140页

7.（a） 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：1000000=106

插入排序平均效率是n2，快速排序的平均效率是1.39\*n\*log2 n。

计算n2/n\*log2 n=1012/1.39\*106 \*log2 106≈36000倍

7（b） 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：插入排序：当n元数组是最优输入，即有序数组的输入效率是n。

快速排序：对最优输入，效率是n\*log2n。

当n=2时，两种排序理论上一样快。当n>2时，插入排序比快速 排序快。所以存在插入排序比快速排序更快的情形。

225页

1. ------切割木棍问题------

对于长度为n的木棍，他的递推关系是：profit[n] = max(pi[i] + profit[length - seg[i]]), 其中i = 1,2,3,...n; 切割长度（seg） 销售价格（pi）

解决的主要思路是，先求解长度为1的最大收益，再到2,3.....一直到n，而每次求解的解都储存起来，时间复杂度为O(nm)，空间复杂度为O(n)

int \_Cut\_Dynamic\_DownToTop(int seg[], int pi[], int arr\_len, int length, int dump[])

{

int tmp;

dump[0] = 0;

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

{

tmp = -1;

for (int j = 0; j < arr\_len; ++j)

{

if (i - seg[j] >= 0)

tmp = max(tmp, pi[j] + dump[i - seg[j]]);

}

dump[i] = tmp;

}

return dump[length];

}

int Cut\_Dynamic\_DownToTop(int seg[], int pi[], int arr\_len, int length)

{

int \*dump = (int \*)malloc(sizeof(int)\*length + 1);

int tmp = \_Cut\_Dynamic\_DownToTop(seg, pi, arr\_len, length, dump);

free(dump);

return tmp;

}

int main()

{ int seg[]={1,2,3,4,5,6,7,8,9,10};

int pi[]={1,5,8,9,10,17,17,20,24,30};

int arr\_len=10;

int length=100;

\_Cut\_Dynamic\_DownToTop(seg, pi, arr\_len, length,0);

Cut\_Dynamic\_DownToTop(seg, pi,arr\_len,length);

return 0;

}

229页

1. ------对于背包问题的自底向上动态规划算法，请证明：

（1）它的时间效率属于O(nW)。

（2）它的空间效率属于O(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)。

答：01背包问题的动态规划解法递归方程为：

F(i, j) = max { F(i-1, j), F(i-1, j-wi) + vi }; j >= wi

F(i, j) = F(i-1, j) j < wi

伪代码

function BeiBao()

for i ⟵ 0 to n do

K[i][0] ⟵ 0

for j ⟵ 1 to W do

K[0][j] ⟵ 0

for i ⟵ 1 to n do

for j ⟵ 1 to W do

if j < wi then

K[i][j] ⟵ K[i-1][j]

else

K[i][j] ⟵ max(K[i-1][j], K[i-1][j-wi]+vi)

return K[n][W]

通过递归方程和伪代码计算F(i, j)值花费常量时间。

（1）（2）显然算法空间复杂度与时间复杂度均为Θ（nW）。其中W为背包容量

（3）为了确定一个最优子集的组成，该算法重复比较前一行中不超过两个单元格的值。因此，其时间效率类为O(n)。因为基本操作是比较“j < wi”。它被执行了n\*w次。

234页

1. ------矩阵连乘------

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

**a．**给出一个三个矩阵连乘的例子，当分别用（A1A2）A3和A1（A2A3）计算时，它们的乘法次数至少相差1000倍。

答：用基于定义的算法乘维的两个矩阵α-by-β和β-by-γ需要进行非典型的乘法。(该产品中有 α ,γ 元素，每个元素都需要进行重复乘法计算。)如果A1、A2和A3的维数分别为d0×d1、 d1×d2和d2×d3，则(A1·A2)·A3需要d0d1d2 + d0d2d3 = d0d2(d1 + d3)

乘法, 当 A1 ·(A2 ·A3)则需要 d1d2d3 + d0d1d3 = d1d3(d0 + d2)

乘法:下面是一个简单的特定值选择，例如，第一个值比第二个值大1000倍:

d0 = d2 = 103, d 1 = d3 =1 .

