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：忽略常数、误差的平均情况中快速排序大约执行107次，插入排序执行1012次，快了10万倍。

2.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：存在，当原数据已经顺序正确的前提下，插入排序比快速排序快。

3.切割木棍 为下列问题设计一个动态规划的算法，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的木棍切割为若干根长度为整数的小木棍，使得所能获得的总销售价格最大？该算法的时间效率和空间效率各是多少？

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;

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。

动态规划自底向上：

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;

}

时间复杂度为O(nm)

空间复杂度为O(n)

4.对于背包问题的自底向上动态规划算法，请证明：

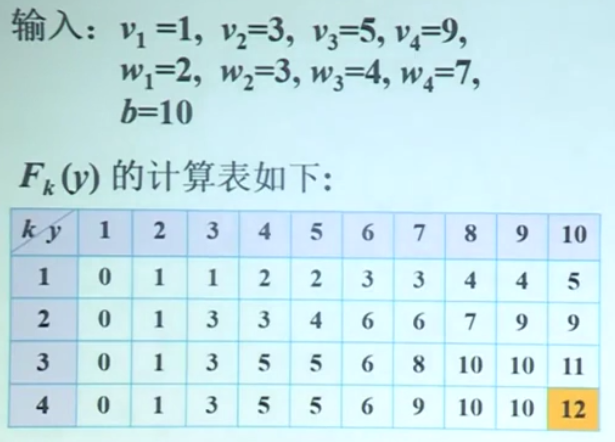
a．他的时间效率属于0（nW）。

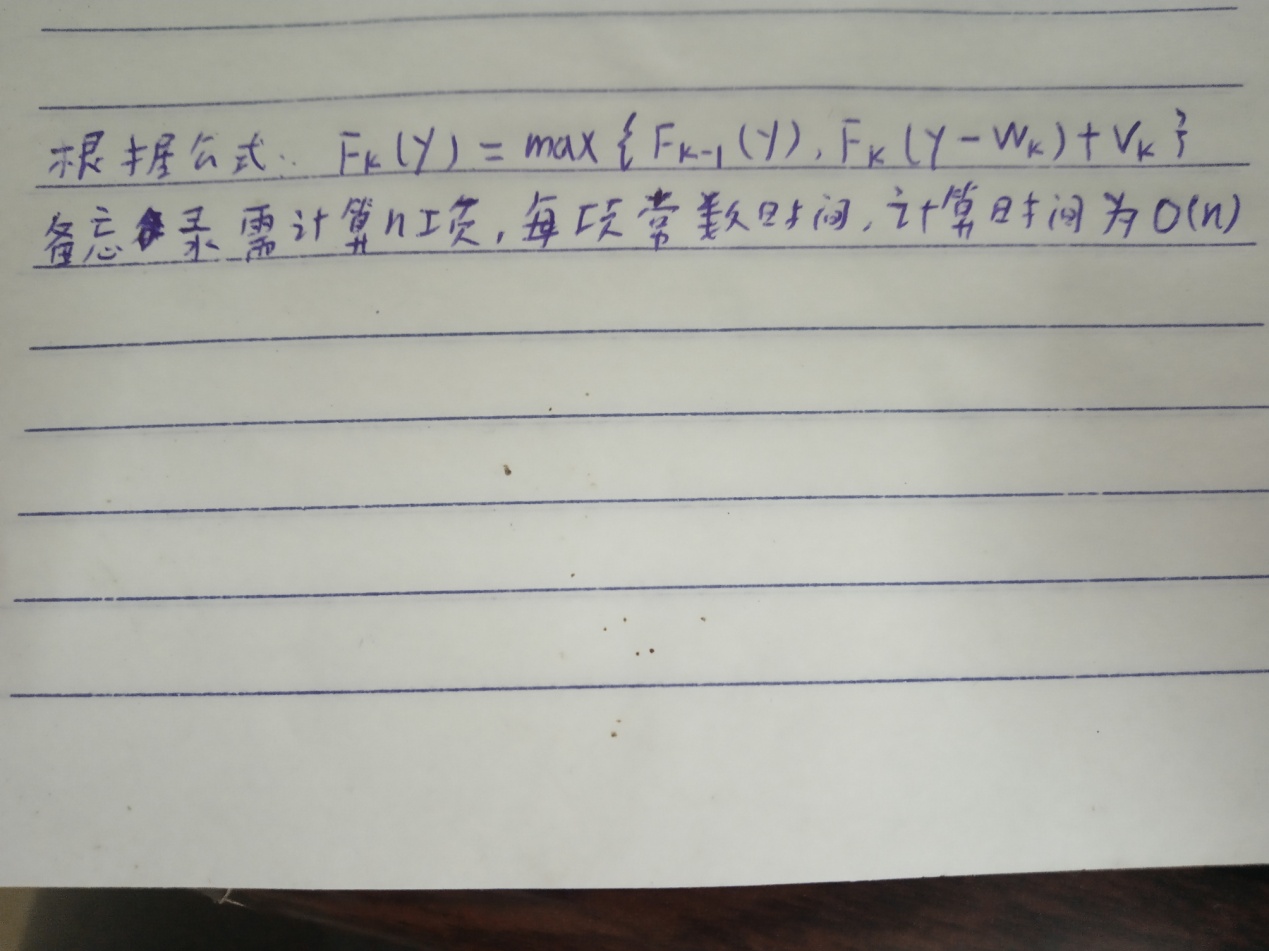
答：此算法是n步叠代计算局部最优解，每一步叠代需要计算m个子项，时间复杂度就是O(m\*n)，属于0（nW）。

