第六题

6.、有一个n个字符构成的字母表，它的哈夫曼编码可能具有的最大码长是多少？（贪心算法）

#include<iostream>

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include<algorithm>

using namespace std;

typedef struct Node

{

char typeChar;

int rate;

Node \*leftChild;

Node \*rightChild;

}charNode;

char \*inputStr;

Node nodes[200 + 199];

int total;

int depth;

int getNum(char \*in, char c)

{

int times = 0;

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

{

if (\*(in + i) == c) times++;

}

return times;

}

int getTypeChar(char \*in)

{

inputStr = (char \*)malloc(sizeof(char)\*200);

memset(inputStr, '\0', sizeof(inputStr));

int p = 0;

bool isAdded;

//cout<<strlen(in)<<endl;

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

{

isAdded = false;

if (p == 0)

{

\*inputStr = \*in;

nodes[p].typeChar = \*inputStr;

nodes[p].rate = getNum(in, \*in);

nodes[p].leftChild = NULL; //叶节点

nodes[p].rightChild = NULL; //叶节点

p++;

continue;

}

for (int j = 0; j<p; j++)

{

if (\*(inputStr + j) == \*(in + i))

{

isAdded = true;

break;

}

}

if (!isAdded)

{

inputStr[p] = \*(in + i);

nodes[p].typeChar = \*(inputStr + p);

nodes[p].rate = getNum(in, \*(in + i));

nodes[p].leftChild = NULL; //叶节点

nodes[p].rightChild = NULL; //叶节点

p++;

}

}

return p;

}

//给排序函数使用的比较函数

bool compNode(const Node & node1, const Node & node2)

{

if (node1.rate <= node2.rate)

{

return true;

}

else

{

return false;

}

}

//哈夫曼编码

void huf(int startIndex, int endIndex, int length)

{

sort(nodes + startIndex, nodes + startIndex + length, compNode);

nodes[endIndex + 1].rate =

nodes[startIndex].rate + nodes[startIndex+1].rate;

nodes[endIndex + 1].leftChild = &nodes[startIndex];

nodes[endIndex + 1].rightChild = &nodes[startIndex+1];

if (endIndex + 1 != total)

{

huf(startIndex + 2, endIndex + 1, length - 1);

}

}

int depthFirst(charNode \*);

int getCodeLength(int lastIndex)

{

charNode \*root = &nodes[lastIndex];

depth = -1;

return depthFirst(root);

}

int depthFirst(charNode \*node)

{

depth++;

if (node->leftChild == NULL && node->rightChild == NULL)

{

return node->rate \* depth;

}

else

{

int leftSize = depthFirst(node->leftChild);

depth--;

int rightSize = depthFirst(node->rightChild);

depth--;

return leftSize + rightSize;

}

}

int main()

{

char \*in;

int num; //不同字符的数量

in = (char \*)malloc(sizeof(char) \* 1000);

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

{

num = getTypeChar(in);

if(num == 1)

{

cout<<strlen(in)<<endl;

continue;

}

total = num \* 2 - 1;

huf(0, num - 1, num);

cout<<getCodeLength(total-1)<<endl;

}

return 0;

}

第二题

2.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？

（类似于找零问题）

此算法的时间效率是O(n^2)，空间效率是O (n)

#include <iostream>

using namespace std;

int main()

{

int n = 10; //木棍长度为10

int price[11] = { 0, 1, 7, 8, 9, 10, 17, 17, 20, 23, 24 }; //每一段的价钱

int \*profit = new int[n + 1];

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

profit[i] = 0; //初始化最大利益

for (i = 1; i <= n; i++)//规模长度为i

{

int q = INT\_MIN;

for (int j = 1; j <= i; j++)//计算规模为i的最大收益

{

if (q < (price[j] + profit[i - j]))//因为i>i-j，所以当计算r[i]时，r[i-j]已经解决，可以直接用

q = (price[j] + profit[i - j]); //迭代q；

}

profit[i] = q; //找出i这个位置的最优解；

}

cout << profit[n] <<endl; //最后是n这个位置，就是n米长的木头的最大价值。

delete profit;

return 0;

