**140页，第7题**

1. **对于一个包含100万随机数的数组排序，快速排序比插入排序快多少倍？**

**答**：420倍，根据算法程序得出。

#include<stdio.h>

#include <time.h>

#include <stdlib.h>

#define N 100000 //随机生成排序数的个数

//插入排序法

void InsertSort(int\* pData,int Count)

{

int iTemp;

int iPos;

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

{

iTemp = pData[i];

iPos = i-1;

while((iPos>=0) && (iTemp<pData[iPos]))

{

pData[iPos+1] = pData[iPos];

iPos--;

}

pData[iPos+1] = iTemp;

}

}

//快速排序法

void run(int \*pData,int left,int right)

{

int middle,i,j;

int t;

i=left;

j=right;

middle=pData[(left+right)/2]; //找到中间元素

do{

while((pData[i]<middle)&&(i<right))

i++;

while((pData[j]>middle)&&(j>left))

j--;

if(i<=j)

{ //交换元素

t=pData[i];

pData[i]=pData[j];

pData[j]=t;

i++;

j--;

}

}while(i<=j);

if(left<j)

run(pData,left,j);

if(right>i)

run(pData,i,right);

}

void QuickSort(int \*pData,int Count)

{

run(pData,0,Count-1);

}

int main(void)

{

clock\_t start;

int arry[N],a,b,c;

int i = 0;

srand(time(NULL));

printf("生成100000个随机数~\n");

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

arry[i] = rand();

printf("插入排序中~\n");

start = clock();

InsertSort(arry, N);

printf("插入排序排序 %d 个数的时间：%d\n", N, clock()-start);

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

arry[i] = rand();

printf("快速排序中~\n");

start = clock();

QuickSort(arry, N);

printf("快速排序排序 %d 个数的时间：%d\n", N, clock()-start);

return 0;

}

1. **是非题：对于n>1的n元素数组，是否存在插入排序比快速排序更快的情形？**

**答**：当划分的两个子数组里面存在0和n-1时，存在最坏的划分情况，此时算法运行时间的递归式是T(n) = T(n-1) + T(0) + O(n) = T(n - 1) + O(n)此时解为O(n^2)。

因此，如果在算法的每一层递归上，划分都是最大程度的不平衡，那么，算法的时间复杂度是O(n^2)，也就是说，在最坏的情况下，快速排序的运行时间并不比插入排序好，此外，当输入数组已经完全有序的时候，快速排序的时间复杂度依然是O(n^2)，而在同样的情况下，插入排序的时间复杂度为O(n)，此时插入排序比快速排序更快。

**第225页 第6题**

切割木棍问题 为下列问题设计一个动态规划算法。已知小木棍的销售价格pi和长度i相关，i=1,2...，n如何把长度为n的木棍切割为若干长度为整数的小木棍，使得能获得总销售价格最大？该算法的时间效率和空间效率各是多少？  
 **答：**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | ... | n |
| Pi | 5 | 7 | 3 | 8 | 9 | ... | Pn |

由题意得知木棍的长度为i（i=1，2，3，...,n）

由表格假设数据得知木棍长度i的价格Pi（Pi=5，7，3，8，9，...,Pn）

得长度为n的最大价值W(n)=MAX(price(i)+price(n-i))

长度为n的价格有两种:

第一种：原始长度为n时的价格W=Pn ；

第二种：加n分割为个小块再加起来的价格：

W[i][P]=max{f[i-1][P],f[i-1][V]-P[i]]+W[i]}

下面的表格是假设有一个长度为五的棍子怎样切才能最大价值，得知为

（1，4）是价值最大为13。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | Pi | Wi | N=1 | 2 | 3 | 4 | 5 |
| 1 | 5 | 1 | 5 | 5 | 5 | 5 | 5 |
| 2 | 7 | 2 | 5 | 7 | 12 | 12 | 12 |
| 3 | 3 | 3 | 5 | 7 | 12 | 12 | 12 |
| 4 | 8 | 4 | 5 | 7 | 12 | 12 | 13 |
| 5 | 9 | 5 | 5 | 7 | 12 | 12 | 13 |

先从最短的长度 1开始找相对应长度可得到的最大价值，因为长度1无法再分，所以W[1] 就为原始长度价格 p[1]

