**习题6**

**2：**

设Cuv表示多段图的有向边<u,v>上的权值，将从源点s到终点t的最短路径长度记为d(s,t),考虑原问题的部分解d(s,v)

有：

1. d(s,v)=Csv (<s,v>属于E)
2. d(s,v)=min{d(s,u)+Cuv} (<u,v>属于E)

那么对于题目所示多段图，具体求解过程如下所示：

1. .首先求解初始子问题，可直接获得：

d(0,1)=c01=5(0->1)

d(0,2)=c02=3(0->2)

1. 再求解下一阶段的子问题：

d(0,3)=d(0,1)+c13=6(1->3)

d(0,4)=min{d(0,1)+c14,d(0,2)+c24}=8(1->4)

d(0,5)=min{d(0,1)+c15,d(0,2)+c25}=10(2->5)

d(0,6)=d(0,2)+c26=3+6=9(2-->6)

1. 再求解下一阶段的子问题：

d(0,7)=min{d(0,3)+c37,d(0,4)+c47}=11(4-->7)

d(0,8)=min{d(0,3)+c38,d(0,4)+c48,d(0,5)+c58,d(0,6)+c68}=min{14,13,13,17}=13 (4-->8或者5-->8)

d(0,9)=min{d(0,5)+c59,d(0,6)+c69}=13(5-->9或者6-->9)

1. 再求解下一阶段的子问题：

d(0,10)=min{d(0,7)+c710,d(0,8)+c810,d(0,9)+c9,10}=min{14,18,13+c910}=14(7-->10)

（备注：因为题目中未给出9到10的路径长度，在此处我姑且认为最短路径为14）

d(0,11)=min{d(0,7)+c711,d(0,8)+c811,d(0,9)+c911}=min{16,15,19}=15(8-->11)

1. 直到最后一个阶段子问题：

d(0,12)=min{d(0,10)+c1012,d(0,11)+c1112}=18(10-->12

或者11-->12)

得到三条路径，分别为：

(1) 0-->1-->4-->7-->10-->12 路径长度为：18

(2) 0-->1-->4-->8-->11-->12 路径长度为：18

(3) 0-->2-->5-->8-->11-->12 路径长度为：18

综上所述：该多段图从0到12最短路径值为18

伪代码：

多段图的最短路径问题：

输入：多段图的代价矩阵

输出：最短路径长度及路径c[n][n]

1. 循环变量j从1-（n-1）重复下述操作，执行填表工作：

1.1考察顶点j的所有入边，对于边<i,j>属于E，执行下述操作：

1.1.1 cost[j]=min{cost[i]+c[i][j]};

1.1.2 path[j]=使cost[i]+c[i][j]最小的i；

1.2 j++

1. 输出最短路径长度cost[n-1];
2. 循环变量i=path[n-1],循环直到path[i]=0,输出最短路径经过的顶点：

3.1 输出path[i];

3.2 i=path[i]。

**3：**

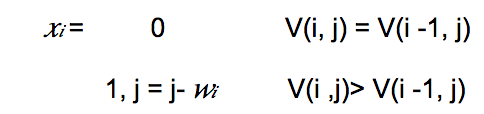
解：令V（i, j ）表示在前i 个（ 1≤ i≤n）个物品中能够装入容量为j(1≤ j≤C)的背包中的物品的最大值。