}

第七题

7.用回溯法生成{1,2,3,4}的所有排列。（回溯法）

利用回溯法求解问题包含三个步骤：

1. 确定问题解空间：为了实现回溯，首先需要为问题定义一个解空间，该空间必须至少包含问题的一个解。
2. 按照易于搜索的策略组织解空间结构：当定义好问题的解空间后，应该将空间很好地组织起来，使得能用回溯法方便地搜索整个解空间，通常将解空间组织成树或图的形式。
3. 以深度优先方式搜索解空间。

#include<iostream.h>

#include<stdio.h>

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

const int N=4;

void print(){

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

cout<<a[i]<<" ";

cout<<endl;

}

void swap(int \*a,int i,int j)

{

int temp;

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

void backtrack(int i)

{

if(i>=N)

{

print();

}

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

{

swap(a,i,j);

backtrack(i+1);

swap(a,i,j);

}

}

int main()

{

cout << "所有组合：" <<endl;

backtrack(0);

return 0;

}

第八题

8.写一个程序分支界限算法对背包问题求解。（分支界限）

利用分支界限求解问题包含三个步骤：

1. 确定问题解空间：为了实现分支界限，首先需要为问题定义一个解空间，该空间至少要包含问题的一个解，可能是最优的。
2. 组织解空间结构：当定义好问题的解空间后，将空间很好地组织起来，使得能用分支界限方便地搜索整个解空间，通常将解空间组织成有序的树结构。
3. 搜索解空间：通常采用广度优先搜索法来搜索这棵有序树。

#include <stdio.h>

#include <stdlib.h>

typedef struct QNode{

int value;

int weight;

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct{

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("input the number of things and the volume of ships:");

scanf("%d%d",&n,&c)；

w=new int[n];

v=new int[n];

printf("input the weights:");

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

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

printf("input the values:");

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

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

initQueue(loadingQueue);

enQueue(loadingQueue,-1,0);

i=0;

ew=0;

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)

{

if(emptyQueue(loadingQueue))

{

printf("the result is %d.\n",bestvalue);

}

enQueue(loadingQueue,-1,0);

deQueue(loadingQueue, ev,ew);

i++;

}

}

return 0;

}

第一题

1.a.对于一个包含100万随机数的数组排列，快速排列比插入排列快多少倍？

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：

直接插入排序是一种简单的排序方式，其核心思想就是将一个记录插入到已经排好顺序的有序表中，从而得到一个新的有序表。快速排序采用一种分治的策略，基本思想是将原问题分解为若干个规模更小但结构与原问题相似的子问题，递归的解这些子问题的解组合为原问题的解，用“分解-求解-组合”解决问题。

a.对于一个包含100万随机数的数组排列，快速排列比插入排列快大约10倍。

b.插入排序的[时间复杂度](http://baike.baidu.com/view/104946.htm)为O(n^2)，快速排序的[时间复杂度](http://baike.baidu.com/view/104946.htm)为O(nlog₂n)。插入排序的效率比快速排序的效率要低，但还要考虑最好情况和最坏情况的输入，插入排序最坏的情况下时间复杂地为O(n^2)，最好的情况下为O(n)),快速排序最坏情况下的时间复杂度为O(n^2)，最好的情况为O(nlog₂n),所以说不会存在插入排序比快速排序更快的情形。

//快速排序

function quickSort(arr){

if(arr.length <= 1){

return arr;

}

var left = [];

var right = [];

var midIndex = parseInt(arr.length / 2);

var mid = arr[midIndex];

for(var i = 0 ; i < arr.length ; i++){

if(i == midIndex) continue;

if( arr[i] < mid){

left.push(arr[i])

}else{

right.push(arr[i]);

}

}

return quickSort(left).concat([mid],quickSort(right));

}

[1,2].concat([3],[4,5]);? //[1, 2, 3, 4, 5]

//插入排序

var arr=[45,1,32,21,56,87,43,12,34,45];

for(var i=0;i<arr.length;i++){

var n=i;

while(arr[n]>arr[n+1] && n>=0)

{

var temp=arr[n];

arr[n]=arr[n+1];

