《数据结构与算法》实验报告5

——字符串匹配算法和稀疏矩阵的转置算法

吉林大学数学学院2020级计算1班 姓名 陈文宇 学号10200115

日期 2023/5/28

一、实验题目

分别编写暴力匹配算法、首尾匹配算法、KMP算法思路并验证结果。

编写稀疏矩阵的三元组存储和转置算法。

二、算法简述（）

暴力匹配算法：从第i位开始匹配，若失败，则 i=i+1，重新开始匹配，若成功则返回i。

首尾匹配算法：从第i位开始匹配，比较首字符和尾字符，若失败，则 i=i+1，重新开始匹配，若成功继续对字符串内部做首尾匹配。

KMP算法思路：从第i位开始匹配，若在第i+j位失配，则寻找模式串的前缀子串和后缀子串中的最长公共子串，依据按其长度记录的next[j],向后移动到i+next[j]位，重新匹配即可。其中next数组组便是获取移动位序的数组,以S[i]结尾失配的前i个元素的最长公共后缀的首位索引 为 next[i]

稀疏矩阵转置算法：本质是计数排序，原稀疏矩阵的三元组是行优先，转置后相当于按列优先排序，可以使用计数排序直接将每个元素找到其对应的位置。

三、程序代码

|  |
| --- |
| String\_Basic\_Opration.cpp |
| #include<iostream>  #include<string>  #include<cstdlib>  using namespace std;  //设计一个合理的字符串结构  //使用字符串编写字符串基本操作  //编写字符串匹配算法  //BF FL KMP  typedef struct String{  char \*ch;  int length;  String\* next;  String(){ //默认构造函数  ch=NULL;  length=0;  next=NULL;  }  }String,\*String\_L; //定义并初始化  void StrAssign(String &T,char\* S);  int StrCompare(char\* T,char\* S);  int StrLength(char\* S);  void Concat(String &T, char \*S1, char \*S2);  void SubString(char \*Sub,char\* S,int pos, int len);  void GetNext(char\* S,int\* next,int lenS);  //字符串匹配算法  int Index\_BF(char\* S, char\* T, int pos);  int Index\_FL(char\* T, char\* S, int pos);  int Index\_KMP(char\* T, char \*S, int pos);  int main(){  String T;    char a[50]="hello world! ";  char c[50]="ChenWenyu! abcdefg";  char b[1000];  char d[50];  StrAssign(T,a);    cout<<"a="<<a<<endl;  cout<<"a\_length="<<StrLength(a)<<endl;    cout<<"b="<<T.ch<<endl;  cout<<"b\_length="<<StrLength(T.ch)<<endl;    cout<<"a compare b ="<<StrCompare(T.ch,a)<<endl;    Concat(T,a,c);      cout<<"b="<<T.ch<<endl;  cout<<"b\_length="<<StrLength(T.ch)<<endl;    SubString(d,T.ch,6,5);  printf("d = %s\n",d);  cout<<"d\_length = "<<StrLength(d)<<endl;    cout<<"暴力匹配算法：";  cout<< Index\_BF(T.ch ,d,0)<<endl;  cout<<"首尾匹配算法：";  cout<< Index\_FL(T.ch ,d,0)<<endl;  cout<<"KMP匹配算法：";  cout<< Index\_KMP(T.ch ,d,0)<<endl;  }  //字符串基本操作  void StrAssign(String &T,char\* S){  if(T.ch) free(T.ch);  int k=0;  while(S[k++]!='\0');  T.ch=(char\*)malloc(sizeof(char)\*k);  for(int i=0; i<k; i++) T.ch[i]=S[i];  T.length=k;  }  int StrCompare(char\* T,char\* S){  int i;  for(i=0; S[i]==T[i] && S[i] !='\0' && T[i] != '\0'; i++ );  if(S[i] = T[i]) return 0;  if(S[i] < T[i]) return -1;  if(S[i] > T[i]) return 1;  }  //不包括'\0' 的元素的个数  int StrLength(char\* S){  int k=0;  while(S[k++] != '\0');  return k-1;  }  void Concat(String &T, char \*S1, char \*S2){    int i=0,k=0;  k=StrLength(S1)+StrLength(S2)+1;  T.ch=(char\*)malloc( sizeof(char)\*k);  k=0;  while(S1[i]!= '\0') T.ch[k++]=S1[i++];  i=0;  while(S2[i]!= '\0') T.ch[k++]=S2[i++];  T.ch[k]='\0';  }  void SubString(char \*Sub,char\* S,int pos, int len){  int i;  for(i=0; i<len ;i++){  Sub[i]=S[i+pos];  }  Sub[i]='\0';  }  //暴力匹配算法  int Index\_BF(char\* S, char\* T, int pos){  int i=pos,j=0;  while(S[i+j] != '\0' && T[j] != '\0'){  if( S[i+j] == T[j]) j++;  else{i++; j=0;}  }  if(T[j]=='\0') return i;  else return -1;  }  //首尾匹配算法  int Index\_FL(char\* T, char\* S, int pos){  int lenS=StrLength(S);  int lenT=StrLength(T);  int i=pos,k=0,j=0;  if(lenT-pos<lenS) return -1;  else{  for(; i<=lenT-lenS; i++){  for(; k<=lenS/2;k++){  if(T[i+k] != S[k] ) break;  if(T[i+lenS-1-k]!=S[lenS-1-k]) break;  if(k==lenS/2) j=1;  }  if(j) return i;  else k=0;  }  }  return -1;  }  //匹配失败后。