b．它的空间效率属于0（nW）。

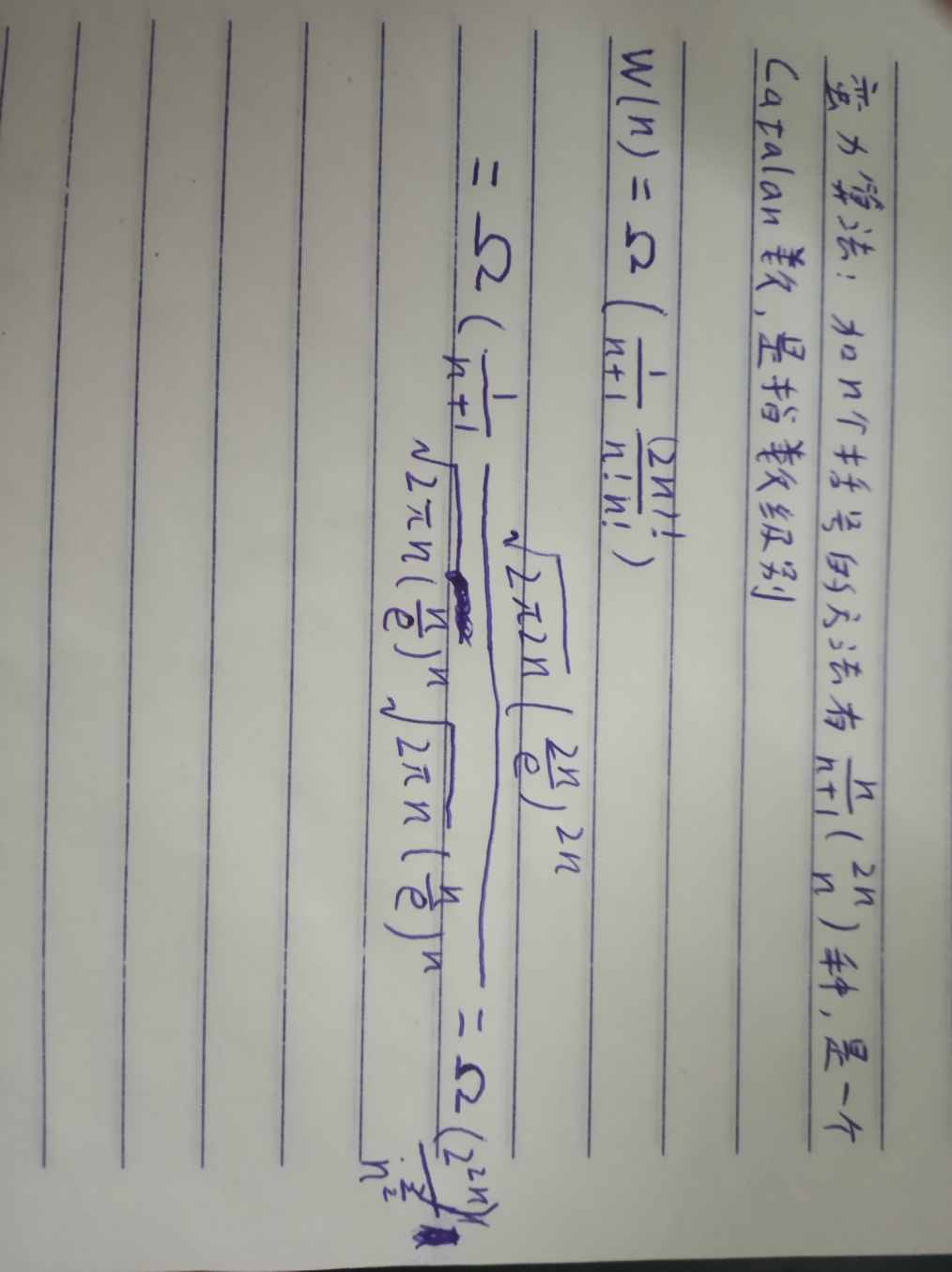
答：如果只保存一步叠代的结果，空间复杂度就是O(m)；如果需要保存k步叠代结果，空间复杂度就是O(m\*k)，属于0（nW）。

c．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0（n）。





5.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d0Xd1，d1Xd2，- - -，dn-1Xdn。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。



