4-3 磁带最优存储问题

1. 问题描述

有 n 个程序{1,2,…, n }要存放在长度为 L 的磁带上。程序 i 存放在磁带上的长度是 Li， 1<= i<= n。这 n 个程序的读取概率分别是 p1,p2,…,pn,且 pi+p2+…+pn = 1。如果将这 n 个程序按 i1,i2,…,in 的次序存放，则读取程序 tr 所需的时间 tr=c*(Pi1Li2+Pi2Li2+…+Pir*Lir)。这 n 个程序的平均读取 时间为 t1+t2+…+tn。磁带最优存储问题要求确定这 n 个程序在磁带上的一个存储次序，使平均读取时间达到 最小。

输入数据由文件名为 input.txt 的文本文件提供。第 1 行是正整数 n，表示文件个数。接下来的 n 行中，每行有 2 个正整数 a 和 b，分别表示程序存放在磁带上的长度和读取概率。实际上第 k 个程序的读取概率为 ak/∑ai.对所有输入均假定 c=1

将计算结果输出到文件 output.txt。 最小平均读取时间。

2. 算法分析

由于该题能从局部最优到整体最优，所以选择贪心法来求解：要使平均读取时间最小，总读取时间需要最小，就要把 Pi*Li 最小的程序放在最前面。

3. 算法实现

```
#include<fstream>
#include <iostream>
#include <vector>
#include<string>
#include<algorithm>
using namespace std;

void readFile(vector<int>& len, vector<double>& p)
{
	fstream in;
	in.open("input.txt", ios::in);
	if (!in)
	{
		return;
	}
	string temp;
	getline(in, temp);
	int n = stoi(temp);
```

```
    len.resize(n);
    p.resize(n);
    int i = 0;
    while (!in.eof())
    {
        getline(in, temp, ' ');
        len[i] = stoi(temp);
        getline(in, temp);
        p[i] = stoi(temp);
        i++;
    }
    in.close();
}

void computeP(vector<double>& p)
{
    double sum = 0;
    for (int i = 0; i < p.size(); i++)
    {
        sum = sum + p[i];
    }
    for (int i = 0; i < p.size(); i++)
    {
        p[i] = p[i] / sum;
    }
}

vector<double> computePL(vector<int> len, vector<double> p)
{

    vector<double> temp(len.size(), 0);
    for (int i = 0; i < len.size(); i++)
    {
        temp[i] = len[i] * p[i];
    }
    return temp;
}

double minTime(vector<int> len, vector<double> p)
{
    vector<double> result(len.size(), 0);
    computeP(p);//计算概率
    result = computePL(len, p);
```

```
	sort(result.begin(), result.end());//贪心
	double mint = 0;
	for (int i = 0; i < result.size(); i++)
	{
		mint = result[i] * (result.size() - i) + mint;
	}

	return mint;

}

void writeFile(double mint)
{
	fstream in;
	in.open("output.txt", ios::out);
	in << mint;
	in.close();
}

int main()
{
	vector<int> len;
	vector<double> p;
	readFile(len, p);
	double mint = minTime(len, p);
	writeFile(mint);
}
```

4. 结果分析

测试文档 1：

input.txt

```
5
71 72
46 452
9 265
73 120
35 85
```

output.txt

```
85.6193
```

测试文档 2：

input.txt
6
23 11
45 25
49 66
75 152
11 457
40 55
output.txt
61.4869

4-9虚拟汽车加油问题

1. 问题描述

一辆虚拟汽车加满油后可以行驶 n km。途中有若干个加油站。设计一个有效的算法，指出应在那个加油站停靠加油，使沿途加油次数最少。
给定 n 和 k 个加油站位置，计算最少加油次数。
输入数据由文件名为 input.txt 的文本文件提供。第 1 行有两个整数 n 和 k，表示汽车加满油后可行驶 n km,且路途中有 k 个加油站。接下来的 1 行中有 k+1 个整数，表示第 k 个加油站与 k-1 个加油站之间的距离。第 0 个加油站表示处出发地，汽车已加满油，第 k+1 个加油站便是目的地。
将计算结果输出到文件 output.txt。最少加油次数

2. 算法分析

每当到达一个加油站的时候，都要进行一次判断，看看剩余的油量能不能维持到下一个加油站。如果可以的话，接着前行；如果不行的话，在当前加油站加满油在出发，此时更新油箱数据为已满。(能不加就不加，减少加油次数)

3. 算法实现

```
#include <iostream>
#include <fstream>
#include <vector>
#include<string>
using namespace std;
```

```
void readFile(int &n, int &k, vector<int> &dis)
{
	fstream in;
	in.open("input.txt", ios::in);
	if (!in)
	{
		return;
	}
	string temp;
	getline(in, temp, ' ');
	n = stoi(temp);
	getline(in, temp);
	k = stoi(temp);
	dis.resize(k + 1);
	for(int i = 0; i < k; i++)
	{
		getline(in, temp, ' ');
		dis[i] = stoi(temp);
	}
	getline(in, temp);
	dis[k] = stoi(temp);
	in.close();
}

int getGas(int n,int k, vector<int> dis)
{
	int cnt = 0;
	int sum = 0;
	int i = 0;
	while(i < k + 1)
	{
		if (sum >= n)
		{
			if (sum > n)//回退
			{
				sum = sum - dis[i];
			}
			cnt++;
			sum = 0;
			//cout << i << "    ";
		}
		else
		{
			sum = sum + dis[i];
```

```
                if(sum <= n)
                    i++;
            }
        }
        //cout << "end" << endl;
        return cnt;
}

void writeFile(int cnt)
{
        fstream in;
        in.open("output.txt", ios::out);
        in << cnt;
        in.close();
}

int main()
{
        int n = 0, k = 0;
        vector<int> dis;
        readFile(n, k, dis);
        int cnt = getGas(n, k, dis);
        writeFile(cnt);
}
```

4. 结果分析

测试文档 1：

input.txt

7 7

1 2 3 4 5 1 6 6

output.txt

4

测试文档 2：

input.txt

8 9

1 2 3 4 5 1 6 6 7 1

output.txt

5