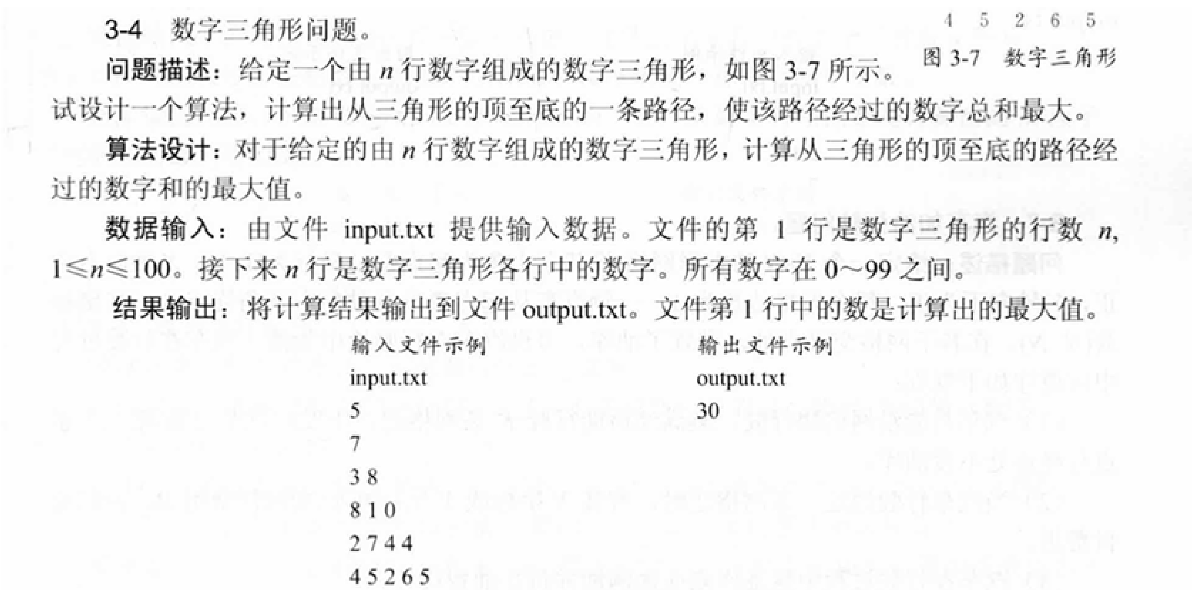
3.结果



4.分析

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

算法实现题 3-4



3-4　数字三角形问题。

4　5　2　6　5

图 3-7　数字三角形

问题描述： 给定一个由 n 行数字组成的数字三角形，如图 3-7 所示。试设计一个算法，计算出从三角形的顶至底的一条路径，使该路径经过的数字总和最大。

算法设计： 对于给定的由 n 行数字组成的数字三角形，计算从三角形的顶至底的路径经过的数字和的最大值。

数据输入： 由文件 input.txt 提供输入数据。文件的第 1 行是数字三角形的行数 n，$1 \leq n \leq 100$。接下来 n 行是数字三角形各行中的数字。所有数字在 0～99 之间。

结果输出： 将计算结果输出到文件 output.txt。文件第 1 行中的数是计算出的最大值。

输入文件示例	输出文件示例
input.txt	output.txt
5	30
7	
3 8	
8 1 0	
2 7 4 4	
4 5 2 6 5	

1.思路

从最后一列开始，逐步往上执行，到达上一层的某个数有两条路径，要么是它下面的数往上走，要么为它下右方的数向左上方走，即 $dp[i][j] = dp[i+1][j] + dp[i][j]$ 或者 $dp[i][j] = dp[i+1][j+1] + dp[i][j]$，取两者最大值更新 $dp[i][j]$ 即可，即：

