# 3-2 最优批处理问题

## 问题描述

在一台超级计算机上，编号为1,2,3···的n个作业等待批处理。批处理的任务就是将这n个作业分成若干批，每批包含相邻的若干作业。从时刻0开始，分批加工这些作业。在每批作业开始前，机器需要启动时间S，而完成这批作业所需的时间是单独完成批中各个作业需要时间的总和。单独完成第i个作业所需的时间是 ti，所需的费用是它的完成时刻乘以一个费用系数fi。同一批作业将在同一时刻完成。例如，如果在时刻 T 开始一批作业 x,x+1,x+2···x+k, 则这一批作业的完成时刻均为T+S+(tx+tx+1tx+2···+tk)。最优批处理问题就是要确定总费用最小的批处理方案。

对于给定的待批处理的个作业，计算其总费用最小的批处理方案。

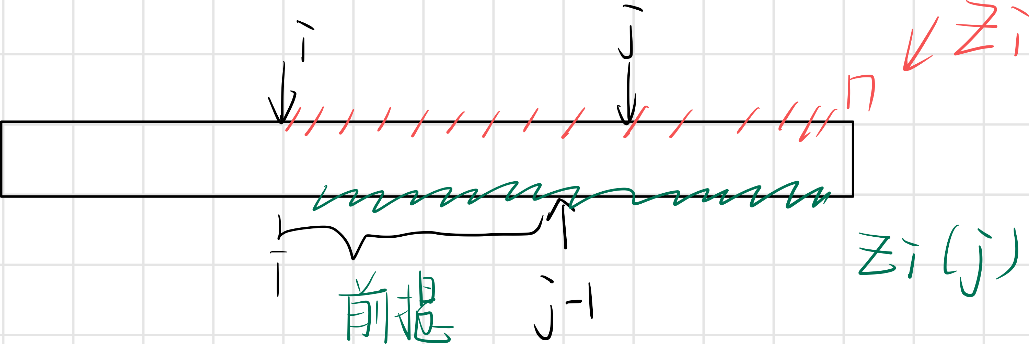
**输入数据由文件名为input.txt的文本文件提供**。文件的第1行为待处理的作业数n，第二行是启动时间，在接下来每行有两个数，分别为单独完成第i个作业所需的时间ti，和所需的费用fi。

**将计算结果输出到文件 output.txt。**最小费用。

## 动态规划分析

设是批处理作业序列i~n所需最小费用

是批处理作业序列下，i~(j-1)为第一批作业时所需最小费用



此时

代表的是,如果把前面的加上，先对i-j进行处理，那么**应该有基础时间**，即，所以在的表达式里存在。

由此可得递推式：

边界条件是

为了直接得到，我们考虑先把用数组保存起来

由此定义 = =

待入递推式可知

在实现中，循环比较i-n之间的数即可得到

时间复杂度为O(n)

## 算法实现

#include<fstream>

#include<iostream>

#include<string>

#include<vector>

using namespace std;

void readFile(int& n, int& S, vector<int>& fee, vector<int>& time)//读文件

{

fstream fin;

fin.open("input.txt", ios::in);

if (!fin)

{

cout << "wrong!!/a can't open it…" << endl;

exit(0);

}

int lines = 1;

while (!fin.eof())

{

string temp;

if (lines == 1)//第一行是总事件数量

{

getline(fin, temp);

n = stoi(temp);

lines++;

}

else if (lines == 2)//启动时间

{

getline(fin, temp);

S = stoi(temp);

lines++;

}

else

{

getline(fin, temp, ' ');

time.push\_back(stoi(temp));

getline(fin, temp);

fee.push\_back(stoi(temp));

}

}

fin.close();

}

void outFile(int result)//输出

{

fstream fin;

fin.open("output.txt", ios::out);

if (!fin)

{

cout << "wrong!!/a can't open it…" << endl;

exit(0);

}

fin << result;

fin.close();

}

int dyna(int n, int S, vector<int> time, vector<int> fee)

{

vector<int> z(n+1, 999999);

vector<int> st(n+1, 0);

vector<int> sw(n+1, 0);

for (int i = n - 1; i >= 0; i--)

