//后缀数组(知识点1:sa数组与倍增算法　　知识点2:rank数组与height数组)

//---------------知识点1---------------//

//后缀数组:把字符串str的所有后缀按字典序排序,sa[i]表示排名i的后缀的开头下标
//排序的规则是strcmp的规则比较不同的后缀的大小,基于ascii码值
/*
例子:str是"abccbac",假设str下标和sa下标从0开始
sa[0]=0------------>"abccbac"　　注意:对于后缀数组而言,整个字符串也是一种特殊的"后缀"
sa[1]=5------------>"ac"
sa[2]=4------------>"bac"
……　……
*/

//求解sa数组的方法:倍增算法
/*
先把以每个位置开始长度为1的字串进行排序,两个长度为1的字串可以拼接为一个长度为2的子串,由此进行长度为2的子串进行排序
同理两个长度为2的子串也可以拼接为一个长度为4的子串,以此类推,可以实现2^k的长度的子串的排序
*/

//时间复杂度:O(nlogn)

//理解sa数组:每一轮循环中,sa数组的排序依据是当前所有后缀的前k个字符(即长度为k的前缀(在下面的代码中就是2*j)),k表示当前判断的字符串的长度(第一第二关键字之和)
//理解x数组与y数组:分别代表长度为k/2的第一关键字和第二关键字的排序(在下面的代码中就是j)
//x[i]表示起始位置为i的第一关键字的排名,y[i]表示第二关键字排名第i名对应的第一关键字的起始位置

//---------------知识点2---------------//

//构建height数组:
//rank数组:rank[i]表示从原字符串数组的i下标开始的字符串的排名,相当于sa数组的逆逻辑,即sa[rank[i]]=i
//height数组:height[i]表示在sa数组中排名第i名和第i-1名的字符串的最长公共前缀
//h数组:h[i]=height[rank[i]]

//height数组的用处:求sa[i]和sa[j]所对应的字符串的最大公共前缀的长度
//LCP(sa[i],sa[j])的值(假设i<j)等于height数组从i+1下标到j下标的数组元素的最小值

//核心思想:height[rank[i]]>=height[rank[i-1]]-1　　即h[i]>=h[i-1]-1

/*
关系式证明过程如下:
若h[i-1]<=1，则结果显然成立

当h[i-1]>1时,假设k=sa[rank[i-1]-1],则h[i-1]=LCP(suffix(k),suffix(i-1))
由于h[i-1]>1,因此suffix(k+1)和suffix(i)的大小关系仍然和suffix(k)与suffix(i-1)的大小关系一样,即suffix(k+1)<suffix(i)
同时LCP(suffix(k+1),suffix(i))=LCP(suffix(k),suffix(i-1))-1=h[i-1]-1
虽然suffix(k+1)<suffix(i),但是sa[rank[i]-1]不一定是k+1,也就是说虽然相对次序没有改变,但是suffix(k+1)和suffix(i)不一定是紧邻的,而是满足
rank[k+1]<=rank[i]-1

LCP(suffix(k+1),suffix(i))的值是height数组从rank[k+1]+1下标到rank[i]下标的数组元素的最小值
LCP(sa[rank[i]-1],sa[rank[i]])=height[rank[i]]
由此可见前者跨越的height数组下标范围更广,因此LCP(suffix(k+1),suffix(i))<=LCP(sa[rank[i]-1],sa[rank[i]])
即h[i-1]-1<=h[i],证明成立
*/