$$dp[i][j] = max(dp[i+1][j] + dp[i][j], dp[i][j] = dp[i+1][j+1] + dp[i][j])$$

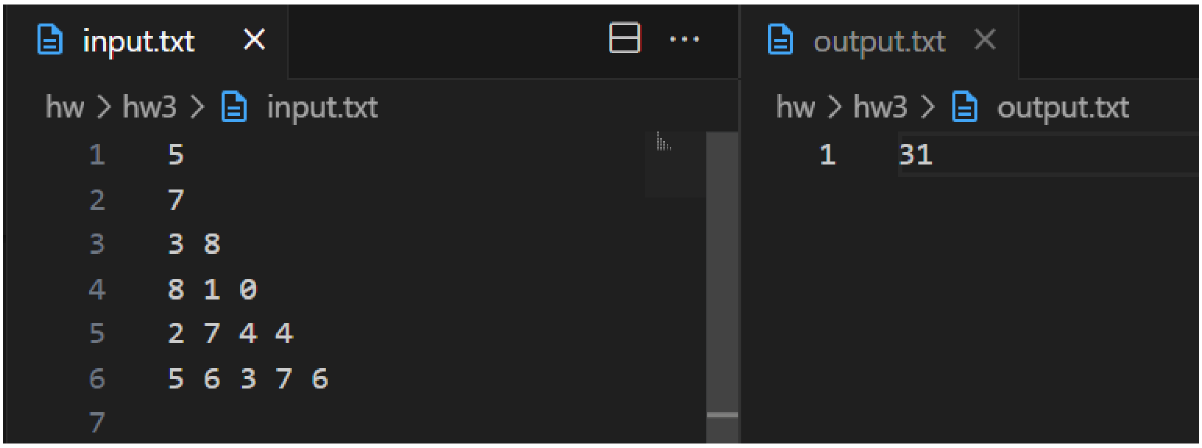
最终，顶点的值就是从顶到底的最大路径和，即 $dp[0][0]$。

2.代码

```
#include <iostream>
using namespace std;

int main() {
```

```
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int n;
    cin >> n;
    int **a =  new int* [n];
    int **dp =  new int* [n];

    for(int i = 0; i < n; ++i) {
        a[i] = new int [i + 1];
        dp[i] = new int [i + 1];
    }

    for(int i = 0 ; i < n; i++) {
        for(int j = 0; j <= i; j++) {
            cin >> a[i][j];
            dp[i][j] = 0;
        }
    }

    for(int i = 0; i < n; i++)
        dp[n - 1][i] = a[n - 1][i];

    for(int  i = n-2; i >= 0; i--) {
        for(int j = 0; j <= i; j++) {
            dp[i][j] = max(dp[i + 1][j + 1], dp[i + 1][j]) + a[i][j];
        }
    }

    cout << dp[0][0];
}
```

3.结果



4.分析

共有 $n \times (n+1)/2$ 个数字，故状态更新需要执行 $n \times (n+1)/2$ 次，故时间复杂度为$O(n^2)$。空间取决于dp数组的大小，为 $O(n^2)$。

算法实现题 3-8

3-8　最小 m 段和问题。

问题描述： 给定 n 个整数组成的序列，现在要求将序列分割为 m 段，每段子序列中的数在原序列中连续排列。如何分割才能使这 m 段子序列的和的最大值达到最小？

算法设计： 给定 n 个整数组成的序列，计算该序列的最优 m 段分割，使 m 段子序列的和的最大值达到最小。

数据输入： 由文件 input.txt 提供输入数据。文件的第 1 行中有 2 个正整数 n 和 m。正整数 n 是序列的长度；正整数 m 是分割的段数。接下来的一行中有 n 个整数。

结果输出： 将计算结果输出到文件 output.txt。文件的第 1 行中的数是计算出的 m 段子序列的和的最大值的最小值。

输入文件示例	输出文件示例
input.txt	output.txt
1 1	10
10	

1.思路

1. 定义状态:

   $dp[i][j]$ 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。

   $sum[i][j]$ 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:`

$$dp[i][j] = min(max(dp[k][j-1], sum[k+1][i])), 1 \leq k < i$$

   $dp[k][j-1]$ 表示前 k 个元素分成 $j-1$ 段的最优解。

   $sum[k+1][i]$ 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。

3. 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。

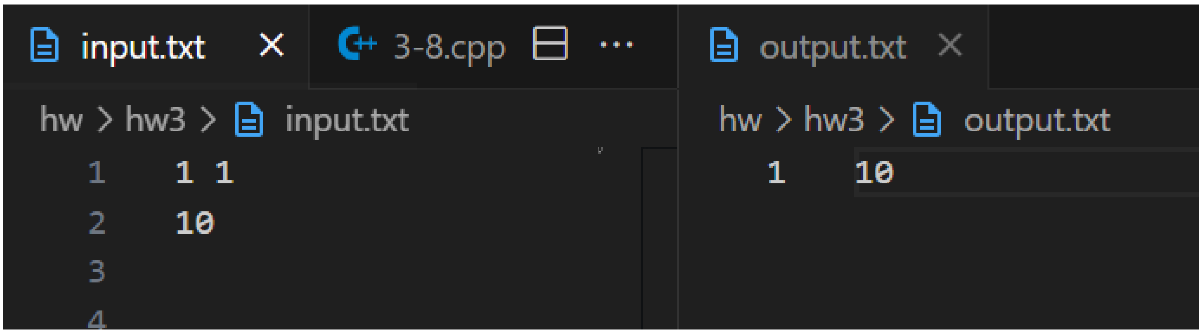
4. 目标:求出 $dp[n][m]$,即将长度为n的数组分成m段的最优解。

2.代码

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    int n, m;
    cin >> n >> m;
    vector<int> a(n + 1, 0);
    vector<int> prefix_sum(n + 1, 0);
    vector<vector<int>> dp(2, vector<int>(n + 1, INT_MAX)); // 滚动数组

    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
        prefix_sum[i] = prefix_sum[i - 1] + a[i];
        }

    // 初始化 dp[i][1]
    for (int i = 1; i <= n; ++i) {
        dp[1][i] = prefix_sum[i];
```

```
    }

    for (int j = 2; j <= m; ++j) { // 分段数
        for (int i = j; i <= n; ++i) { // 长度
            dp[j % 2][i] = INT_MAX; // 重置当前状态
            for (int k = j - 1; k < i; ++k) {
                dp[j % 2][i] = min(dp[j % 2][i], max(dp[(j - 1) % 2][k],
prefix_sum[i] - prefix_sum[k]));
            }
        }
    }

