1. 填空题
2. 时间 空间
3. 
4. 输出 有限性
5. O()
6. 分治
7. 动态规划
8. （1）{i|<} （2）非减序 非增序
9. 局部
10. 贪心算法效率最高
11. pos-from+1
12. 简答题

1.确定性、可实现性、输入、输出、有穷性

2，分析算法占用计算机资源的情况，对算法做出比较和评价，设计出额更好的算法。 3.算法的时间复杂性与问题的规模相关，是问题大小n的函数。

4，当问题的规模n趋向无穷大时，影响算法效率的重要因素是T（n）的数量级，而其他 因素仅是使时间复杂度相差常数倍，因此可以用T（n）的数量级（阶）评价算法。时间复杂 度T（n）的数量级（阶）称为渐进时间复杂性。

5.最坏情况下的时问复杂性和平均时问复杂性考察的是n固定时，不同输入实例下的 算法所耗时间。最坏情况下的时间复杂性取的输入实例中最大的时间复杂度：

W(n) = max{ T(n, I) }, I ∈ Dn

平均时间复杂性是所有输入实例的处理时间与各自概率的乘积和：

A (n) = ΣP(I) T(n, I) , I ∈ Dn

1. 程序填空题

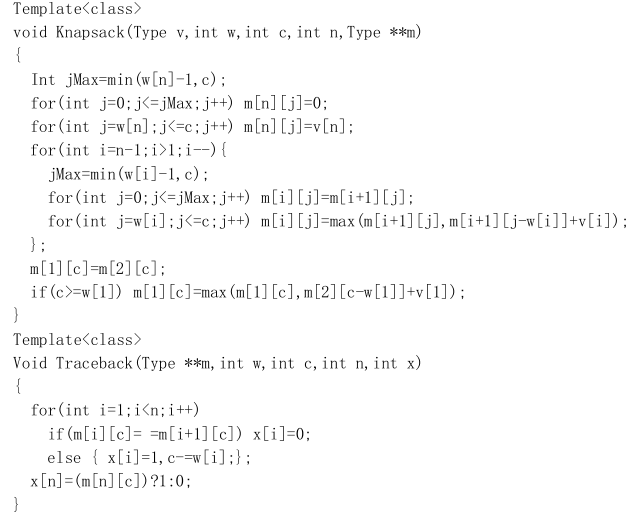
1.

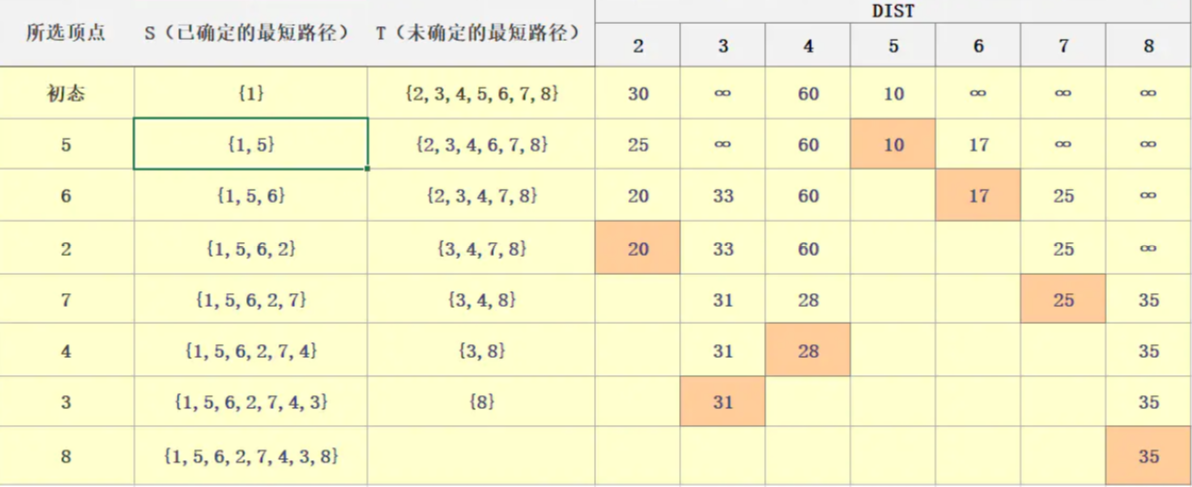
1. p[i]/w[i]
2. mm - w[i]
3. pp + p[i]
4. pp + x[i] \* p[i]

2.