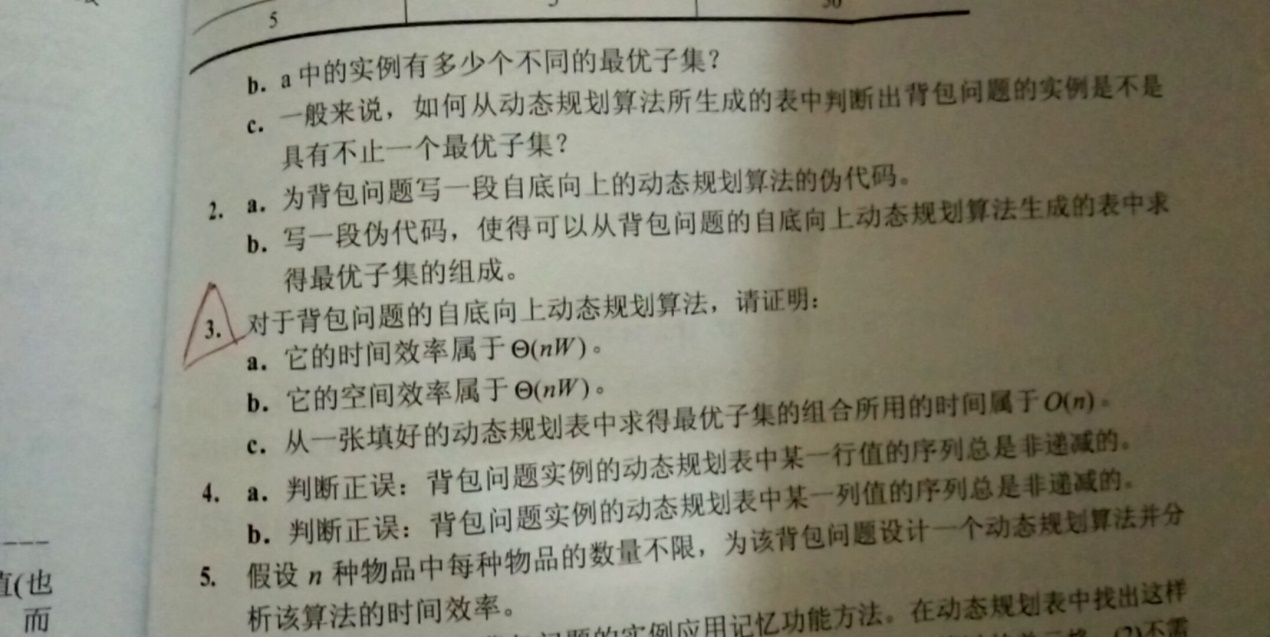
arr[n+1]=temp;

n--;

}

}

第三题



//对背包问题的自底向上的动态规划算法，证明:....

//给定N个物品和一个背包。物品i的重量是Wi,其价值位Vi，背包的容量为C。问应该如何选择装入背包的物品，使得转入背包的物品的总价值为最大？？

//在选择物品的时候，对每种物品i只有两种选择，即装入背包或不装入背包。

//令V(i,j)表示在前i(1<=i<=n)个物品中能够装入容量为就j(1<=j<=C)的背包中的物品的最大价值

//V(i,0)=V(0,j)=0

//V(i,j)=V(i-1,j) j<wi

//V(i,j)=max{V(i-1,j) ,V(i-1,j-wi)+vi) } j>wi

#include<stdio.h>

int V[200][200];//前i个物品装入容量为j的背包中获得的最大价值

int max(int a,int b)

{

if(a>=b)

return a;

else return b;

}

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

{

int i,j;

//填表,其中第一行和第一列全为0

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

V[i][0]=0;

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

V[0][j]=0;

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

{

printf("%d %d %d ",i,w[i-1],v[i-1]);

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

{

if(j<w[i-1])

{

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

printf("[%d][%d]=%2d ",i,j,V[i][j]);

}

else

{

V[i][j]=max(V[i-1][j],V[i-1][j-w[i-1]]+v[i-1]);

printf("[%d][%d]=%2d ",i,j,V[i][j]);

}

}

printf("\n");

}

//判断哪些物品被选中

j=C;

for(i=n;i>=1;i--)

{

if(V[i][j]>V[i-1][j])

{

x[i]=1;

j=j-w[i-1];

}

else

x[i]=0;

}

printf("选中的物品是:\n");

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

printf("%d ",x[i]);

printf("\n");

return V[n][C];

}

void main()

{

int s;//获得的最大价值

int w[15];//物品的重量

int v[15];//物品的价值

int x[15];//物品的选取状态

int n,i;

int C;//背包最大容量

n=5;

printf("请输入背包的最大容量:\n");

scanf("%d",&C);

printf("输入物品数:\n");

scanf("%d",&n);

printf("请分别输入物品的重量:\n");

