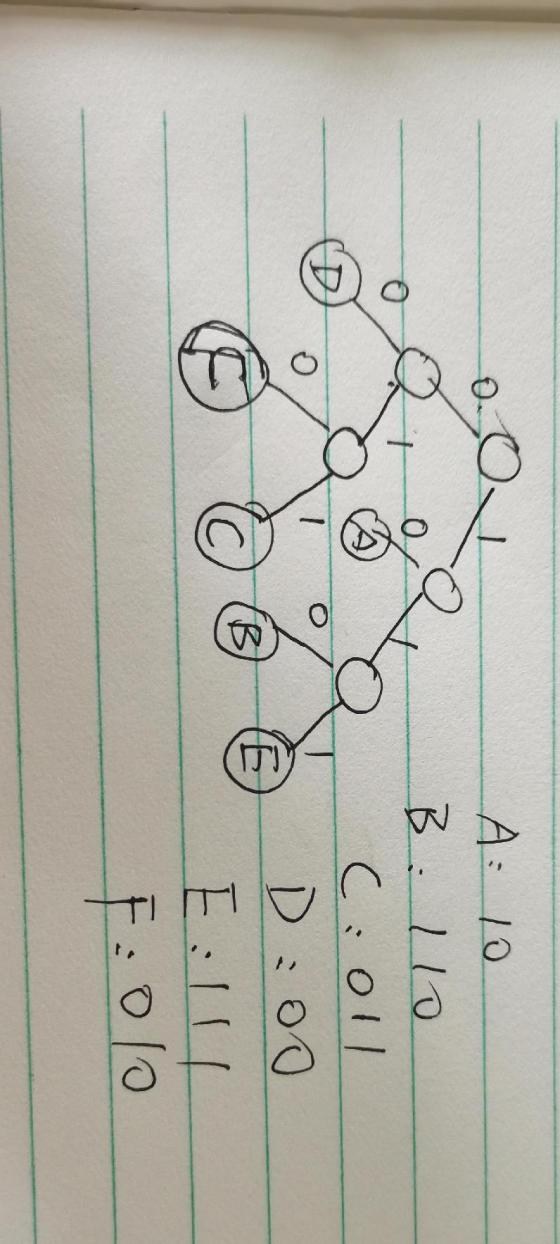
大作业二

哈夫曼树的应用

1. 各个字符出现的概率作为权

A:25 B:15 C:13 D:20 E:16 F:8

2.根据设计的哈夫曼树得到的哈夫曼编码

3.代码：#include<stdio.h>

#include<string.h>

#include<malloc.h>

typedef struct hnode

{ int weight;

int lchild,rchild,parent;

}HTNode,\*HuffmanTree;

typedef char \*\*HuffmanCode;

void Select(HTNode HT[],int len,int &s1,int &s2)

{

int i,min1=32767,min2=32767;

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

{

if(HT[i].weight<min1&&HT[i].parent==0)

{

s2=s1;

min2=min1;

min1=HT[i].weight;

s1=i;

}

else if(HT[i].weight<min2&&HT[i].parent==0)

{ min2=HT[i].weight;

s2=i;

}

}

}

void CreateHuffman\_tree(HuffmanTree &HT,int n)

{

if(n<=1) return;

int m,i,s1,s2;

m=2\*n-1;

HT=new HTNode[m+1];

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

{

HT[i].parent=0;

HT[i].lchild=0;

HT[i].rchild=0;

}

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

{

scanf("%d",&HT[i].weight);

}

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

{

Select(HT,i-1,s1,s2);

HT[s1].parent=i;

HT[s2].parent=i;

HT[i].lchild=s1;

HT[i].rchild=s2;

HT[i].weight=HT[s1].weight+HT[s2].weight;

}

}

void Huffman\_code(HuffmanTree HT,HuffmanCode &HC,int n)

{

int i,f,start,c;

char \*cd;

HC=new char\*[n+1];

cd=new char[n];

cd[n-1] ='\0';

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

{

start=n-1;

c=i;f=HT[i].parent;

while(f!=0)

{

start--;

if(HT[f].lchild==c)

cd[start]='0';

else

cd[start]='1';

c=f;f=HT[f].parent;

}

HC[i]=new char[n-start];

strcpy(HC[i],&cd[start]);

}

delete cd;

}

int main()

{

HuffmanTree HT;

HuffmanCode HC;

int i, n;

scanf("%d",&n);

CreateHuffman\_tree(HT, n);

Huffman\_code(HT,HC,n);

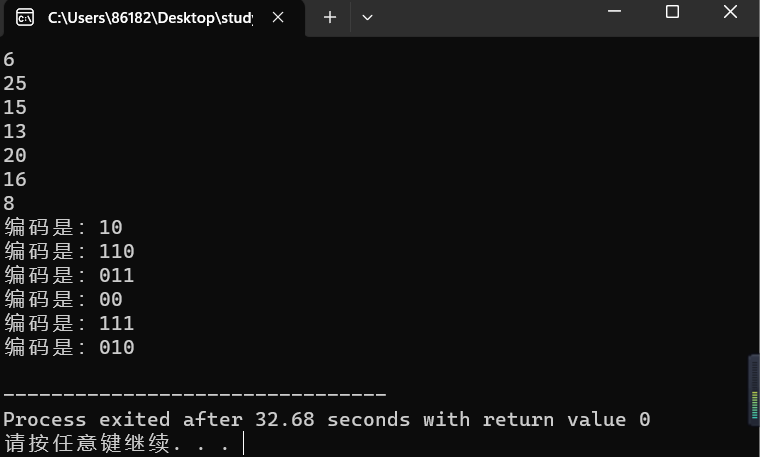
for(i=1;i<=n;i++)

printf("编码是：%s\n",HC[i]);

return 0;

}

输入与运行结果



1. 算法效率分析

时间复杂度分析：

建立哈夫曼树的时间复杂度为O(n^2)

哈夫曼树编码的时间复杂度为O(nlogn)

该算法的总时间复杂度为O(n^2+nlogn)，即O(n^2)  
空间复杂度分析：  
 建立哈夫曼树需要申请2n-1个结点的空间，空间复杂度为O(n)。  
 哈夫曼树编码需要申请n个HuffmanCode类型的指针，每个指针指向一个编码结果，每个编码结果的长度为字符集大小n，另外还需要申请一个char类型的数组cd，用于存储临时编码结果，其长度也为n，空间复杂度为O(n^2)。

所以总的空间复杂度为O（n^2+n),即O(n^2）。