{

st[i] = st[i + 1] + time[i];

sw[i] = sw[i + 1] + fee[i];

}

z[n] = 0;

for (int i = n - 1; i >= 0; i--)

{

for (int j = i + 1; j <= n; j++)

{

z[i] = min(z[i], z[j] + sw[i] \* (S + st[i] - st[j]));

}

}

return z[0];

}

int main()

{

int n, S;

vector<int> fee;

vector<int> time;

readFile(n, S, fee, time);

int result = dyna(n, S, time, fee);

outFile(result);

}

## 结果分析

### 测试文档1：

Input.txt output.txt

5 153

1

1 3

3 2

4 3

2 3

1 4

### 测试文档2：

Input.txt output.txt

7 573

2

1 5

2 3

3 2

5 14

4 5

1 3

5 4

# 乘法表问题

|  |  |  |  |
| --- | --- | --- | --- |
|  | **a** | **b** | **c** |
| **a** | b | b | a |
| **b** | c | b | a |
| **c** | a | c | c |

## 问题描述

定义于字母表∑{a,b,c)上的乘法表如表所示：

依此乘法表,对任一定义于∑上的字符串,适当加括号表达式后得到一个表达式。例如,对于字符串x=bbbba,它的一个加括号表达式为(b(bb))(ba)。依乘法表,该表达式的值为a。试设计一个动态规划算法,对任一定义于∑上的字符串x=x1x2…xn，计算有多少种不同的加括号方式,使由x导出的加括号表达式的值为a

**输入数据由文件名为input.txt的文本文件提供**。文件的第1行给出一个字符串

**将计算结果输出到文件 output.txt。**文件第一行的数是计算出的加括号方式数

## 动态规划分析

乘法表问题是计算所有加括号运算结果为a的情况数，并不要求输出加括号方式。那么可以从乘法的最小单元两个符号相乘的所有可能开始，接着在两个符号相乘的基础上计算三个符号相乘的所有可能。直到计算N长度的符号1-N的所有可能。

定义一个三维数组 dp[n][n][3], n为输入字符串的长度

dp [i][j][0]为从字符串中第i个元素到第j个元素的子串表达式值为a的加括号方式数

dp [i][j][1]为从字符串中第i个元素到第j个元素的子串表达式值为b的加括号方式数

dp [i][j][2]为从字符串中第i个元素到第j个元素的子串表达式值为c的加括号方式数

根据所给乘法表，可**得递推公式**

dp [i][j][0] = dp[i][k][0] \* dp[k+1][j][2] + dp[i][k][1] \* dp[k+1][j][2] + dp[i][k][2]\* dp[k+1][j][0]

即i~j子串表达式为a的数量， 等于i~k字串 (k+1)~j字串相乘为a的数量乘积，由于有三种方式可以得到a，所以相加为三个式子。

dp [i][j][1] = dp[i][k][0] \* dp[k+1][j][0] + dp[i][k][0] \* dp[k+1][j][1] + dp[i][k][1]\* dp[k+1][j][1]

dp [i][j][2] = dp[i][k][1] \* dp[k+1][j][0] + dp[i][k][2] \* dp[k+1][j][1] + dp[i][k][2]\* dp[k+1][j][2]

## 算法实现

#include <iostream>

#include<fstream>

#include<iostream>

#include<vector>

#include<string>

using namespace std;

string readFile()

{

fstream fin;

fin.open("input.txt", ios::in);

if (!fin)

{

cout << "open failed!" << endl;

exit(0);

}

string temp;

getline (fin,temp);

fin.close();

return temp;

}

void outFile(int result)

{

fstream fin;

fin.open("output.txt", ios::out);

fin << result;

fin.close();

}

int dnya(string S, int n)

{

vector<vector<vector<int>>> dp(n, vector<vector<int>>(n, vector<int>(3, 0)));//初始化一个大小为n\*n\*3的三维数组

//dp[i][j][0]表示i-j端为a， dp[i][j][1]表示i-j端为b, dp[i][j][2]表示i-j端为c 的加括号总数

for (int i = 0; i < n; i++)//初始化

{

if (S[i] == 'a')

dp[i][i][0] = 1;

else

dp[i][i][0] = 0;

if (S[i] == 'b')

dp[i][i][1] = 1;

else

dp[i][i][1] = 0;

if (S[i] == 'c')

dp[i][i][2] = 1;

else

dp[i][i][2] = 0;

