



南京林业大学  
NANJING FORESTRY UNIVERSITY

2024 ~ 2025 学年第 1 学期 课程设计

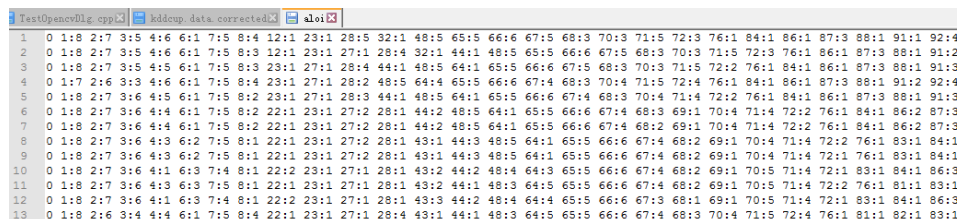
课程名称: C++实习

班号	23508014
专业	计算机科学与技术
学号	2351610105
姓名	方泽宇
教师	常娜
内容目录	
1、题目.....2	
2、需求分析..... 2	
3、方案设计.....2	
4、核心代码.....2	
5、测试用例及运行结果.....7	
6、总结: .....10	
本人签名:	
2024 年 12 月 23 日	

## 题目：稀疏数据文件处理程序

为节省存储空间和提高文件的网络传输效率，数据文件常采用稀疏方式存储，如图像压缩、稀疏编码等技术。而在计算时，又需要从稀疏数据（sparse data）中恢复出原始数据(full data)，以便采用向量或矩阵运算。现有如下稀疏数据，如 LIBSVM 提供的公开数据 aloi 文件（附下载网址），格式如下图所示：

附网址：<https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/aloi.bz2>



1	0	1:8	2:7	3:5	4:6	6:1	7:5	8:4	12:1	23:1	28:5	32:1	48:5	65:5	66:6	67:5	68:3	70:3	71:5	72:3	76:1	84:1	86:1	87:3	88:1	91:1	92:4
2	0	1:8	2:7	3:5	4:6	6:1	7:5	8:3	12:1	23:1	27:1	28:4	32:1	44:1	48:5	65:5	66:6	67:5	68:3	70:3	71:5	72:3	76:1	86:1	87:3	88:1	91:2
3	0	1:8	2:7	3:5	4:5	6:1	7:5	8:3	23:1	27:1	28:4	44:1	48:5	64:1	65:5	66:6	67:5	68:3	70:3	71:5	72:2	76:1	84:1	86:1	87:3	88:1	91:3
4	0	1:7	2:6	3:3	4:6	6:1	7:5	8:4	23:1	27:1	28:2	48:5	64:4	65:5	66:6	67:4	68:3	70:4	71:5	72:4	76:1	84:1	86:1	87:3	88:1	91:2	92:4
5	0	1:8	2:7	3:6	4:5	6:1	7:5	8:2	23:1	27:1	28:3	44:1	48:5	64:1	65:5	66:6	67:4	68:3	70:4	71:4	72:2	76:1	84:1	86:1	87:3	88:1	91:3
6	0	1:8	2:7	3:6	4:4	6:1	7:5	8:2	22:1	23:1	27:2	28:1	44:2	48:5	64:1	65:5	66:6	67:4	68:3	69:1	70:4	71:4	72:2	76:1	84:1	86:2	87:3
7	0	1:8	2:7	3:6	4:4	6:1	7:5	8:2	22:1	23:1	27:2	28:1	44:2	48:5	64:1	65:5	66:6	67:4	68:2	69:1	70:4	71:4	72:2	76:1	84:1	86:2	87:3
8	0	1:8	2:7	3:6	4:3	6:2	7:5	8:1	22:1	23:1	27:2	28:1	43:1	44:3	48:5	64:1	65:5	66:6	67:4	68:2	69:1	70:4	71:4	72:2	76:1	83:1	84:1
9	0	1:8	2:7	3:6	4:3	6:2	7:5	8:1	22:1	23:1	27:2	28:1	43:1	44:3	48:5	64:1	65:5	66:6	67:4	68:2	69:1	70:4	71:4	72:1	76:1	83:1	84:1
10	0	1:8	2:7	3:6	4:1	6:3	7:4	8:1	22:2	23:1	27:1	28:1	43:2	44:2	48:4	64:3	65:5	66:6	67:4	68:2	69:1	70:5	71:4	72:1	83:1	84:1	86:3
11	0	1:8	2:7	3:6	4:3	6:3	7:5	8:1	22:1	23:1	27:1	28:1	43:2	44:1	48:3	64:5	65:5	66:6	67:4	68:2	69:1	70:5	71:4	72:2	76:1	81:1	83:1
12	0	1:8	2:7	3:6	4:1	6:3	7:4	8:1	22:2	23:1	27:1	28:1	43:3	44:2	48:4	64:4	65:5	66:6	67:3	68:1	69:1	70:5	71:4	72:1	83:1	84:1	86:3
13	0	1:8	2:6	3:4	4:4	6:1	7:5	8:4	22:1	23:1	27:1	28:4	43:1	44:1	48:3	64:5	65:5	66:6	67:4	68:3	70:4	71:5	72:4	76:1	81:1	82:1	83:1

