1. 需求分析

实验一：二叉树的最近公共祖先问题

问题描述：

设计一个算法，对于给定的树中2 结点返回它们的最近公共祖先。

编程任务：

对于给定的树，和树中结点对，编程计算结点对的最近公共祖先。

实验二：图的2着色问题

问题描述：

对于给定的图G，如果存在一种用两种颜色对顶点着色的方案，使得图中任意一条边所连接的2个顶点着有不同颜色，则称图G是可2着色的。

编程任务：

对于给定的图G，计算图G是否可2着色的。

实验三：

问题描述：

给定一本辞典dict和一段文本text，现在要统计文本text中不在辞典dict中的单词，且希望找到text中出现次数最多的不在辞典dict中的单词。

编程任务：

设计并实现解辞典问题的O(nlogn)时间算法，对于给定的辞典dict和文本text，找出text中出现次数最多的不在辞典dict中的单词。

实验四：DNA排序问题

问题描述：

对于给定的全序集中排序元素序列{ a1, a2,... , an}，元素ai 的逆序数定义为

inv(ai)=|{ak|ai>ak,i<k<=n}|。序列A 的逆序数定义为inv(A)=inv(a1)+inv(a2)+.....+inv(an)事实上，序列A 的逆序数刻画出序列A中元素已排序的程度。逆序数越小，序列A 已排序的程度

就越高。当序列A 已排好序时，其逆序数为0。

生物信息学家在进行分子计算研究DNA序列时需要将若干长度相同的DNA串按其逆序

数从小到大排序。例如，给定6个长度为10的DNA串：AACATGAAGG，TTTTGGCCAA，

TTTGGCCAAA ，GATCAGATTT ，CCCGGGGGGA ，ATCGATGCAT ，按其逆序数从

小到大排序为：CCCGGGGGGA，AACATGAAGG，GATCAGATTT，ATCGATGCAT，

TTTTGGCCAA，TTTGGCCAAA。

DNA 排序问题就是要对给定的长度相同的DNA 串按逆序数排序。

编程任务：

对于给定的长度相同的DNA 串，按其逆序数从小到大排序。

二、概要设计

1.抽象数据类型

实验三：

struct node//字典树的单个结点

{

int num;

node\* next[52];

}\*dict, \*re;

实验四：

struct DNA//包含DNA序列以及逆序数

{

int nixu;

char xulie [110];

};

2.算法

三、详细设计

程序代码（注释）

实验一：

#include <stdio.h>

#define MAXN 1000

#include <vector>

#include <string.h>

#include <fstream>

#include <iostream>

int father[MAXN];

int str1[500], str2[500];//父节点列表

int len1, len2;

void getFather(int node, int &len, int \*str)

{

//由于不确定树的阶数，故要得到结点的所有父节点比较

while(node != 0)

{

node = father[node];

str[len++] = node;

}

}

int main()

{

FILE \*fp = fopen("input.txt", "r");

FILE \*fp2 = fopen("output.txt", "w");

memset(father, 0, sizeof(father));

int n;

fscanf(fp, "%d", &n);

int num, to, tmp;

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

{

fscanf(fp, "%d", &num);

while(num--)

{

fscanf(fp, "%d", &to);

father[to] = i;//建树

}

}

fscanf(fp, "%d", &num);

int a, b;

while(num--)

{

fscanf(fp, "%d%d", &a, &b);

fprintf(fp2, "%d %d ", a, b);

if(a == b)

{

printf("%d\n", a);//同一个结点

continue;

}

memset(str1, 0, sizeof(str1));

memset(str2, 0, sizeof(str2));

len1 = len2 = 0;

getFather(a, len1, str1);

getFather(b, len2, str2);

tmp = std::min(len1, len2);

for(int i = tmp; i > 0; i--)//同深度的父结点从下至上比较

{

if(str1[len1-i] == str2[len2-i])

{

fprintf(fp2, "%d\n", str1[len1-i]);

break;

}

}

}

pclose(fp);

pclose(fp2);

return 0;

}

实验二：

#include <stdio.h>

#include <string.h>

#define MAXN 1000

#include <fstream>

int father[MAXN<<1];

//并查集

int getFather(int n)//查

{

if(father[n] == 0)

return n;

return father[n] = getFather(father[n]);

}

void unions(int a, int b)//并

{

a = getFather(a);

b = getFather(b);

if(a != b)

father[a] = b;

}

int main()

{

FILE \*fp = fopen("input.txt", "r");

FILE \*fp2 = fopen("output.txt", "w");

int n, m;

memset(father, 0, sizeof(father));

fscanf(fp, "%d%d", &n, &m);

int a, b;

bool flag = true;

while(m--)