动态规划函数：

V(i ,0) = V(0, j) = 0  （6.11）

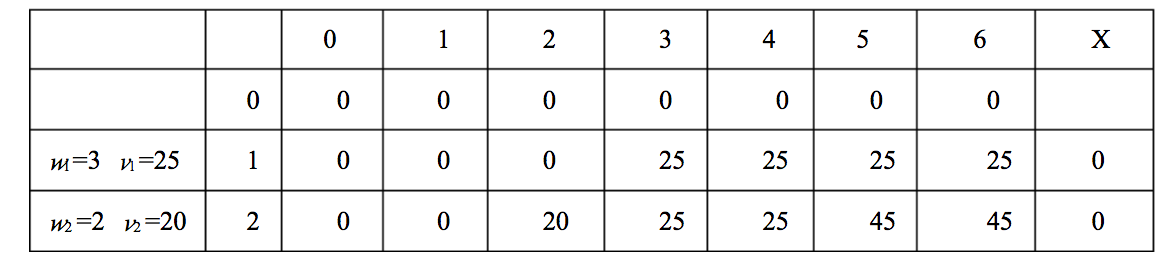
V(i , j) =  V（i-1, j）

函数：



根据动态规划函数，用一个（n+1) x (C+1)的二维表V，V[i][j]表示把前物品装入容量为

的背包中获得的最大价值。根据式（6.11）把表的第0行和第0列初始化为0，然后一行一行地计算V[i][j]，如下图所示；



从图中可以看到，装入背包的物品的最大价值是65，所以装入背包的物品为

{ 0,0,1,0,1}

**4 ：**

(1)若xm=yn,则zk=xm=yn，且Zk-1是Xm-1和Yn-1的最长公共自序列；

(2)若xm≠yn,且zk≠xm，则Zk是Xm-1和Yn的最长公共自序列；

(3)若xm≠yn,且zk≠yn，则Zk是Xm和Yn-1的最长公共自序列；

设L(m,n)表示序列X={x1,x2,…,xm}和Y={y1,y2,…,yn}的最长公共子序列的长度

L表示已经决策的长度

S表示每个决策的状态

L(0,0)=L(0,j)=0 1≤i≤m, 1≤j≤n

L(i-1,j-1)+1 xi=yi,i≥1,j≥1

L(i,j)=

max{L(i,j-1),(L(i-1,j)} xi≠yi,i≥1,j≥1

xi=yi

S(i,j)= xi≠yi 且L(i,j-1)≥L(i-1,j)

xi≠yi 且L(i,j-1)< L(i-1,j)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **x** | **z** | **y** | **z** | **z** | **y** | **x** |
|  | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **z** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **1** |
| **x** | **0** | **1** | **1** | **1** | **1** | **2** | **2** | **2** |
| **y** | **0** | **1** | **1** | **2** | **2** | **2** | **2** | **2** |
| **y** | **0** | **1** | **1** | **2** | **2** | **3** | **3** | **3** |
| **z** | **0** | **1** | **1** | **2** | **2** | **3** | **3** | **4** |
| **x** | **0** | **1** | **1** | **2** | **3** | **3** | **3** | **4** |
| **z** | **0** | **1** | **2** | **2** | **3** | **3** | **4** | **4** |

**习题7**

**1：**

贪心算法是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。

①单价分别为（5，5/3，3，1，6，4.5，3）；

     按单价的大小分别为重量和价值重新排序! 排序为：

     重量：（1，2，4，5，1，3，7）

     价值：（6，10，18，15，3，5，7）

                   x1，x2，x3，x4，x5，x6，x7

② 当 W=15

     （1）1<15，x1=1，  C=15-1=14；     [v=6]

     （2）2<14，x2=1，  C=14-2=12；     [v=10]

     （3）4<12，x3=1，  C=12-4=8；      [v=18]

     （4）5<8， x4=1，   C=8-5=3；      [v=15]

     （5）1<3， x5=1，   C=3-1=2；      [v=3]

     （6）3>2， x6=2/3，C=2-3\*2/3=0；   [v = 5\*2/3 = 10/3]

最后结果：x1=1  x2=1  x3=1  x4=1  x5=1  x6=2/3   x7=0

最大价值：6+10+18+15+3+10/3  =  55又1/3

**6**

假设原问题为T,而我们已经知道了某个最优服务系列, 即最优解为A={t(1),t(2),⋯.t(n)}(其中t(i)为第i个用户 需要的服务时间),则每个用户等待时间为:

T(1)=t(1);

T(2)=t(1)+t(2);

 ⋯⋯

T(n)=t(1)+t(2)+t(3)+⋯⋯t(n);

那么总等待时间,即最优值为:

TA=n\*t(1)+(n-1)\*t(2)+⋯+(n+1-i)\*t(i)+⋯2\*t(n-1)+t (n)

由于平均等待时间是n个顾客等待时间的总和除以n,故实际上就是求使顾客等待时间的总和最小的服务次序。

贪心策略如下:对服务时间最短 的顾客先服务的贪心选择策略。首先对需要服务时间最短的顾客进行服务,即做完第一次选择后,原问题T变成了需对 n-1 个顾客服务的新问题 T’。新问题和原问题相同,只是问 题规模由n减小为n-1。

基于此种选择策略,对新问题T’,选择n-1顾客中选择服务时间最短的先进行服务,如此进行下 去,直至所有服务都完成为止。

#include<fstream.h>

#include<iostream.h>

#include<stdlib.h>

#include<iomanip.h>

long n=-1;

long \*wait;

voidinputData() {

ifstream fin;

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

if(!fin) {

cout<<"File Open Error!"<<endl; return;

}

fin>>n;

wait=new long[n];

for(long i=0;i<n;i++) {

fin>>wait[i];

}

fin.close();

}

void ShellSort(long \*a)

{

long i,j,x,gap=n/2;

while(gap>0) {

for(i=gap;i<n;i++) {

j=i-gap;

while(j>=0) {

if(a[j]>a[j+gap])

{x=a[j];a[j]=a[j+gap];a[j+gap]=x;j=j-gap; }

else { j=-1;}

}

}

gap=gap/2; }

}

double AveWait(long \*a){

double ave=0.0;

ShellSort(a);

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

ave+=1.0\*(n-i)\*a[i];

}

ave/=n;

return ave;