图 1. 机器学习常用数据 aloi 的数据格式

文件的第一列表示样本的类别，共有 1000 类，采用 0-999 标记；而对于二分类数据，其类别符号采用“+1”和“-1”标示。图 1 中，每一行表示一个样本，样本的结束符采用回车符。例如，第一行中的“76:1”，它表示该样本的第 76 个属性值为 1。每行中未列出的属性，它们的属性值均为 0，故无需在文件中存储。按要求完成以下任务：

- 自动计算 LIBSVM 类型数据的样本数和特征数（属性个数）。
- 从稀疏文件中恢复出全部数据（去除列标记，每个样本的属性值全部列出，以空格分隔），并将类别标记写入与该文件对应的文本文件中，如记为“aloi\_full.txt”和“aloi\_label.txt”
- 为便于共享计算结果，结果仍采用稀疏形式存储和传输，即实现问题 2 的逆变换。受限于目前知识，本题暂不考虑计算问题，仅要求从格式形如文件“aloi\_full.txt”和“aloi\_label.txt”，获得稀疏的 aloi 数据，记为“restore\_aloi.txt”。
- 比对 aloi 和 restore\_aloi.txt 文件的差异，并记录文件正逆变换的时间，将结果回显在屏幕上。

### 需求分析：

- 题目需要我们将一个压缩后的向量文件给恢复成原始文件，并统计特征向量的数量和特征向量的种类数量。然后再将原始的向量文件压缩，并对压缩前后的文件进行比较。
- 题目需要统计每一步操作所需要的时间。

### 方案设计：

- 创建一个类的实例，在实例中读取压缩后的文件，逐行读取并对数据文件进行处理。在完成实习之后我发现其实可以不写的那么复杂，可以直接通过格式化输入完成这样的一个过程。
- 程序采用面向对象的编程思路，将函数进行类封装，仅对外暴露极少部分的函数。

### 核心代码：

```
1. #include<stdio.h>
2. #include<string.h>
3. #include<time.h>
4. #include<stdlib.h>
5. const int LINE_MAX=1e5;
6. const int VECTOR_MAX=129;
```

```

7.     const int SAMPLE_MAX=1e3+10;
8.     class Line
9.     {
10.         private:
11.             char line[LINE_MAX];
12.             int vectors[VECTOR_MAX];
13.             int sample_cnt[SAMPLE_MAX];
14.             FILE *fp,*wfp,*lfp;
15.             int sample_num;
16.             int eigenvector;// 记录特征向量, 也就是第一个参数的值
17.             void read_file(const char filename[])
18.             {
19.                 fp=fopen(filename,"r+");
20.                 if(fp==nullptr) printf("File open failed\n");
21.             }
22.             void write_file(const char filename[],const char lable[])
23.             {
24.                 wfp=fopen(filename,"w");
25.                 lfp=fopen(lable,"w");
26.                 if(wfp==nullptr||lfp==nullptr) printf("File open failed\n");
27.             }
28.             void rtrim(char *str)
29.             {
30.                 size_t len=strlen(str);
31.                 while(len>0&&(str[len-1]==' '||str[len-1]=='\n')) str[--len]='\0';
32.             }
33.         public:
34.             bool read_one_line()// 读取成功返回1, 否则返回0
35.             {
36.                 ++sample_num;
37.                 fgets(line,sizeof line,fp);
38.                 int linelength=strlen(line);
39.                 int tmp=0,tmp_cnt=0;
40.                 int vector_position;// 记录向量的位置
41.                 memset(vectors,0,sizeof vectors);
42.                 line[linelength-1]=' ';
43.                 line[linelength]='\0';
44.                 linelength=strlen(line);
45.                 for(int i=0;i<linelength+1;++i)
46.                 {
47.                     if(line[i]!=' ' && line[i]!=':')
48.                     {
49.                         tmp=tmp*10+line[i]-'0';
50.                     }