然后长度2的可得到的最大价值W[2]就为W[1] +W[1] 和 p[2]之中最大的那个。3的可得到的最大价值W[3]就为W[1]+W[2] 、W[2]+W[1]和p[3]中最大的那个。因为比当前长度小的所有整数长度的对应的最大价格都是已知的，所以长度为n时只需要找到W[1]+W[n-1]、W[2]+W[n-2]、…、W[i]+W[n-i]、…、W[n-1]+W[1]、p[n]中最大的值，再赋值给W[n]

此算法的时间效率是O(N^2)，空间效率是O (N)。

**第229页 第3题**

**对于背包问题的自底向上的动态规划算法，请证明：**

1. **它的时间效率属于θ（nW）**
2. **它的空间效率属于θ（nW）**
3. **从一张填好的动态规划表中求得最优子集的组合所用的时间属于O（n**）

**234页 第11题**

**矩阵连乘 考虑如何使得在计算n个矩阵的乘积A1，A2，。。An时，总的乘法次数最小，这些矩阵的维度分别为的d0\*d1，d1\*d2，。。。dn-1\*dn。假设所有两个矩阵的中间乘积都使用蛮力算法（基于定义）计算。**

**答：**设计一个求n个矩阵乘法最优次数的动态规划。

设M[i,j]为计算Ai·…·Aj所需的最优(最小)乘法数。如果k是最后一个矩阵乘积的第一个因子中最后一个矩阵的索引，则

M (i, j) k = max 1≤≤j−1 {M (i (k) + M [k + 1, j] + di−1 dkdj} 1≤我< j≤n M[我]= 0。

这个递归，与最优二叉搜索树问题的递归非常相似，建议对角填充n +1×n +1表，如下算法所示:

37/5000

矩阵链乘法(D[0..n])

//用动态规划求解矩阵链乘法问题

//输入:一个数组D[0..]n个矩阵的维数n]

//输出:相乘所需的最小乘法数

//给定维数的n个矩阵链和表T[1..n,1..n]

//为了得到乘法的最优阶

for i ←1 to n do M[i,i] ←0

for d←1 to n−1 do //对角线数

for i←1 to n−d do j ← i + d

minval ←∞

for k ← i to j −1 do

temp ← M[i,k]+M[k +1,j]+D[i−1]∗D[k]∗D[j]

if temp < minval

minval ← temp

kmin ←k

T [i,j] ← kminreturn M[1,n ],T

为求矩阵链乘的最优阶，调用最优乘阶(1,n):

算法最优顺序(i, j)

//输出n个矩阵相乘的最优阶

//输入:Ai中第一个和最后一个矩阵的索引i和j…Aj与表T[1..n,1..]n]由矩阵链乘法生成

//输出: 用括号表示的Ai,Aj用于最优乘法

if i = j print(“Ai”)

else

k ← T[i,j]

print(“(”)

OptimalOrder(i,k)

OptimalOrder(k +1,j)

print(“)”)

**第249页 第7题**

**谣言传播 有n个人，每个人都拥有不同的谣言。通过发电子信息，他们想互相共享所有的谣言。假定发送者会在信息中包含他已知的所有谣言，而且一条信息只有一个收信人。 设计一个贪心算法，保证在每个人都能获得所有谣言的条件下，使发送的信息数最小。**

**答：**将这n个人标记为1, 2, …, n，按照1发信给2, 2发信给3, 3发信给4，…，n-1发信给n的方式发送谣言，该贪心算法基于每次发信都使得当前收信人掌握的谣言更多，最后由n将所有谣言发送给其他n-1个人。

发送信息总数为2n-2，这是最小的发信息数。因为每增加一个人，至少需要增加两次发送信息，当n=2是，发送信息数为2，归纳法可证明2n-2为最小发信息数。

#include<bits/stdc++.h>

#include<stdio.h>

using namespace std;

int v[101][101],d[101][101];

int t[101];

int n,m,x,y,N;

int main()

{

memset(v,63,sizeof(v));//预处理

scanf("%d",&n);

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

{

scanf("%d",&N);

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

{

scanf("%d%d",&x,&y);

v[i][x]=y;

}

}

for(int k=1;k<=n;k++)

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

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

{

if(v[i][k]+v[k][j]<v[i][j])

v[i][j]=v[i][k]+v[k][j];

}//搜索最短路径

int minn=1e9,sum;