6.谣言传播 又n个人，每个人有不同的谣言，通过发电子信息，他们想相互共享谣言。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，保证每个人都能获取所有谣言的条件下，使发送的信息数最小。

7.a．写一个程序，为给定的英文文本构造一套哈夫曼编码，并对该文本编码。

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

char c;

int count;

}nd;

int cmp(const void\* p1, const void\*p2)

{

nd\*c = (nd\*)p1;

nd\*d = (nd\*)p2;

if(c->count != d->count) return d->count - c->count;

else return c->c - d->c;

}

int main()

{

nd a[26];

int i;

for(i=0;i<26;i++)

{

a[i].c='a'+i;

a[i].count=0;

}

char s[128];

while(scanf("%s",s)!=EOF)

{

for(i=0;s[i]!=0;i++)

{

a[s[i]-'a'].count++;

}

qsort(a,26,sizeof(a[0]),cmp);

for(i=0;a[i].count;i++)

{

printf("%c %d\n",a[i].c,a[i].count);

}

}

return 0;

}

b．写一个程序，对一段哈夫曼编码的英文文本进行解码。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define n 8

#define m 2\*n-1

#define max 2000

typedef struct

{

int wi;

char data;

int Parent,Lchild,Rchild;

}huffm;

huffm HT[m+1];

typedef struct

{

char bits[n+1];

int start;

char ch;

}ctype;

void HuffmTree(huffm HT[m+1]);

void Huffmcode(ctype code[n+1]);

void Output (ctype code[n+1]);

/\* 构造HuffmTree的函bai数du\*/

void HuffmTree(huffm \*HT)

{

int i,j,p1,p2;

int s1,s2;

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

// {

// scanf("%d",&w);

// HT[i].wi = w;

// }

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

{

p1=p2=0;

s1=s2=max;

for(j=1;j<=i-1;j++)

if(HT[j].Parent ==0)

if(HT[j].wi <s1)

{

s2=s1;

s1=HT[j].wi;

p2=p1; p1=j;

}

else if(HT[j].wi<s2)

{

s2=HT[j].wi ;

p2=j;

}

HT[p1].Parent = HT[p2].Parent =i;

HT[i].Lchild =p1;

HT[i].Rchild =p2;

HT[i].wi =HT[p1].wi + HT[p2].wi ;

}

// printf("\n OK!");

// for(i=1;i<=m;i++)

// {

// printf("\n");

// printf("%d ",HT[i].wi);

// }

// getchar();

return ;

}

/\* 求HuffmTree编码的函数\*/

void Huffmcode(ctype code[n+1])

{

int i,p,s;

ctype md;

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

{

md.ch = code[i].ch;

md.start = n+1;

s = i;

p = HT[i].Parent;

while(p!=0)

{

md.start--;

if(HT[p].Lchild ==s)

md.bits[md.start]='0';

else

md.bits[md.start] ='1';

s=p;

p=HT[p].Parent;

}

code[i] = md;

}

}

/\*打印编码函数\*/

void Output(ctype code[n+1])

{

int i,j;

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

{

printf("\n");

printf("%c ",code[i].ch );

for(j=1;j<=8;j++)

{

if(j<code[i].start)

printf(" ");

else

if((code[i].bits[j] == '0')||zhi(code[i].bits[j] == '1'))

printf("%c",code[i].bits[j]);

}

printf(" %d",code[i].start);

}

}

void main()

{

int i,j;

int w;

int flag=1;

int choice;

ctype code[n+1];

char temp[n+1];

int temp2[n+1];

printf("Would you want to play?(1-Yes and Start/0-No and Exit)");

scanf("%d",&choice);

while(flag&&(choice==1))

{

choice = 0;

for(i=1;i<=m;i++)//初始化

{

HT[i].data =NULL;

HT[i].wi=0;

HT[i].Parent = 0;

HT[i].Lchild = HT[i].Rchild = 0;

}

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

{

code[i].start = 0;

code[i].ch = NULL;

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

code[i].bits[j] = NULL;

}

printf("Please input %d char: \n",n);

getchar();

scanf("%c %c %c %c %c %c %c %c",&temp[1],&temp[2],&temp[3],&temp[4],&temp[5],&temp[6],&temp[7],&temp[8]);

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

{

// scanf("%c",&w);

code[i].ch =temp[i];

HT[i].data =temp[i];