1. thissum +=a [k]
2. besti = i
3. 算法设计题

1.





#include <iostream>

using namespace std;

int main(){

int m,n;//m是有多少个顶点，n是从哪个顶点开

cin>>m>>n;

int p[m][m];

    for(int i=0;i<m;i++){

        for(int j=0;j<m;j++){

            cin>>p[i][j];

        }

    }

    int d[m];//用来存储n顶点到其他顶点的最短路径，输出从1到m-1的数，默认是0

    for(int i=0;i<m;i++){

        d[i]=0;

    }

    //要想得到其他最短路径，需要循环m-1次，每次确定一个最短路径

    int k=m-1;

    int v[m];//看是否已经确定是最短路径了，初始化为0如果是1标识确定了

    for(int i=0;i<m;i++){

        v[i]=0;

    }

    int p1=n;

    int min;

    int min\_i;

    bool b=true;

    for(int i=0;i<m;i++){

        if(i==n){

            continue;

        }

        if(p[n][i]!=0&&b){//找到第一个非0的元素标识为最小的

            min=p[n][i];

            min\_i=i;

            b=false;

            d[i]=p[n][i];

        }

        if(!b&&p[n][i]!=0){

            if(p[n][i]<min){

                min=p[n][i];

                min\_i=i;

            }

            d[i]=p[n][i];

        }

    }

    v[min\_i]=1;//标识确定了最短路径的下标

    n=min\_i;//有最小的从最小的那个开始，和上一次的d[i]对比，看是否有比他小的

    k=k-1;

    for(int i=0;i<m;i++){

        cout<<d[i]<<" ";

    }

    cout<<n<<endl;

    while(k--){//进行m-1次遍历，确定最小

        for(int i=0;i<m;i++){

            if(v[i]==1||i==p1){

                continue;

            }

            if(p[n][i]!=0&&d[i]!=0){

                if(p[n][i]+d[min\_i]<d[i]){

                    d[i]=p[n][i]+d[min\_i];

                }

            }

            if(p[n][i]!=0&&d[i]==0){

                d[i]=p[n][i]+d[min\_i];

            }

        }

        b=true;

        //确定d[i]不为0的最小下标并且v[i]==0的下标

        for(int i=0;i<m;i++){

            if(d[i]!=0&&b&&v[i]==0){//找到第一个非0的元素标识为最小的

                min=d[i];

                min\_i=i;

                b=false;

            }

            if(!b&&d[i]!=0&&v[i]==0){

                if(d[i]<min){

                    min=d[i];

                    min\_i=i;

                }

            }

        }

        v[min\_i]=1;

        n=min\_i;

        for(int i=0;i<m;i++){

            cout<<d[i]<<" ";

        }

        cout<<n<<endl;

    }

    for(int i=0;i<m;i++){

        if(i!=p1){

            if(d[i]==0){

                cout<<-1<<" ";

            }else{

                cout<<d[i]<<" ";

            }

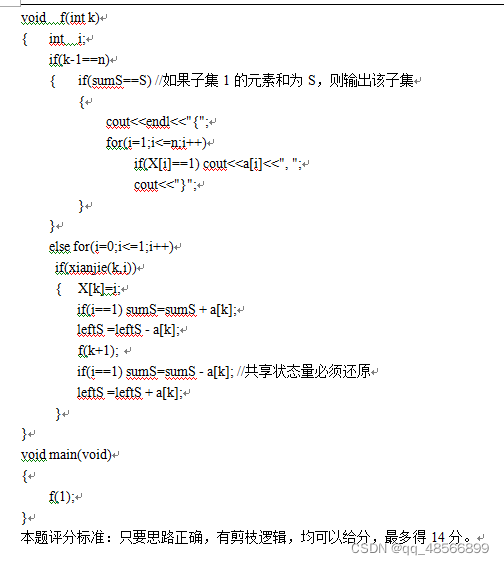
        }

    }

    return 0;

}

3.