第一部分：关于构建sa数组的函数

```
void build_sa(int n,int m,char * s,int * sa,int * x,int * y,int num)
{

    //---------------第一部分:对于长度为1的字符进行基数排名,实现sa数组的初始化---------------//

    int * ws=(int *)calloc(m,sizeof(int));      //创建一个基数数组
    for (int i=0;i<n;i++)
    {
        ws[x[i]=s[i]]+=1;
    }
    for (int i=1;i<m;i++)       //计算基数排序的前缀和,此时ws[i]的值表示ascii码值<=i的字符的个数
    {
        ws[i]+=ws[i-1];
    }
    for (int i=n-1;i>=0;i--)        //注意:一定是从后向前确定字符的排序位置,因为对于相同ascii码值的字符,在数组中靠后的字符对应的排序位也更靠后
    {
        sa[--ws[s[i]]]=i;       //由于标准sa数组的下标从0开始,而ws表示的个数从1开始,为保证sa数组下标从0开始,需要--ws[s[i]]而非ws[s[i]]--
    }
```

```
 int pos=0;
    for (int j=1;;j*=2)     //j表示当前字符串的长度的一半,大小等于第一关键字长度,等于第二关键字长度
    {

        //---------------第二部分:求出第二关键字的排序(更新y数组)---------------//

        pos=0;
        for (int i=n-j;i<n;i++)     //对于第二关键字不存在的情况直接放置在y数组最前面
                                    //当第二关键字不存在时相当于这些字符串的第二关键字都同时为0,此时直接按顺序放入不会出错,因为后续按照第一关键字排序时
会确定真正的位置
        {
            y[pos++]=i;
        }
        for (int i=0;i<n;i++)
        {
            if (sa[i]>=j)       //对于第二关键字存在的情况,需要根据第二关键字的大小依据升序标准放入y数组中
            {
                y[pos++]=sa[i]-j;
    }
}
```

```
//---------------第三部分:确定第一关键字的排序,并结合第二关键字排序确定sa数组的排序(更新sa数组)---------------//

    for (int i=0;i<m;i++)       //清空原来的基数数组
    {
        ws[i]=0;
    }
    for (int i=0;i<n;i++)
    {
        ws[x[y[i]]]++;
    }
    for (int i=1;i<m;i++)
    {
        ws[i]+=ws[i-1];
    }
    for (int i=n-1;i>=0;i--)
    {
        sa[--ws[x[y[i]]]]=y[i];     //在保持第二关键字的相对顺序不变后,根据第一关键字进行排序(这里是以第一关键字为主导,结合第二关键字进行的排序,因此此时sa数组排序的考虑范围已经变成2j了)
    }
```

```
    //---------------第四部分:更新x数组同时检验排序是否已经完成----------//

        {
            int * temp=x;
            x=y;
            y=temp;
        }//交换空间,实现空间复用:原本由于本轮循环中y已经用不到了,因此将y数组的空间用于容纳上一次的第一关键字排名,空出来的x数组的空间用于容纳新的x数组的排名



        num=0;      //让第一关键字的新排序也从0下标开始(在字符长度为1时直接使用ascii码当作排序,虽然相对位置正确但是下标没有从0开始导致比较混乱)
        x[sa[0]]=0;
        for (int i=1;i<n;i++)
        {
            x[sa[i]]=(y[sa[i]]==y[sa[i-1]]&&y[sa[i]+j]==y[sa[i-1]+j])?num:++num;        //第三部分的sa数组按顺序排序,没有处理相等的情况(即便完全相同也不是并列的排名,而是一前一后)
            //因此这行代码处理了相等的情况,即x数组允许并列排名的存在(本身也符合第一关键字的基数排序),若相邻字符串的第一关键字和第二关键字分别相同,则排名也相同,否则排名差1
                //若新x数组没有并列存在,就代表此时已经完成了全部排序
                //这里是要将sa数组2k排序范围提供给x数组,让x数组的排序范围也变成2k,为了下一次循环提供第一关键字的排序做准备
                //注意:为了防止数组下标越界,x和y应该开辟2n空间
        }
        if (num==n-1)
        {
            break;
        }
        m=num+1;
    }
}
```

第二部分：关于构建 height 数组的函数

```
void build_height(int n,char * s,int * sa,int * rank,int * height)
{
    for (int i=0;i<n;i++)
    {
        rank[sa[i]]=i;
    }
    int k=0;
    for (int i=0;i<n;i++)
    {
        if (rank[i]==0)
        {
            height[0]=0;
            continue;
        }
        if (k)
        {
            k--;
        }
        int j=sa[rank[i]-1];
        while(i+k<n&&j+k<n&&s[i+k]==s[j+k])
        {
            k++;
        }
        height[rank[i]]=k;
    }

}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXSIZE 200
void build_sa(int n,int m,char * s,int * sa,int * x,int * y,int num);
void build_height(int n,char * s,int * sa,int * rank,int * height);
int main(void)
{
    char * s=(char *)calloc(MAXSIZE,sizeof(char));
    int * sa=(int *)calloc(MAXSIZE,sizeof(int));
    int * x=(int *)calloc(MAXSIZE,sizeof(int));     //x数组表示第一关键字的排序结果
    int * y=(int *)calloc(MAXSIZE,sizeof(int));     //y数组表示第二关键字的排序结果
    strcpy(s,"mississippi");
    int num=0;
    build_sa(strlen(s),128,s,sa,x,y,num);
    for (int i=0;i<strlen(s);i++)
    {
        printf("%4d",sa[i]);
    }
    printf("\n");
    int * rank=(int *)calloc(strlen(s),sizeof(int));
    int * height=(int *)calloc(strlen(s),sizeof(int));
    build_height(strlen(s),s,sa,rank,height);
    for (int i=0;i<strlen(s);i++)
    {
        printf("%4d",rank[i]);
    }
    printf("\n");
    for (int i=0;i<strlen(s);i++)
    {
        printf("%4d",height[i]);
    }
    return 0;
}
```

主函数整合输出