```

```

51.         else if(line[i]==' '&&tmp_cnt==0)
52.         {
53.             eigenvector=tmp;
54.             ++sample_cnt[tmp];
55.             tmp=0;
56.             ++tmp_cnt;
57.         }
58.         else if(line[i]==' '||i==linelength-1)
59.         {
60.             vectors[vector_position]=tmp;
61.             tmp=0;
62.             ++tmp_cnt;
63.         }
64.         else if(line[i]==':')
65.         {
66.             vector_position=tmp;
67.             tmp=0;
68.             ++tmp_cnt;
69.         }
70.     }
71.     if(feof(fp)) return 0;
72.     else return 1;
73. }
74. void write_one_line()
75. {fprintf(lfp,"%d ",eigenvector);
76.     for(int i=1;i<=1<<7;++i)
77.     {
78.         fprintf(wfp,"%d ",vectors[i]);
79.     }
80.     fprintf(wfp,"\n");
81. }
82. void print_newest_line()
83. {
84.     for(int i=1;i<=1<<7;++i)
85.     {
86.         printf("%d ",vectors[i]);
87.     }
88. }
89. void restore_file(const char lable[],const char full[],const char
restore[])
90. {
91.     FILE *restore_lable=fopen(lable,"r+");
92.     FILE *restore_full=fopen(full,"r+");
93.     FILE *restore_restore=fopen(restore,"w");
94.     do

```

```

95.         {
96.             fscanf(restore_lable,"%d",&eigenvector);
97.             fprintf(restore_restore,"%d ",eigenvector);
98.             fgets(line,sizeof line,restore_full);
99.             int linelength=strlen(line);
100.            line[linelength]=' ';
101.            int tmp=0,tmp_cnt=0;
102.            for(int i=0;i<linelength;++i)
103.            {
104.                if(line[i]==' ')
105.                {
106.                    ++tmp_cnt;
107.                    if(tmp==0) continue;
108.                    else
109.                    {
110.                        fprintf(restore_restore,"%d:%d ",tmp_cnt,tmp);
111.                    }
112.                    tmp=0;
113.                }
114.                else tmp=tmp*10+line[i]-'0';
115.            }
116.            fprintf(restore_restore,"\n");
117.        }while(!feof(restore_full)||!feof(restore_lable));
118.        fclose(restore_lable);
119.        fclose(restore_full);
120.        fclose(restore_restore);
121.    }
122.    void display_sample()
123.    {
124.        int cnt=0;
125.        for(int i=0;i<=999;++i)
126.            if(sample_cnt[i]) ++cnt;
127.        printf("eigenvector cnt:%d\n",cnt);
128.        printf("sample cnt:%d\n",sample_num);
129.    }
130.    void compare(const char file_1[],const char file_2[])
131.    {
132.        FILE *file1=fopen(file_1,"r+");
133.        FILE *file2=fopen(file_2,"r+");
134.        if (!file1||!file2)
135.        {
136.            printf("Compare:File open failed\n");
137.            if (file1) fclose(file1);
138.            if (file2) fclose(file2);
139.        }