**b．**有多少种不同的方法来计算n个矩阵的连乘乘积。

答：设m(n)为计算n个矩阵A1·…·An的链积的不同方法的个数。链的任何括号化都将导致相乘，作为最后的操作，前k个矩阵(A1·…·Ak)和后n−k个矩阵(Ak+1·…·An)的乘积。做前者有m(k)种方法，做后者也有m(n−k)种方法。因此，对于n个矩阵的矩阵链插入方法的总数，我们有如下递归式:

由于将n个矩阵链插入括号进行乘法与构造n个节点的二叉树非常相似，因此上面的递归与前面的递归非常相似也就不足为奇了

提到的二叉树的数目。它们的解非常相似也不足为奇:即n≥1时，m(n)=b(n−1)，其中b(n)是有n个节点的二叉树的数目。让我们用数学归纳法来证明这个断言。基础立即检查:m(1) = b(0) = 1。对于一般情况，我们假设m(k)=b(k−1)对于不超过某个正整数n的所有正整数(我们使用数学归纳法的强版本);我们会证明这个等式对n+1也的确成立。

**c．**设计一个求n个矩阵乘法最优次数的动态规划。

答：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) = max {M (i ,k) + M [k + 1, j] + [di−1]\*dk\*dj}

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:

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 ]

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(“)”)

249页

1. ------谣言传播------

有N个人,每个人都有不同的谣言.通过发电子信息,他们想相互共享所有的谣言.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,保证每个人都能获得所有谣言的条件下,使发送的信息数最小.

Input用数字N(N<=100)来代表。

接下来N行，用来描述这N个人。

每一行给出这个人他认识几个人，然后给出所认识的人的编号及传消息给他所要花的时间.

最简便的方法则是Floyd算法，n^3是不会超时的，所以本题求出每两个点的最短路径，再去搜索最后得知消息的那个点，最后去作比较即可。

代码如下:

#include<bits/stdc++.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页9

（a）写一个程序,为给定的英文文本构造一套哈夫曼编码,并对该哈夫曼编码.

（b）写一个程序,对一段用哈夫曼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）对编码程序做一个实验,测试如果用标准的估计频率代替英文文本中的字符的实际出现频率,该程序的压缩率会出现什么样的变化.

#include <iostream>

#include <fstream>

#include <queue>

#include <string>

#include <algorithm>

using namespace std;

class Node

{

public:

char mchar;

int mweight;

Node \*lchild,\*rchild;

Node()

{

mchar='!'; //不是叶子结点就叹号表示

lchild=rchild=NULL;

}

Node(char c,int w)

{

mchar=c;

mweight=w;

lchild=rchild=NULL;

}

Node(Node &n)

{

mchar=n.mchar;

mweight=n.mweight;

lchild=n.lchild;

rchild=n.rchild;

}

};

bool operator <(Node a,Node b)

{

return a.mweight>b.mweight;

}

class HuffmanTree

{

public:

long mbitlenlastcoding;

int mSumBitLen;

Node \*root;

//queue头中自带有优先队列.如此定义可以小的先出队.注意要如此重载<运算符

priority\_queue<Node> H;

vector<Node\*> V;

vector<string> CodeTable;

HuffmanTree( )

{

mbitlenlastcoding=0;

mSumBitLen=0;

root=NULL;

}

void CreateTree(int \*farray,string chartable,int num)

{

SetQueueH(farray,chartable,num);

for(int k=0;k<num-1;k++)

{

Node \*i=new Node(H.top());

//auto\_ptr<Node> api(i);

H.pop();

Node \*j=new Node(H.top());

//auto\_ptr<Node> apj(j);

H.pop();

Node \*fk=new Node;

//auto\_ptr<Node> apfk(fk);

fk->lchild=i;

fk->rchild=j;

fk->mweight=i->mweight + j->mweight;

H.push(\*fk);

root=fk; //运算到第n-1次时.FK就是root

}

LinearizeTree(); //线性存储huffman树

string str(" ");

CreateCodeTable(root,str,-1); //创建编码表

sort(CodeTable.begin(),CodeTable.end()); //对编码表按字母序排序.方便压缩时查找

}

void Coding(const char\* str) //编码

{

ofstream fout;

fout.open("text.dat",ios::binary);

int bit=0;

int bitcount=0;

for(int i=0;i<strlen(str);i++)