}

void outputData(double out) {

ofstream fout;

fout.open("output.txt"); fout<<setiosflags(ios::fixed)<<setprecision(2)<<out<<endl;

fout.close();

}

void main() {

inputData();

if(n!=-1) {

double avewait=AveWait(wait);

outputData(avewait);

}

}

时间复杂度分析：

程序主要是花费在对各顾客所需服务时间的排序和贪心算法,即计算平均服务时间上面。其中,贪心算法部分只有 一重循环影响时间复杂度,其时间复杂度为O(n);而排序在 这里采用希尔(Shell)排序,其时间复杂度为O(nlogn)。因此,综合来看算法的时间复杂度为O(nlogn)

**附录：**

**6.2代码**

#include<stdio.h>

const int MAX=1000;

int arc[100][100];

int CreatGraph()

{

int i,j,k;

int weight;

int vnum,arcnum;

printf("请输入顶点的个数和边的个数：");

scanf("%d%d",&vnum,&arcnum);

for(i=0;i<vnum;i++)

for(j=0;j<vnum;j++)

arc[i][j]=MAX;

for(k=0;k<arcnum;k++)

{

printf("请输入边的两个顶点和权值：");

scanf("%d%d%d",&i,&j,&weight);

arc[i][j]=weight;

}

return vnum;

}

int Back(int n)

{

int i,j;

int cost[100],path[100];

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

{

cost[i]=MAX;

path[i]=-1;

}

cost[0]=0;

path[0]=-1;

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

{

for(i=j-1;i>=0;i--)

{

if(arc[i][j]+cost[i]<cost[j])

{

cost[j]=arc[i][j]+cost[i];

path[j]=i;

}

}

}

printf("%d",n-1);

i=n-1;

while(path[i]>=0)

{

printf("<--%d",path[i]);

i=path[i];

}

return cost[n-1];

}

int main()

{

int n;

n=Create();

printf("\n%d\n",Back(n));

return 0;

}

6.4代码：

#include <iostream>

#include <string>

using namespace std;

int main()

{

string str1 = "xzyzzyx";

string str2 = "zxyyzxz";

int x\_len = str1.length();

int y\_len = str2.length();

int arr[50][50] ={{0,0}};

int i = 0;

int j = 0;

for(i = 1; i <= x\_len; i++)

{

for(j = 1; j <= y\_len; j++)

{

if(str1[i - 1] == str2[j - 1])

{

arr[i][j] = arr[i - 1][j - 1] + 1;

}

else if(arr[i][j - 1] >= arr[i - 1][j])

arr[i][j] = arr[i][j - 1];

else

arr[i][j] = arr[i -1][j];

}

}

for(i = 0 ; i <= x\_len; i++)

{

for( j = 0; j <= y\_len; j++)

{

cout << arr[i][j] << " ";

}

cout << endl;

}

for(i = x\_len, j = y\_len; i >= 1 && j >= 1;)

{

if(str1[i - 1] == str2[j - 1])

{

cout << str1[i - 1] << " ";

i--;

j--;

}

else if(arr[i][j -1] > arr[i - 1][j])

j--;

else

i--;

}

cout << endl;

return 0;

}

**7.1代码：**

#include <iostream>

using namespace std;

const int n = 3;

int KnapSack(int w[ ], int v[ ], int n, int C);

int main( )

{

Int w[n] = {10, 30, 20}, v[n] = {50, 120, 60};

    int C = 50;

    int value = KnapSack(w, v, 3, C);

    cout<<"背包获得的最大价值是："<<value<<endl;

    return 0;

}

int KnapSack(int w[ ], int v[ ], int n, int C)

{

   double x[10] = {0};

   int maxValue = 0;

   for (int i = 0; w[i] < C; i++)

    {

        x[i] = 1;

        maxValue += v[i];

        C = C - w[i];

    }

    x[i] = (double)C/w[i];

    maxValue += x[i] \* v[i];

    return maxValue;

}

**7.6代码：**

#include <iostream>

#include <algorithm>

#include <cstdio>

#include <cstring>

#include <string>

using namespace std;

int a[1000];

int ser[100];//服务窗口的顾客等待时间

int sum[100];//服务窗口顾客等待时间的总和。

int main()

{

int n,s;

    while(~scanf("%d%d",&n,&s))

    {

        for(int i=0;i<n;i++)

        {

         scanf("%d",&a[i]);

        }

        sort(a,a+n);

        memset(ser,0,sizeof(ser));

        memset(sum,0,sizeof(sum));

        int i=0;

        int j=0;

        while(i<n)

        {

            ser[j]+=a[i];

            sum[j]+=ser[j];

            i++,j++;

            if(j==s)

                j=0;

        }

        double t=0;

        for(int i=0;i<n;i++)

        {

            t+=sum[i];

        }

        t/=n;

        printf("%0.3lf\n",t);

    }

    return 0;

}