}

for (int r = 2; r <= n; r++)//接着从长度为 2 的子字符串计算，直至计算好整串 str

{

for (int i = 0; i <= n - r; i++)

{

int j = i + r - 1;//计算str[i:j]

for (int k = i; k < j; k++)

{//根据题目中的表，计算加括号法

dp[i][j][0] += dp[i][k][0] \* dp[k + 1][j][2] + dp[i][k][1] \* dp[k + 1][j][2] + dp[i][k][2] \* dp[k + 1][j][0];

dp[i][j][1] += dp[i][k][0] \* dp[k + 1][j][0] + dp[i][k][0] \* dp[k + 1][j][1] + dp[i][k][1] \* dp[k + 1][j][1];

dp[i][j][2] += dp[i][k][1] \* dp[k + 1][j][0] + dp[i][k][2] \* dp[k + 1][j][1] + dp[i][k][2] \* dp[k + 1][j][2];

}

}

}

return dp[0][n - 1][0];

}

int main()

{

string S = readFile();

int n = S.size();

int result = dnya(S,n);

outFile(result);

}

## 结果分析

### 测试文档1：

Input.txt bbbba

output.txt 6

### 测试文档2：

Input.txt bbbcbaaccbac

output.txt 27548

# 汽车加油行驶问题

## 问题描述

给定一个 N\*N的方形网格，设其左上角为起点◎，坐标为（ 1，1），X轴向右为正， Y轴向下为正，每个方格边长为 1，如图所示。一辆汽车从起点◎出发驶向右下角终点▲，其坐标为（ N，N）。在若干个网格交叉点处，设置了油库，可供汽车在行驶途中加油。汽车在行驶过程中应遵守如下规则：

(1)汽车只能沿网格边行驶，装满油后能行驶 K条网格边。出发时汽车已装满油，在起点与终点处不设油库。

(2)汽车经过一条网格边时，若其 X坐标或 Y坐标减小，则应付费用 B，否则免付费用。

(3)汽车在行驶过程中遇油库则应加满油并付加油费用 A。

(4)在需要时可在网格点处增设油库，并付增设油库费用 C（不含加油费用 A）。

(5)(1)～(4)中的各数 N、K、A、B、C均为正整数，且满足约束：2 ≤ N ≤ 100，2 ≤ K ≤ 10。设计一个算法，求出汽车从起点出发到达终点的一条所付费用最少的行驶路线。

**输入数据由文件名为input.txt的文本文件提供**。文件的第1行是N，K，A，B，C的值，第二行起是一个N\*N的方阵

**将计算结果输出到文件 output.txt。**所需最少费用

## 动态规划分析

dp[x][y][0]表示从（1,1）到（x,y）所需最少费用。

dp[x][y][1]表示从汽车行驶到（x,y）还能行驶的网格边数。

**最终即求dp[N][N][0]**

假设我们现在到了（x,y）,上个位置有4种可能（不过考虑越界的情况），这四个位置都对应一个费用。先选择上个位置是a，现在再去考虑当前位置（到油站否，有没有有油），并加上相应的价格。这时我们会得到一个（x,y）处的费用=a位置费用+当前位置的费用。我们遍历4个位置，寻找到(x,y)处的最小费用即可。

**递推式可得**

dp[1][1][0]=0 dp[1][1][1]=k 初始

设s数组代表四个不同的走向s={ {1，0，0},{0，1，0},{-1，0，B},{0，-1，B}}

**dp[x][y][0] = min( dp[x + s[i][0]][y + s[i][1]][0], s[i][2]) 从上个位置到现在这个位置**

**dp[x][y][1] = dp[x + s[j][0]][ y + s[j][1]][1] – 1;**

**dp[x][y][0] = dp[x][y][0] + A dp[x][y][1] = k 是油库**

**dp[x][y][0] = dp[x][y][0] + C + A dp[x][y][1] = k 不是油库，但是没油了，必须建**

## 算法实现

#include <iostream>

#include<fstream>

#include<iostream>

#include<vector>

#include<string>

using namespace std;

void readFile(int &N, int &k , int &A, int &B, int &C, vector<vector<int>> &map)