3. #include <iostream>(供参考)

#include <bits/stdc++.h>

using namespace std;

int re[200][200];//记录路径数组1

int re1[200][200];//记录路径数组2

char s1[200],s2[200];//记录最长子序列数组1、2

int i1=0,i2=0;

void p(char t[],int n,int m){//根据记录路径数组1求最长子序列

if(m==0 || n==0) return;

if(re[n][m]==0)//判断来源是否为左上角

{

p(t,n-1,m-1);//传参为左上角进行递归

s1[i1]=t[n-1];

i1++;

}

if(re[n][m]==1)//判断来源是否为上边

p(t,n-1,m);//传参为上面一格进行递归

if(re[n][m]==2)//判断来源是否为左边

p(t,n,m-1);//传参为左边一格进行递归

}

void p1(char t[],int n,int m){//根据记录路径数组2求最长子序列 与p相同

if(m==0 || n==0) return;

if(re1[n][m]==0)

{

p1(t,n-1,m-1);

s2[i2]=t[n-1];

i2++;

}

if(re1[n][m]==1)

p1(t,n-1,m);

if(re1[n][m]==2)

p1(t,n,m-1);

}

int main()

{

int n,m,i,j,l=0,k;

cin>>n>>m;

//分别输入字符串长度 （这里只是作业要求

//实际上可以直接输入字符串然后用函数求长度）

getchar();

char a[n],b[m];

int z[n+1][m+1];//记录赋值数组

//初始化赋值

for(i=0;i<n+1;i++)//对第一列初始化

z[i][0]=0;

for(i=0;i<m+1;i++)//对第一行初始化

z[0][i]=0;

for(int i=0;i<n;i++)//分别输入字符串

scanf("%c",&a[i]);

getchar();

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

scanf("%c",&b[i]);

//对赋值数组进行判断和填充

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

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

{

if(a[i-1]==b[j-1])//判断行和列内容是否相同

{

z[i][j]=z[i-1][j-1]+1;//此格值等于左上角+1

re[i][j]=0;//设0代表来源为左上角

}

else if(z[i-1][j]>z[i][j-1])//上面的值大于左面的

{

z[i][j]=z[i-1][j];//赋值为上面的值

re[i][j]=1;//设1代表来源为上面

}

else //此时为左面的值大于等于上面的

{z[i][j]=z[i][j-1];//赋值为左面的值

re[i][j]=2;//设2代表来源为左面

}

}

//下面这次遍历路径存储在re1数组里

//与上面的区别是当上面和左面相等时取上面的值

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

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

{

if(a[i-1]==b[j-1])

{

re1[i][j]=0;//左上角

}

else if(z[i-1][j]>=z[i][j-1])//这里的判定条件不同

{

re1[i][j]=1;//上面

}

else {

re1[i][j]=2;//左面

}

}

//输出赋值数组

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

{

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

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

printf("\n");

}

printf("\n");

p(a,n,m);

p1(a,n,m);

for(i=0;i<i1;i++)

{

if(s1[i]!=s2[i]){//判断两种判定条件下所得最长公共子序列是否相同

k=1;break;

}

else k=0;

}

if(k==1)

{//不相同则全部输出

for(j=0;j<i1;j++)

printf("%c ",s1[j]);

printf("\n");

for(j=0;j<i2;j++)

printf("%c ",s2[j]);

}

else

for(j=0;j<i1;j++)

printf("%c ",s1[j]);

printf("\n");

return 0;

}