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

{

int maxx=0;

for(int k=1;k<=n;k++)

if(v[i][k]>maxx&&i!=k)

maxx=v[i][k];//找出最晚得知消息的权值，若不能到达则maxx自动赋值为最大值

if(maxx<minn) minn=maxx,sum=i;//与当前最短路径作比较

}

if(minn==1e9)

printf("disjoint");//若搜完后发现不能遍历整个图，则输出"disjoint"

else

printf("%d %d",sum,minn);//否则输出最短路径及开始点

}

**264 第9题**

1. **写一个程序，为给定的英文文本构造一套哈夫曼编码，并对该文本编码。**

**b.写一个程序，对一段用哈夫曼码编码的英文文本进行解码。**

**C.做一个实验，测试对包含1000 个词的一段英文文本进行哈夫曼编码时，典型的压缩率位于什么样的区间。**

**d.对编码程序做个实验， 测试如果用标准的估计频率代替英文文本中字符的实际出现频率，该程序的压缩率会有什么样的变化。**

#include<iostream>

#include<string>

#include<iomanip>

#include <cstdlib>

using namespace std;

struct Element

{

char ch;

int weight;

int lchild, rchild, parent;

};

struct HCode

{

char data;

char code[100];

};

int Select(Element H[], int i) //选择两个最小的

{

int min = 11000;

int min1;

for (int k = 0; k<i; k++)

{

if (H[k].weight<min && H[k].parent == -1)

{

min = H[k].weight;

min1 = k;

}

}

H[min1].parent = 1;

return min1;

}

void Reverse(char c[]) //将字符串倒置

{

int n = 0;

char temp;

while (c[n + 1] != '\0')

{

n++;

}

for (int i = 0; i <= n / 2; i++)

{

temp = c[i];

c[i] = c[n - i];

c[n - i] = temp;

}

}

void HuffmanTree(Element H[], int w[], int n) //构造哈夫曼树

{

int i1 = 0, i2 = 0;

for (int i = 0; i<2 \* n - 1; i++)

{

H[i].lchild = -1;

H[i].parent = -1;

H[i].rchild = -1;

}

for (int l = 0; l<n; l++)

{

H[l].weight = w[l];

}

for (int k = n; k<2 \* n - 1; k++)

{

int i1 = Select(H, k);

int i2 = Select(H, k);

if (i1>i2)

{

int temp;

temp = i1;

i1 = i2;

i2 = temp;

}

H[i1].parent = k;

H[i2].parent = k;

H[k].weight = H[i1].weight + H[i2].weight;

H[k].lchild = i1;

H[k].rchild = i2;

}

}

void CreateCodeTable(Element H[], HCode HC[], int n) //输出哈弗曼编码表

{

HC = new HCode[n];

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

{

HC[i].data = H[i].ch;

int ic = i;

int ip = H[i].parent;

int k = 0;

while (ip != -1)

{

if (ic == H[ip].lchild) //左孩子标'0'

HC[i].code[k] = '0';

else

HC[i].code[k] = '1'; //右孩子标'1'

k++;

ic = ip;

ip = H[ic].parent;

}

HC[i].code[k] = '\0';

Reverse(HC[i].code);

}

cout << setiosflags(ios::left)

<< setw(5) << "n"

<< setw(12) << "weight"

<< setw(12) << "LChild"

<< setw(12) << "RChild"

<< setw(12) << "parent"

<< setw(12) << "char"

<< setw(12) << "code"

<< endl;

for (int i = 0; i<2 \* n - 1; i++)

{

if (i<n)

{

cout << setiosflags(ios::left)

<< setw(5) << i

<< setw(12) << H[i].weight

<< setw(12) << H[i].lchild

<< setw(12) << H[i].rchild

<< setw(12) << H[i].parent

<< setw(12) << HC[i].data

<< setw(12) << HC[i].code

<< endl;

}

else

cout << setiosflags(ios::left)

<< setw(5) << i

<< setw(12) << H[i].weight

<< setw(12) << H[i].lchild

<< setw(12) << H[i].rchild

<< setw(12) << H[i].parent

<< setw(12) << "\\0"

<< setw(12) << "\\0"

<< endl;

}

}

void Decode(Element H[], HCode HC[], int n, char \*s) //解码函数

{

cout << "解码数据为：";

int i = 2 \* (n - 1); //根结点

while (\*s != '\0')

{

if (\*s == '0')

i = H[i].lchild;

else

i = H[i].rchild;

if (H[i].lchild == -1)

{

cout << H[i].ch;

i = 2 \* n - 2;

}

s++;

}

cout << endl;

}

int main()

{

Element H[20];

HCode HC[20];

int n;

int select;

while (1)