```

```

140.         char f1[LINE_MAX],f2[LINE_MAX];
141.         int line_number=0;
142.         while(1)
143.         {
144.             char *line1=fgets(f1,sizeof(f1),file1);
145.             char *line2=fgets(f2,sizeof(f2),file2);
146.             ++line_number;
147.             if((!line1&&!line2)||strcmp(line1,"\n")||strcmp(line2,"\n"
148. ))
149.             {
150.                 fclose(file1);
151.                 fclose(file2);
152.                 printf("File is same\n");
153.                 return;
154.             }
155.             if((!line1||!line2)&&(strlen(line1)>2||strlen(line2)>2))
156.             {
157.                 printf("Line %d different\n",line_number);
158.                 printf("f1:%s",f1);
159.                 printf("f2:%s\n",f2);
160.                 fclose(file1);
161.                 fclose(file2);
162.                 return;
163.             }
164.             rtrim(f1);
165.             rtrim(f2);
166.             if (strcmp(f1,f2)!=0)
167.             {
168.                 printf("Line %d different\n",line_number);
169.                 printf("f1:%s\n",f1);
170.                 printf("f2:%s\n",f2);
171.                 fclose(file1);
172.                 fclose(file2);
173.                 return;
174.             }
175.         }
176.         Line(const char readfile[],const char writefile[],const char write
177.         lable[])
178.         {
179.             read_file(readfile);
180.             write_file(writefile,writelable);
181.         }
182.         ~Line()
183.         {

```

```

183.         memset(line,0,sizeof line);
184.         fclose(fp);
185.         fclose(lfp);
186.         fclose(wfp);
187.     }
188. };
189. int main()
190. {
191.     Line *line=new Line("aloi","aloi_full.txt","aloi_lable.txt");
192.     int cnt=0;
193.     clock_t start=clock();
194.     double cpu_time_used;
195.     while(line->read_one_line()) line->write_one_line();
196.     line->display_sample();
197.     //line->print_newest_line();
198.     clock_t end=clock();
199.     cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
200.     printf("Unzip_Process_Time%.2fseconds\n", cpu_time_used);
201.     start=clock();
202.     line->restore_file("aloi_lable.txt","aloi_full.txt","aloi_restore.
    txt");
203.     end=clock();
204.     cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
205.     start=clock();
206.     line->compare("aloi","aloi_restore.txt");
207.     //printf("%d",line->debug);
208.     delete line;
209.     end=clock();
210.     cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
211.     printf("Compare_Process_Time%.2fseconds\n", cpu_time_used);
212.     return 0;
213. }