子串 移至 失配位前的最长公共后缀上，即可 next组便是获取移动位序的数组  int Index\_KMP(char\* T, char \*S, int pos){  int lenS=StrLength(S);  int lenT=StrLength(T);  int next[lenS];  GetNext(S,next,lenS);  int i=pos,j=0,k;  if(lenT-pos<lenS) return -1;  else{  while(i<=lenT-lenS){  while(j<lenS && T[i+j]==S[j]) j++;  if(j==lenS) return i;  else {  i=i+lenS-next[j];  j=next[j]+1;  }  }  }  return -1;  }  void GetNext(char\* S,int\* next,int lenS){  int i=0,j=-1;  // 以S[i]结尾失配 的 前 i 个元素的最长公共后缀的首位索引 为 next[i]  next[0]=-1; //默认 S[0] 结尾为失配 默认其索引为 -1  while (i < lenS){  if (j == -1 || S[i] == S[j]){  ++i;++j;  next[i] = j;  }  else j = next[j];  }  } |
| TSMaxtrix.cpp |
| //#include"stdafx .h"  #include<iostream>  #include<stdlib.h>  #include<malloc.h>  using namespace std;  const int MAXSIZE=100;  typedef int ElemType;    typedef struct{  int i,j; //非零元的行下标和列下标  ElemType e; //该非零元的元素值  }Triple;  typedef struct{  Triple data[MAXSIZE]; //非零元三元组表，data[0]未用  int mu,nu,tu; //稀疏矩阵的行数，列数和非零元个数  }TSMatrix;    void TSMattrans(int\*\* M,TSMatrix &MS,int m,int n);  void transpose(int\*\* M, int\*\* T,int m,int n);  void coutMat(int\*\* M,int m, int n);  void coutTSMat(TSMatrix MS);  void createRpos(TSMatrix MS,int\* rpos);  void FastTransposeTSMatrix(TSMatrix MS,TSMatrix &TS);  int main(){    int m=3,n=3;  int\*\* M=new int\*[m];  int\*\* T=new int\*[m];  TSMatrix MS,TS;    //初始化  for(int i=0;i<m;i++){  M[i]=new int[n];  T[i]=new int[n];  }  MS.mu=m; MS.nu=n;  TS.mu=n; TS.nu=m;  for(int p=0;p<m;p++){  for(int q=0;q<n;q++){  M[p][q]=0;  }  M[p][p]=p+1;  }  M[0][n-1]=1;    //以三元组 形式存储  TSMattrans(M,MS,m,n);  coutMat(M,m,n);  cout<<"MSMatrix = "<<endl;  coutTSMat(MS);    //一般矩阵转置  transpose(M,T,m,n);  TSMattrans(T,TS,m,n);  coutMat(T,m,n);  cout<<"TSMatrix = "<<endl;  coutTSMat(TS);    //稀疏矩阵装置  FastTransposeTSMatrix(MS,TS);  cout<<"TSMatrix = "<<endl;  coutTSMat(TS);  //销毁矩阵  for(int p=0;p<m;p++){  delete[] M[p];  delete[] T[p];  }  delete[] M;  delete[] T;  }  //用于录入稀疏矩阵 并以三元组 形式存储  void TSMattrans(int\*\* M,TSMatrix &MS,int m,int n){  int k=1;  for(int p=0; p<m; p++){  for(int q=0; q<n; q++){  if(M[p][q] != 0 ){  MS.data[k].i=p+1;  MS.data[k].j=q+1;  MS.data[k].e=M[p][q];  k++;  }  }  }  MS.tu=k-1;  }  //num[] 计数器 rops[] 计序器  void createRpos(TSMatrix MS,int\* rpos){  int nums[MS.nu+1];  int p;  //rops 求 M 中每一列的第一个非零元 在T.data 中的序号 称为位序器  //nums 求 M 中每一列的非零元的个数  for(p=1;p<=MS.nu;p++) nums[p]=0;  for(p=1;p<=MS.tu;p++) nums[MS.data[p].j]++;  rpos[1]=1;  for(p=2;p<=MS.nu;p++) rpos[p]=rpos[p-1]+nums[p-1];  }  //一般矩阵转置  void transpose(int\*\* M, int\*\* T,int m,int n){    for(int p=0;p<m;p++){  for(int q=0;q<n;q++){  T[q][p]=M[p][q];  }  }  }  //稀疏矩阵转置  void FastTransposeTSMatrix(TSMatrix MS,TSMatrix &TS){  int rpos[MS.nu+1];  int q,col;  if(MS.tu){  createRpos(MS,rpos);    for(int p=1;p<=MS.tu;p++){  col=MS.data[p].j;  q=rpos[col];    TS.data[q].i=MS.data[p].j;  TS.data[q].j=MS.data[p].i;  TS.data[q].e=MS.data[p].e;  ++rpos[col];//同一行的下一个非零元的位置加 1  }//for  }//if  }  //矩阵输出  void coutMat(int\*\* M,int m, int n){  cout<<"Matrix = "<<endl;  for(int i=0;i<m;i++){  for(int j=0;j<n;j++){  cout<<M[i][j]<<" ";  }  cout<<endl;  }  cout<<endl;  }  //三元组输出  void coutTSMat(TSMatrix MS){    for(int p=1;p<=MS.tu;p++){  cout<<"("<<MS.data[p].i<<","<<MS.data[p].j<<","<<MS.data[p].e<<")"<<endl;  }  cout<<endl;  } |

四、实验结果