{

fstream fin;

fin.open("input.txt", ios::in);

if (!fin)

{

cout << "open failed!" << endl;

exit(0);

}

string temp;

getline(fin, temp, ' ');

N = stoi(temp);

getline(fin, temp, ' ');

k = stoi(temp);

getline(fin, temp, ' ');

A = stoi(temp);

getline(fin, temp, ' ');

B = stoi(temp);

getline(fin, temp);

C = stoi(temp);

map.resize(N+1);

for(int i = 0; i < N+1; i++)

map[i].resize(N+1);

int line = 1;

while (!fin.eof())

{

for (int i = 1; i < N; i++)//注意是从map[1][1]开始存的

{

getline(fin, temp, ' ');

map[line][i] = stoi(temp);

}

getline(fin, temp);

map[line][N] = stoi(temp);

line++;

}

fin.close();

}

void outFile(int result)

{

fstream fin;

fin.open("output.txt", ios::out);

fin << result;

fin.close();

}

int dnya(int N, int K, int A, int B, int C, vector<vector<int>> map)

{

vector<vector<vector<int>>> dp(N + 1, vector<vector<int>>(N + 1, vector<int>(3, 0)));//初始化一个大小为n\*n\*3的三维数组

int s[4][3] = { {-1,0,0},{0,-1,0},{1,0,B},{0,1,B} };//4个方向，第三个参数是价格B

for (int i = 1; i <= N; ++i)

for (int j = 1; j <= N; ++j)

{

dp[i][j][0] = 99999;

dp[i][j][1] = K;

}//初始化

dp[1][1][0] = 0;

dp[1][1][1] = K;//初始化

for (int x = 1; x <= N; ++x)

{

for (int y = 1; y <= N; ++y)

{

if (x == 1 && y == 1) //当为起点，直接跳过，已做过初始化

continue;

int minmoney = 999999;

int minstep;//minmoney存储（x,y）上个位置的最小money

int tmpmoney, tmpstep;

for (int i = 0; i <= 3; ++i)//遍历上一个位置的所有可能情况

{

if (x + s[i][0]<1 || x + s[i][0]>N || y + s[i][1]<1 || y + s[i][1]>N) //超出边界，不做处理

continue;

tmpmoney = dp[x + s[i][0]][y + s[i][1]][0] + s[i][2];//按递推式写

tmpstep = dp[x + s[i][0]][y + s[i][1]][1] - 1;

if (map[x][y] == 1) //遇油站

{

tmpmoney += A;

tmpstep = K;

}

if (map[x][y] == 0 && tmpstep == 0 && (x != N || y != N)) //没油了

{

tmpmoney += A + C;

tmpstep = K;

}

if (minmoney > tmpmoney)//更新最小值

{

minmoney = tmpmoney;

minstep = tmpstep;

}

}

if (dp[x][y][0] > minmoney)//更新dp

{

dp[x][y][0] = minmoney;

dp[x][y][1] = minstep;

}

}

}

return dp[N][N][0];

}

int main()

{

int N, K, A, B, C;//N表示格点数，k表示一桶油能走多远，A表示加油费用，B表示倒退费用， C表示增设油库费用

vector<vector<int>> map;

readFile(N, K, A, B, C, map);

int result = dnya(N, K, A, B, C, map);

outFile(result);

}

## 结果分析

### 测试文档1：

Input.txt

9 3 2 3 6

0 0 0 0 1 0 0 0 0

0 0 0 1 0 1 1 0 0

1 0 1 0 0 0 0 1 0

0 0 0 0 0 1 0 0 1

1 0 0 1 0 0 1 0 0

0 1 0 0 0 0 0 1 0

0 0 0 0 1 0 0 0 1

1 0 0 1 0 0 0 1 0

0 1 0 0 0 0 0 0 0

output.txt 12

### 测试文档2：

Input.txt

5 2 3 1 4

0 1 0 1 1

1 0 0 1 1

1 0 0 0 1

0 1 0 0 1

0 0 0 0 0

output.txt 15