```

## 测试用例及运行结果:

在 macOS14.6.1 操作系统下, 采用如下编译命令:

```
g++ -std=c++14 conduct.cpp -o conduct
```

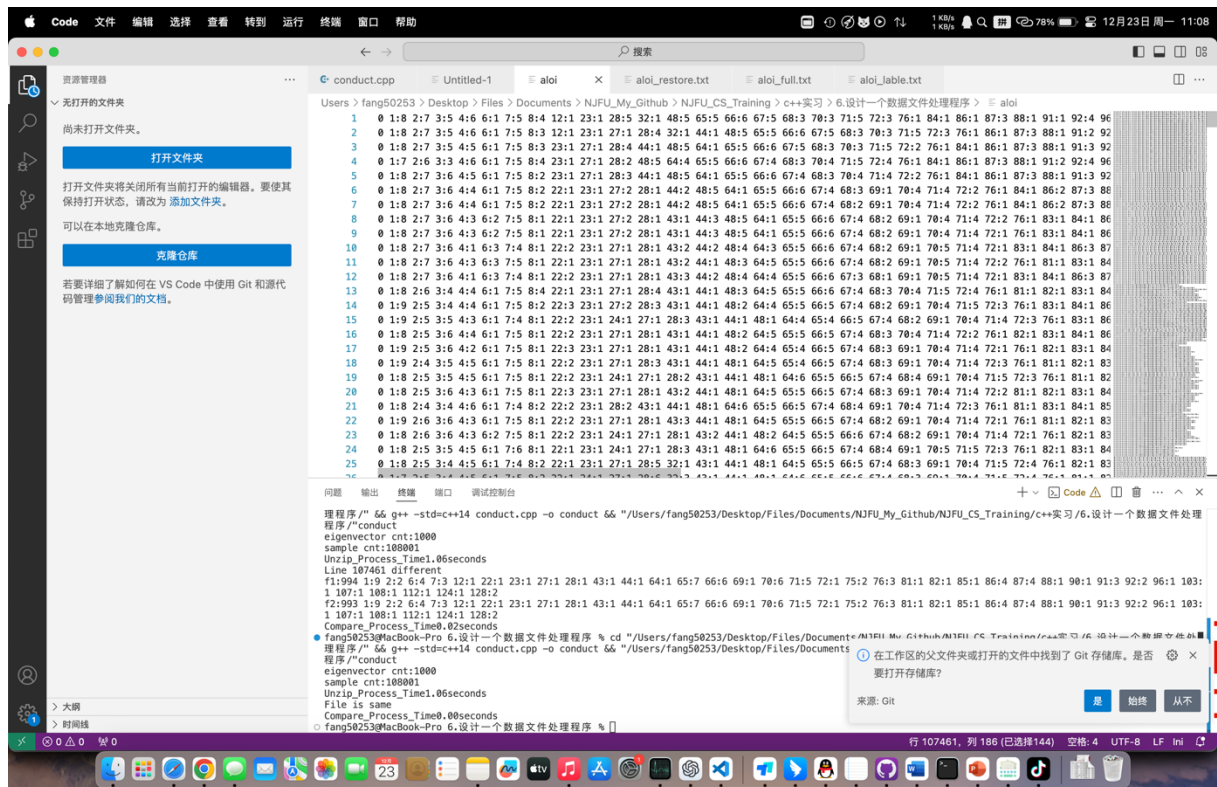
运行结果如下:

```

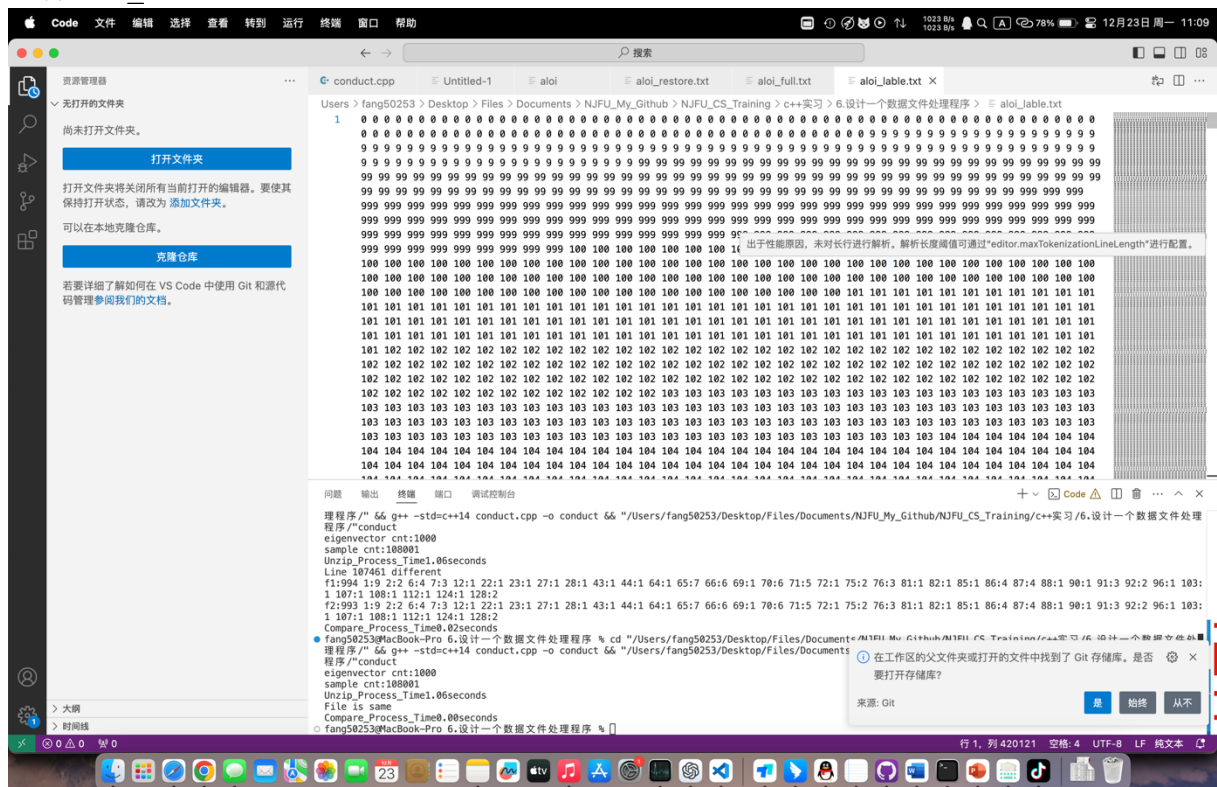
eigenvector cnt:1000
sample cnt:108001
Unzip_Process_Time1.06seconds
File is same
Compare_Process_Time0.00seconds