{

fscanf(fp, "%d%d", &a, &b);

unions(a, b + n);//a与b标记为不同集合

unions(b, a + n);

if(getFather(a) == getFather(a+n) || getFather(b) == getFather(b+n))

{

flag = false;//出现矛盾情况

}

}

if(flag)

{

fprintf(fp2, "Yes\n");

int temp = getFather(1);

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

if(getFather(i) == temp)

fprintf(fp2, "1");

else

fprintf(fp2, "0");

printf("\n");

}

else

fprintf(fp2, "No\n");

pclose(fp);

pclose(fp2);

return 0;

}

实验三：

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <fstream>

using namespace std;

struct node

{

int num;//以该结点为结尾的字符串个数

node\* next[52];

}\*dict, \*re;

int add(char str[], node \*head)

{

//字典树的建立，并返回当前串加入后该串的个数

node \*p = head;

for(int i = 0; str[i] != '\0'; i++)

{

int tmp = str[i]-'A';

if(p->next[tmp] == NULL)

{

p->next[tmp] = new node;

p->num = 0;

}

p = p->next[tmp];

}

p->num++;

return p->num;

}

bool query(char str[], node \*head)

{

//字典树的查询

node \*p = head;

for(int i = 0; str[i] != '\0'; i++)

{

int tmp = str[i]-'A';

if(p->next[tmp] == NULL)

return false;

p = p->next[tmp];

}

if(p->num == 0)

return false;

else

return true;

}

int main()

{

FILE \*fp = fopen("input.txt", "r");

FILE \*fp2 = fopen("output.txt", "w");

int n, tmp = 0, mx = 0;

char str[15];

fscanf(fp, "%d", &n);

dict = new node;

re = new node;

while(n--)

{

fscanf(fp, "%s", str);

add(str, dict);

}

fscanf(fp, "%d", &n);

while(n--)

{

fscanf(fp, "%s", str);

if(!query(str, dict))

tmp = add(str, re);

mx = max(mx, tmp);//不存在的串放入另外一个串，并求最大值

}

fprintf(fp2, "%d\n", mx);

pclose(fp);

pclose(fp2);

return 0;

}

实验四：

#include <stdio.h>

#include <algorithm>

#include <fstream>

using namespace std;

struct DNA

{

int nixu;

char xulie [110];

};

int nixushu(char\* xulie, int len)

{

int nixu = 0;

int a = 0 ,c = 0, g = 0;//numberOf

for(int i = len - 1; i >= 0; i--)

{

switch(xulie[i])

{

case 'A':

a++;

break;

case 'C':

c++;

nixu += a;

break;

case 'G':

g++;

nixu += a;

nixu += c;

break;

case 'T':

nixu += a;

nixu += c;

nixu += g;

break;

}

}

return nixu;

}

bool cmp(DNA a, DNA b)

{

return a.nixu < b.nixu;

}

int main()

{

FILE \*fp = fopen("input.txt", "r");

FILE \*fp2 = fopen("output.txt", "w");

int len, num;

while(fscanf(fp, "%d%d", &len, &num) && len)

{

DNA\* arr = new DNA [num];

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

{

fscanf(fp, "%s", arr[i].xulie);

arr[i].nixu = nixushu(arr[i].xulie, len);

}

sort(arr, arr+num, cmp);

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

fprintf(fp2, "%s\n", arr[i].xulie);

delete [] arr;

}

pclose(fp);

pclose(fp2);

return 0;

}

四、调试分析

调试过程中所做的工作，时间复杂度等

实验一：

得到父结点列表O(nlogn)

实验二：

初次查找O(nlogn)

之后查找O(1)

实验三：

建立字典树O(nlogn)

查询字典树O(nlogn)

实验四：

计算逆序O(n)

排序O(nlogn)

五、测试结果

输入数据和输出数据示例

实验一：

input.txt

12

3 2 3 4

2 5 6

0

0

2 7 8

2 9 10

0

0

0

2 11 12

0

0

5

3 11

7 12

4 8

9 12

8 10

output.txt

3 11 1

7 12 2

4 8 1

9 12 6

8 10 2

实验二：

input.txt

8 7

1 3

1 6

2 8

3 7

4 5

5 6

5 8

output.txt

Yes

11001010

实验三：

input.txt

3

Ae jd opq

9

Ae jd jda opq jda ae ld jd opq

output.txt

2

实验四：

input.txt

10 6

AACATGAAGG

TTTTGGCCAA

TTTGGCCAAA

GATCAGATTT

CCCGGGGGGA

ATCGATGCAT

0 0

output.txt

CCCGGGGGGA

AACATGAAGG

GATCAGATTT

ATCGATGCAT

TTTTGGCCAA

TTTGGCCAAA

六、说明

编程语言： C++语言