{

int j=0;

if(str[i]>='A' && str[i]<='Z')

j=str[i]-'A'+1;

else if(str[i]>='a' && str[i]<='z')

j=str[i]-'a'+1;

else if(' '==str[i])

j=0;

string code(CodeTable[j].substr(2));

mbitlenlastcoding+=code.length(); //保存实际编码的位数,供解压缩使用;

for(int k=0;k<code.length();k++)

{

if('0'==code.at(k)) //按序把编码放到高位.也可以放到低位

bit=bit<<1;

else

bit=(bit<<1)+1; // 放低位,此步改为与0x80000000相或,bit=(bit|0x80000000);

bitcount=(bitcount+1)%32;

if(0==bitcount)

fout.write((char\*)&bit,sizeof(bit));

}

}

if(0!=bitcount) //将最后不满32位的位移动到最高位

{

bitcount=32-bitcount;

bit=bit<<bitcount;

fout.write((char\*)&bit,sizeof(bit));

}

fout.close();

}

void Decodeing(char \*infilename,char \*outfilename) //解码

{

ifstream fin;

fin.open(infilename,ios::binary);

if(true==fin.fail())

{

cerr<<"file read fail"<<endl;

return;

}

bool overflag=false;

long bitcount=0;

int buffsize=sizeof(int)\*8;

char buff[sizeof(int)\*8+1]; //+1用来保存串结束符

int bit;

Node \*p=root;

string strResult;

while(!overflag)

{

fin.read((char\*)&bit,sizeof(bit));

mbitlenlastcoding=mbitlenlastcoding-buffsize;

\_itoa(bit,buff,2);

//因为\_itoa函数转换二进制数时.若前几位为零时,不会转换位零,要自行添加

string str(buff);

str.insert(0,buffsize-strlen(buff),'0');

if(mbitlenlastcoding<0)

{

mbitlenlastcoding=buffsize+mbitlenlastcoding;

str=str.substr(0,mbitlenlastcoding);

buffsize=str.length();

overflag=true; //置结束标志

}

int i=0;

while(i<buffsize )

{

if('!' !=p->mchar)

{

strResult.push\_back(p->mchar);

p=root;

}

else

{

if('0'==str.at(i++))

p=p->lchild;

else

p=p->rchild;

}

}

}

cout<<strResult<<endl;

ofstream fout(outfilename);

fout<<strResult;

fout.close();

fin.close();

}

void ShowQueueH()

{

while(!H.empty())

{

Node t=H.top();

H.pop();

cout<<t.mchar<<" "<<t.mweight<<endl;

}

}

void ShowTree()

{

for(int i=0;i<V.size();i++)

cout<<i<<" "<<V[i]->mchar<<" "<<V[i]->mweight<<endl;

}

void ShowCodeTable()

{

for(int i=0;i<CodeTable.size();i++)

cout<<CodeTable.at(i)<<endl;

}

void PrintCodeTable(char \*filename)

{

ofstream fout(filename);

for(int i=0;i<CodeTable.size();i++)

fout<<CodeTable[i]<<endl;

fout.close();

}

~HuffmanTree()

{

for(int i=0;i<V.size();i++)

{

if(NULL!=V[i])

delete V[i];

}

}

private:

void SetQueueH(int \*farray,string chartable,int num)

{

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

{

Node t(chartable.at(i),farray[i]);

H.push(t);

}

}

void LinearizeTree() //线性存储huffman树

{

queue<Node> S;

S.push(\*root);

V.push\_back(root);

while(!S.empty())

{

Node t=S.front();

S.pop();

if(NULL != t.lchild)

{

S.push( (\*t.lchild) );

V.push\_back(t.lchild);

}

if(NULL != t.rchild)

{

S.push( (\*t.rchild) );

V.push\_back(t.rchild);

}

}

}

void CreateCodeTable(Node \*p,string str,int flag)

{

if (NULL == p ) return ;

if(0==flag) str.push\_back('0');//str.append("0");

else if(1==flag) str.push\_back('1');//str.append("1");

if('!' != p->mchar)

{

mSumBitLen+=(str.length()-1); //加上编码的长度

str.insert(str.begin(),p->mchar);

CodeTable.push\_back(str);

return;

}

else

{

if(NULL!=p->lchild)

CreateCodeTable(p->lchild,str,0);

if(NULL!=p->rchild)