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

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

printf("请分别输入物品的价值:\n");

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

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

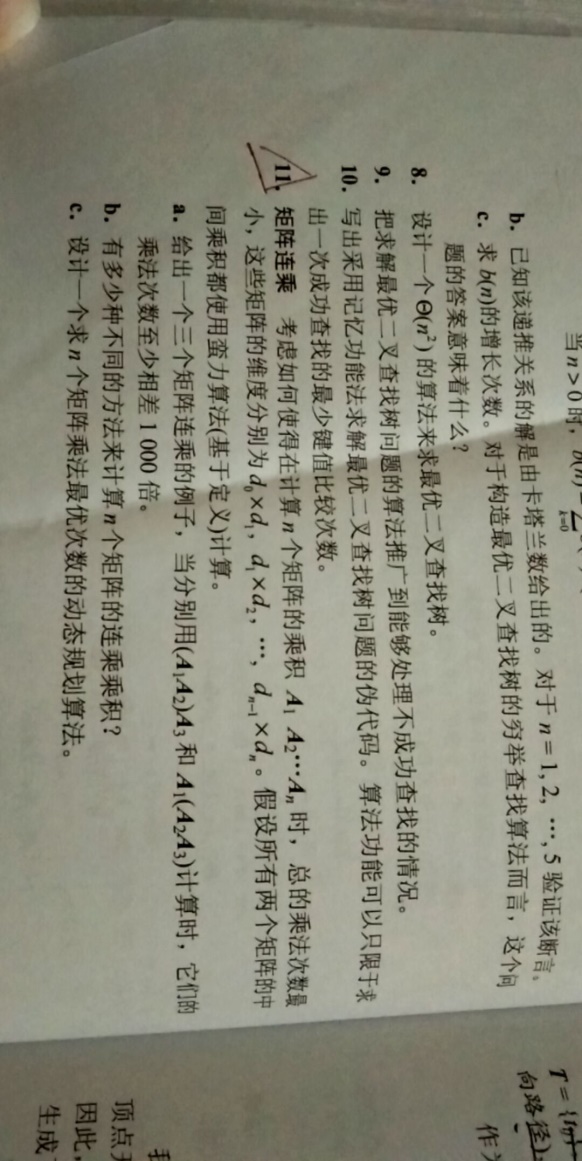
s=KnapSack(n,w,v,x,C);

printf("最大物品价值为:\n");

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

}

第四题



#include<stdio.h>

int r[100],com[100][100];

long int course(int i,int j)

{

long int u,t;

int k;

if(i==j)

return 0;

if(i==j-1)

{

com[i][i+1]=i;

return(r[i]\*r[i+1]\*r[i+2]);

}

u=course(i,i)+course(i+1,j)+(long)r[i]\*r[i+1]\*r[j+1];

com[i][j]=i;

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

{

t=course(i,k)+course(k+1,j)+(long)r[i]\*r[k+1]\*r[j+1];

if(t<u)

{

u=t;

com[i][j]=k;

}

}

return u;

}

void main()

{

int n,i,j;

printf("请输入矩阵个数\n");

scanf("%d",&n);

printf("请输入矩阵型号\n");

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

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

printf("最小的乘法次数为%ld\n",course(1,n));

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

{

printf("\n");

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

printf("%5d",com[i][j]);

}

getchar();

}

第五题

5. 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她已知的所有谣言，而且一条信息只有一个收信人设计。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。

**解答：**

**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。**

**对于n=4，信息数量最少是6**

#include<stdio.h>

int pre[20105];

int Find(int x)

{

int i=x;

int r=x;

int j;

while(pre[i]!=i)

i=pre[i];

while(r!=i)

{

j=pre[r];

pre[r]=i;

r=j;

}

return i;

}

void mix(int x,int y)

{

int fx=Find(x);

int fy=Find(y);

if (fx!=fy)

{

pre[fx]=fy;

}

}

int main()

{

int T,t,n,m,i,a,b,temp,ok;

scanf("%d",&T);

while(T--)

{ ok=1;

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

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

pre[i]=i;

scanf("%d",&m);

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

{

scanf("%d%d",&a,&b);

mix(a,b);

}

temp=Find(t);

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

if (Find(i)!=temp)

{

ok=0;

break;

}

if (ok)

printf("Yes\n");

else

printf("No\n");

}

return 0;

}