}

printf("Please input %d rate: \n",n);

getchar();

scanf("%d %d %d %d %d %d %d %d",&temp2[1],&temp2[2],&temp2[3],&temp2[4],&temp2[5],&temp2[6],&temp2[7],&temp2[8]);

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

{

//scanf("%d",&w);

w= temp2[i];

HT[i].wi = w;

}

HuffmTree(HT);

Huffmcode(code);

Output(code);

printf("\nContinue?1-Contine,0-Exit\n");

scanf("%d",&choice);

if(choice!=1)

break;

}

getchar();

return 0;

}

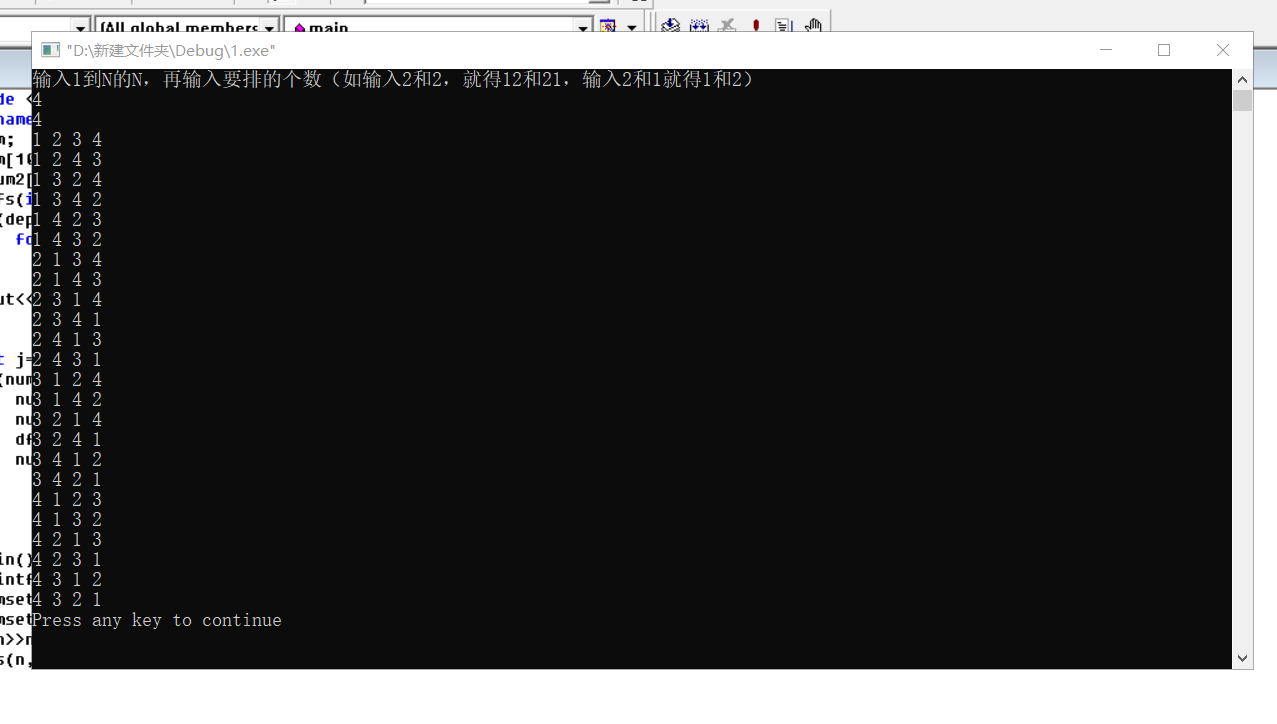
c．做一个实验，测试对包含1000个词的一段英文文本进行哈夫曼编码时，典型的压缩率位于什么区间。

答：其平均码长是等长码的87%。 所以平均压缩率为13%。

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字符的实际出现频率，该程序的压缩率会有什么样的变化。

8.用回溯生成{1，2，3，4}的所有排列。

#include <iostream>  
using namespace std;  
int n,m;  
int num[10];  
bool num2[10];  
void dfs(int n,int depth){  
if(depth==m+1){  
for(int i=1;i<=m;i++){  
cout<<num[i]<<&apos; &apos;;  
}  
cout<<endl;  
}  
else{  
for(int j=1;j<=n;j++){  
if(num2[j]){  
num[depth]=j;  
num2[j]=false;  
dfs(n,depth+1);  
num2[j]=true;  
}  
}  
}  
}  
int main(){  
printf("输入1到N的N，再输入要排的个数（如输入2和2，就得12和21，输入2和1就得1和2）\n");  
memset(num,0,sizeof(num));  
memset(num2,true,sizeof(num2));  
cin>>n>>m;  
dfs(n,1);  
}



9.写一个程序用分支界限算法对背包问题求解。