```

## 文件 aloi 摘要:

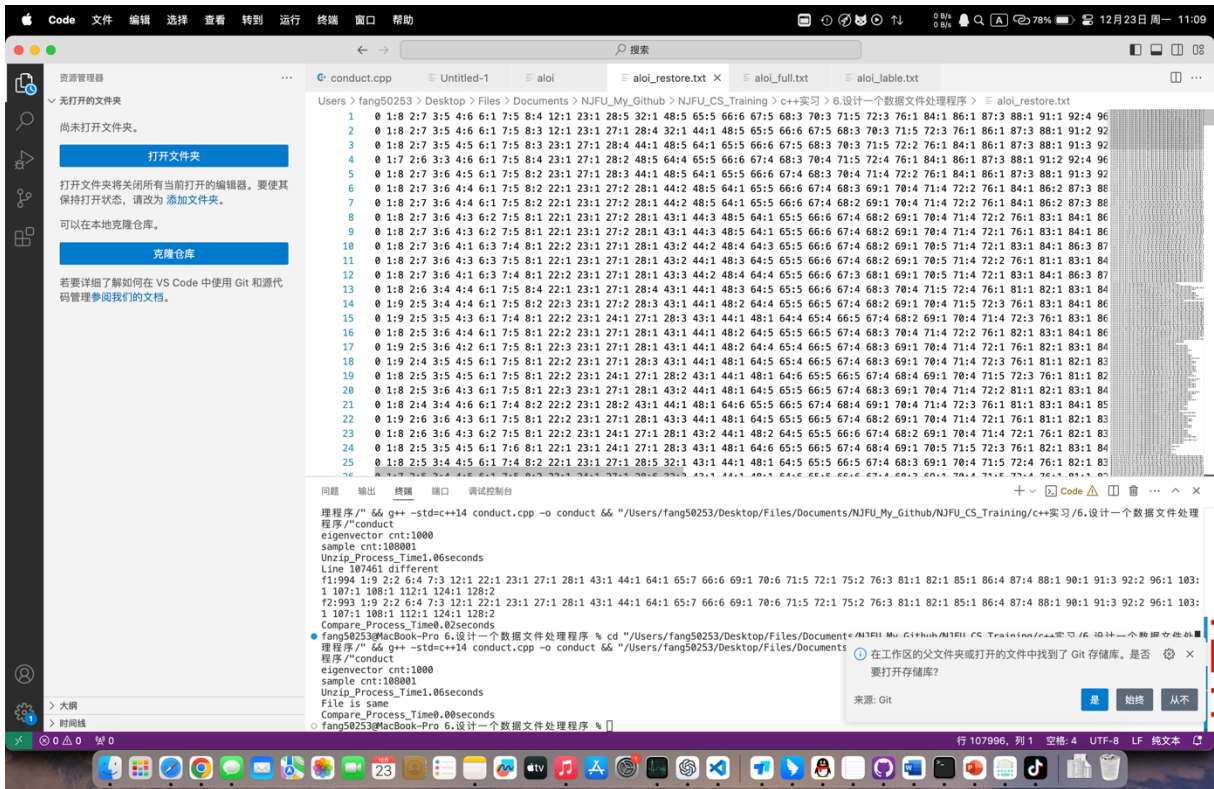


## 文件 aloi\_lable.txt 摘要:

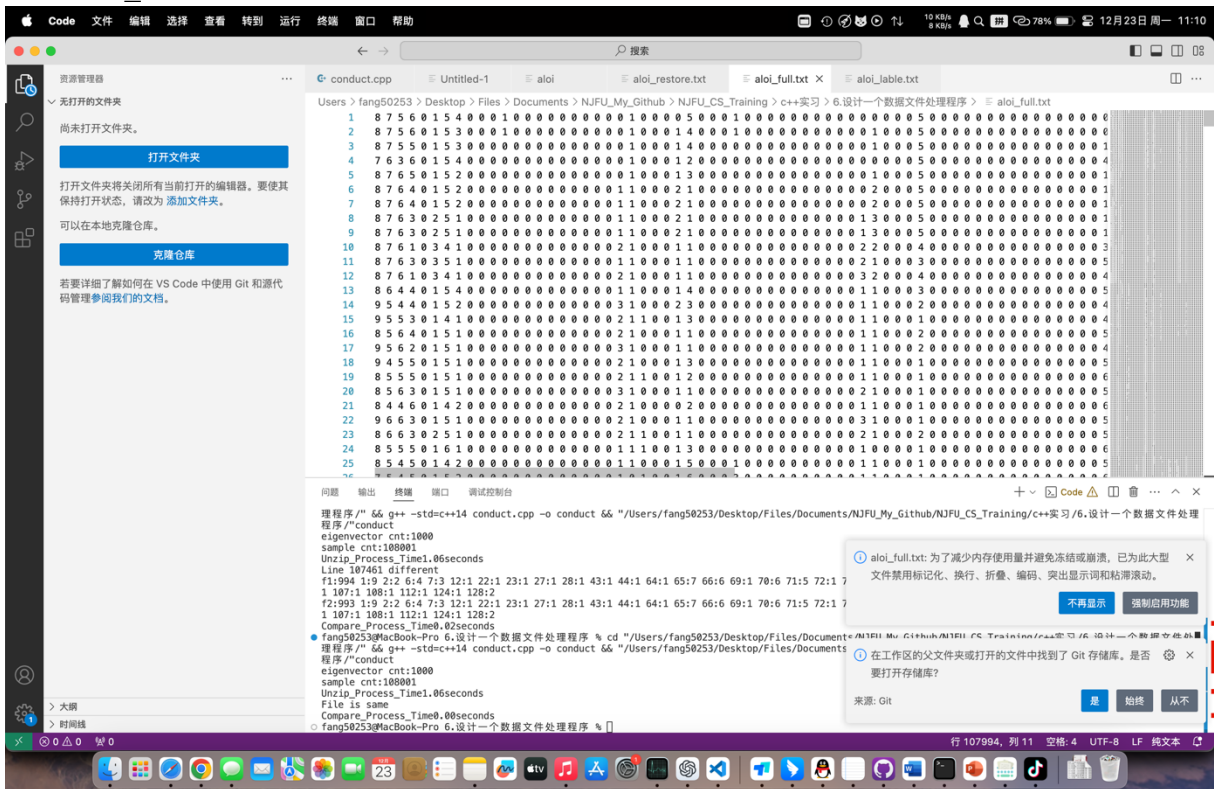


## 文件 aloi\_restore.txt 摘要:





## 文件 aloi\_full.txt 摘要:



## 总结:

- 本次实习过程中我使用了大量的字符串处理代码，但是后来发现可以通过文件的格式化输入避开这一麻烦的操作，并且也有助于提高代码的运行效率。
- 在本次实习中我遇到过文件比较不一致的问题，后来通过修改代码改变了文件读取到末尾的判断方式，从而解决了这个问题。

## 说明：

1. 题目可以直接复制；
2. 需求分析：可根据题目要求，分析需要解决什么样的问题；
3. 设计思想和方案设计：该部分可写设计思路，如怎样构造类，该类的变量及成员函数功能是什么；变量、函数名规定等等，特别是抽象问题是如何具体化的；
4. 核心代码：如果代码比较多，复制主要代码即可，量少就全部复制；
5. 测试用例：如果少数键盘输入，直接给出用例；以文件输入，交待下文件格式，并附简要截图；
6. 总结部分：主要学会了什么，还有哪些问题没能实现（与现实问题相比较，因为专业知识所限，目前我们能解决的问题往往都是实际问题的简化版）；课设过程遇到了什么样的问题，你是如何解决的；对比课设前后都有哪方面的提高；后续如何学习等。