{

system("cls");

cout << "\t \*------------------------------------------\*\n";

cout << "\t \* 1——输入编译字符集 \*\n";

cout << "\t \* 2——输出编码表 \*\n";

cout << "\t \* 3——解码 \*\n";

cout << "\t \* 4——退出 \*\n";

cout << "\t \*------------------------------------------\*\n";

cout << "\t 你要输入的编号是（1--4）:";

cin >> select;

if (select == 4) break;

switch (select) {

case 1:

{

cout << endl;

cout << "请输入字符集大小：";

cin >> n;

cout << endl;

char s;

HCode HC[20];

int e[20];

for (int t = 0; t < n; t++)

{

cout << "请输入第" << t + 1 << "个字符：";

cin >> s;

H[t].ch = s;

HC[t].data = H[t].ch;

cout << "请输入该字符的权值：";

cin >> e[t];

cout << endl;

}

HuffmanTree(H, e, n);

system("pause");

break;

}

case 2:

CreateCodeTable(H, HC, n);

system("pause");

break;

case 3:

{

cout << endl;

cout << "请输入解码数据：";

char s[200] = { '\0' };

cin >> s;

Decode(H, HC, n, s);

system("pause");

break;

}

default:

cout << "没有此选项,请重新选择！" << endl;

}

}

}

**331页 第7题**

**用回溯法生成{1, 2, 3, 4}的所有排列**。答：代码如下

#include <iostream>

using namespace std;

void swap(int arr[], int i, int j)

{

int tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

}

void FullSort(int arr[], int start, int end)

{

if(start == end)

{

for(int i = 0; i <= end; i++)

{

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

}

cout << endl;

return;

}

else

{

for(int i = start; i <= end; i++)

{

cout << "i = " << i << endl;

swap(arr, i, start);

cout << "arr[start] = " << arr[start] << endl;

FullSort(arr, start+1, end);

swap(arr, i, start);

}

}

}

int main()

{

int arr[] = {1,2,3,4};

FullSort(arr, 0, sizeof(arr) / sizeof(arr[0]) - 1);

}

**338页第7题**  
**写一个程序用分支界限算法对背包问题求解**

#include <stdio.h>

#include <stdlib.h>

typedef struct QNode{

int value; //当前结点的总价值

int weight; //当前的总重量

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct{//结构

QueuePtr front;

QueuePtr rear;

}Queue;

int initQueue(Queue &Q){

Q.front=Q.rear=(QueuePtr)malloc(sizeof(QNode));

if(!Q.front)

return -1;

Q.front->next=NULL;

return 1;

}

int emptyQueue(Queue Q){

if (Q.front==Q.rear)

return 1;

else

return 0;

}

int destroyQueue(Queue &Q){

while(Q.front){

Q.rear=Q.front->next;

free(Q.front);

Q.front=Q.rear;

}

return 1;

}

int enQueue(Queue &Q, int value, int weight)

{

QueuePtr p=(QueuePtr)malloc(sizeof(QNode));

if(!p)

return -1;

p->value=value;

p->weight=weight;

p->next=NULL;

Q.rear->next=p;

Q.rear=p;

return 1;

}

int deQueue(Queue &Q, int &value, int &weight)

{

QueuePtr p;

if(Q.front==Q.rear)

return -1;

p=Q.front->next;

value=p->value;

weight=p->weight;

Q.front->next=p->next;

if(Q.rear==p)

Q.rear=Q.front;

free(p);

return 1;

}

Queue loadingQueue;

int bestvalue, n;

void inQueue(int value, int weight, int i)

{

if(i==n-1)

{

if(value>bestvalue)

bestvalue=value;

}

else

enQueue(loadingQueue,value,weight);

}

int main()

{

int i,j,k;

int \*w, \*v, ew, ev;

int c;

printf("输入物品的数量和背包的容量:");

scanf("%d%d",&n,&c);

w=new int[n];

v=new int[n];

printf("输入重量:");

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

scanf("%d",&w[i]);

printf("输入价值:");

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

scanf("%d",&v[i]);

initQueue(loadingQueue);

enQueue(loadingQueue,-1,0);

i=0; //层数

ew=0; //扩展结点对应的载重量

ev=0;

while(true)

{

if(ew+w[i]<=c)

inQueue(ev+v[i],ew+w[i],i);

inQueue(ev,ew,i);

deQueue(loadingQueue, ev,ew);

if(ev==-1) //同层结点尾部

{

if(emptyQueue(loadingQueue))

{

printf("结果是：%d\n",bestvalue);

}

enQueue(loadingQueue,-1,0);

deQueue(loadingQueue, ev,ew);

i++;

}

}

return 0;

}