CreateCodeTable(p->rchild,str,1);

}

}

};

void main()

{

string chartable(" etaoinshrdlcumwfgypbvkjxqz");

int farray[]={183,102,77,68,59,58,55,51,49,48,35,34,26,24,21,19,18,17,16,16,13,9,6,2,2,1,1};

char text[]="Chapter Graphs surveys the most important graph processing problems including depth first search breadth first search minimum spanning trees and shortest paths ";

HuffmanTree ht;

ht.CreateTree(farray,chartable,chartable.length());

ht.ShowTree();

ht.ShowCodeTable();

ht.PrintCodeTable("CodeTable.txt");

ht.Coding(text);

ht.Decodeing("text.dat","Decoding.txt");

cout<<"平均编码长度:"<<ht.mSumBitLen/27.0<<endl;

}

331页

1. --用回溯法生成{1,2,3,4}的所有排列------

#include <iostream> //--输入输出流头文件

using namespace std; //--配合头文件使用

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

int tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

}

void Backtrack(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;

Backtrack(arr, start+1, end);

swap(arr, i, start);

}

}

}

int main() //--主函数main()--

{

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

//cout << sizeof(arr) / sizeof(arr[0]) << " ";

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

}

338页

7. ------写一个程序用分支界限算法对背包问题求解------

#include<bits/stdc++.h>

using namespace std;

int n; //物品数量

int limitWeight; //背包上限重量

const int maxn = 1010;

int bestValue = 0; //最优值 背包能装的最大价值

//物品node 两个属性 价值 和 重量

struct node{

int v;

int w;

};

//定义排序规则

bool cmp(node a,node b){

double flag1 = a.v\*1.0/a.w;

double flag2 = b.v\*1.0/b.w;

return flag1 > flag2; //按价值与重量比排序

}

struct node nod[maxn];//存放物品的结构体数组

int takeNum[maxn]; //记录拿物品的数量

int bestTake[maxn]; //记录最优的 拿物品的数量

//第x个物品时 当前背包的重量 code by:fishers

void dfs(int x,int curWeight,int curValue){

if(x == n+1){ //递归出口 拿第x+1个物品了 就是到达了叶节点

if(curValue > bestValue && curWeight <= limitWeight){

bestValue = curValue; //更新最优值

for(int i=1;i<=n;i++) bestTake[i] = takeNum[i];

}

return; //结束递归

}

int nextMaxValue = curValue + (limitWeight - curWeight)\*(nod[x].v/nod[x].w); //分支界限: 以后最大也就是用背包剩下的重量全拿这个物品x(因为之前按重量价值比排序啦,小贪心)

if(nextMaxValue < bestValue) return; //和最优值(已经搜索出的背包最大价值)比较(剪枝优化),小于最优值就直接返回了

int curMaxNum = (limitWeight - curWeight)/nod[x].w; //当前一层最多拿物品x的个数 (以确保不超过最大背包重量上限limitWeight)

for(int i=curMaxNum;i>=0;i--){ //枚举每一个分支 i表示拿这个物品的个数:拿i个

int nextWeight = curWeight + i \* nod[x].w; //如果这一次拿了i个物品 算出nextWeight重量 上一轮重量+这一轮i个物品x的重量

if(nextWeight > limitWeight) continue;

int nextActualValue = curValue + i \* nod[x].v; //拿了i个物品后实际的价值

takeNum[x] = i; //记录第x个物品 拿的个数i

dfs(x+1,nextWeight,nextActualValue); //递归搜索下一个 物品的情况

takeNum[x] = 0;

}

}

void input(){

cin>>n>>limitWeight; //输入物品个数n,背包承受的重量上限

for(int i=1;i<=n;i++) cin>>nod[i].v; //输入每个物品的价值

for(int i=1;i<=n;i++) cin>>nod[i].w; //输入每个物品的重量

sort(nod+1,nod+n+1,cmp); //把物品按 价值重量比 排序 (贪心,也是后面分支界限来剪枝的前提)

}

int main(){

input();

dfs(1,0,0);

cout<<"背包最多装的价值是"<<bestValue<<endl;

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

cout<<"拿了"<<bestTake[i]<<"个"<<"重量为"<<nod[i].w<<",价值为"<<nod[i].v<<"的物品."<<endl;

}

return 0;

}