    cout << dp[m % 2][n] << endl; // 最终答案
    return 0;
}
```

3.结果



4.分析

时间复杂度为 $O(n^2 \times m)$，其中 n 是数组的长度，m 是分成的非空连续子序列的个数。总状态数为 $O(n \times m)$，而状态转移的时间复杂度为 $O(n)$，因此总时间复杂度为 $O(n^2 \times m)$。空间复杂度为 $O(n \times m)$，这主要是由于动态规划数组的开销。

算法实现题 3-25

3-25　m 处理器问题。

问题描述：在网络通信系统中，要将 n 个数据包依次分配给 m 个处理器进行数据处理，并要求处理器负载尽可能均衡。设给定的数据包序列为 $\{\sigma_0, \sigma_1, \cdots, \sigma_{n-1}\}$。$m$ 处理器问题要求的是 $r_0=0\leqslant r_1\leqslant\cdots\leqslant r_{m-1}\leqslant n=r_m$，将数据包序列划分为 m 段：$\{\sigma_0, \cdots, \sigma_{r_1-1}\}$，$\{\sigma_{r_1},\cdots,\sigma_{r_2-1}\}$，$\cdots$，$\{\sigma_{r_{m-1}},\cdots,\sigma_{n-1}\}$，使 $\max\limits_{i=0}^{m-1}\{f(r_i,r_{i+1})\}$ 达到最小。式中，$f(i,j)=\sqrt{\sigma_i^2+\cdots+\sigma_j^2}$ 是序列 $\{\sigma_i, \cdots, \sigma_j\}$ 的负载量。

$\max\limits_{i=0}^{m-1}\{f(r_i,r_{i+1})\}$ 的最小值称为数据包序列 $\{\sigma_0, \sigma_1, \cdots, \sigma_{n-1}\}$ 的均衡负载量。

算法设计：对于给定的数据包序列 $\{\sigma_0, \sigma_1, \cdots, \sigma_{n-1}\}$，计算 m 个处理器的均衡负载量。

093

数据输入：由文件 input.txt 给出输入数据。第 1 行有 2 个正整数 n 和 m。n 表示数据包个数，m 表示处理器数。接下来的 1 行中有 n 个整数，表示 n 个数据包的大小。

结果输出：将计算的处理器均衡负载量输出到文件 output.txt，且保留 2 位小数。

输入文件示例	输出文件示例
input.txt	output.txt
6 3	12.32
2 2 12 3 6 11	

1.思路

1. 定义状态:

   $dp[i][k]$ 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。

2. 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 $j+1$ 开始分成 $k-1$ 段。

3. `状态转移方程`:

$$dp[i][k] = min_{j=i}^{n-k} max(f(i,j), dp[j+1][k-1])$$

   当前段的平方和平方根。$dp[j+1][k-1]$ 为剩下的 $k-1$ 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。

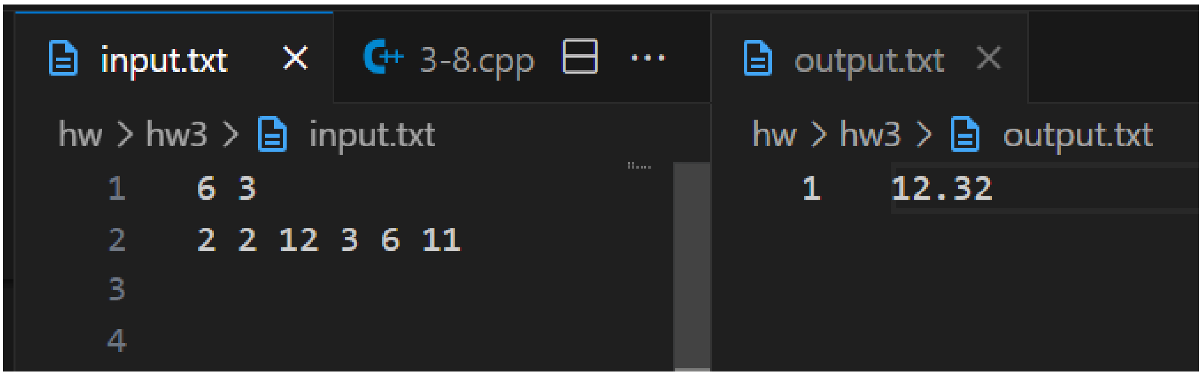
5. 目标:求 $dp[0][m]$，将整个序列分成 m 段的最优解。

2.代码

```
// 预处理平方和
double f(int i, int j) {
    return sqrt(prefix_square[j + 1] - prefix_square[i]);
}

double solve() {
    for (int i = n - 1; i >= 0; i--) {
        dp[i][1] = f(i, n - 1); // 初始化 dp[i][1]
```

```
    }

    for (int k = 2; k <= m; k++) { // 分成 k 段
        for (int i = n - 1; i >= 0; i--) { // 起点 i
            double tmp = INT_MAX;
            for (int j = i; j <= n - k; j++) {
                double maxt = max(f(i, j), dp[j + 1][k - 1]);
                tmp = min(tmp, maxt);
            }
            dp[i][k] = tmp;
        }
    }

    return dp[0][m];
}
```

3.结果



4.分析

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 $O(n^2 \times m)$。

空间复杂度：为 $O(n \times m)$。