**算法分析与设计课程设计报告书 评分：\_\_\_\_\_\_\_\_\_\_\_\_**

**题目：基于Ullmann算法的图同构识别**

设计人：何树深

班级：13网络2班 学号：1314080903212

1. 实验环境和地点：

1、硬件环境：个人机，CPU主频：3.1GHz 内存：8G

2、软件环境：操作系统：ubuntu

编程语言：C++

3、地点：5号楼1楼智能信息处理实验室

目录

[一.实验任务解决方案： 4](#_Toc585010968)

[1、Ullmann算法的流程图。 4](#_Toc307748735)

[2、Ullmann算法实现的关键代码。 5](#_Toc1181997440)

[1). 构造矩阵: 5](#_Toc968237966)

[2).构造矩阵M的代码: 6](#_Toc1496667370)

[3). 深度优先搜索: 8](#_Toc1388080270)

[二. Ullmann算法的计算复杂度分析： 9](#_Toc1087364691)

[三. 利用图同构充要条件的算法: 10](#_Toc1918664024)

[1. 流程图 10](#_Toc1311640184)

[2.关键代码 10](#_Toc593304489)

[1). 求各顶点的度 10](#_Toc39706925)

[2).邻接矩阵的秩 11](#_Toc21389241)

*[3.](#_Toc375434438)* [算法的计算复杂度分析: 13](#_Toc375434438)

[四. 总结综合设计心得体会： 13](#_Toc2118438828)

[五. 附录 14](#_Toc663585855)

[1. 自己写的算法的全部源码 14](#_Toc264234746)

[1). Main.c 14](#_Toc808552111)

[2).graph\_malloc.c 17](#_Toc2025934884)

[3).graph\_input.c 19](#_Toc719597262)

[4). graph\_degree.c 21](#_Toc2112266509)

[5). Graph\_rand.c 22](#_Toc333058260)

[6). Graph\_free.c 24](#_Toc683765155)

注: 这个课程设计包含了两份算法, 一份是github上的Ullmann算法, 另一份是自己写的算法.

# 一.实验任务解决方案：

## 1、Ullmann算法的流程图。

开始

结束

输出结果

进行深度优先搜索

构造矩阵M

构造图B的邻接矩阵

构造图A的邻接矩阵

输入图B的数据

输入图A的数据

## 2、Ullmann算法实现的关键代码。

### 1). 构造矩阵:

A.

void Isomorphism::build\_matrix(Matrix<bool>& matrix, const Graph& graph)

{

matrix.clear();

matrix.resize(graph.size(), graph.size());

for (size\_t i = 0; i < graph.size(); ++i) {

const struct vertex\_t& vertex = graph.get\_vertex(i);

size\_t from\_id = vertex.id;

for (size\_t j = 0; j < vertex.edges.size(); ++j) {

size\_t to\_id = vertex.edges[j].to;

matrix.set(from\_id, to\_id, 1);

}

}

}

}//namespace ullman

B.

void Isomorphism::build\_matrix(Matrix<bool>& matrix, size\_t nrows, size\_t ncolumns)

{

matrix.clear();

matrix.resize(nrows, ncolumns);

for (size\_t i = 0; i < \_m\_columns\_used.size(); ++i) {

if (\_m\_columns\_used[i] != -1)

matrix.set(\_m\_columns\_used[i], i, 1);

}

}

### 2).构造矩阵M的代码:

bool Isomorphism::construct\_match(Graph& query\_graph, Graph& entry\_graph)

{

//prune : the size

if (query\_graph.size() > entry\_graph.size())

return false;

const std::map<struct edge\_label\_list\_t, std::vector<struct edge\_t \*> >& query\_list =

query\_graph.get\_edge\_label\_map();

std::map<struct edge\_label\_list\_t, std::vector<struct edge\_t \*> >::const\_iterator query\_list\_it =

query\_list.begin();

const std::map<struct edge\_label\_list\_t, std::vector<struct edge\_t \*> >& entry\_list =

entry\_graph.get\_edge\_label\_map();

std::map<struct edge\_label\_list\_t, std::vector<struct edge\_t \*> >::const\_iterator entry\_list\_it =

entry\_list.begin();

while (query\_list\_it != query\_list.end() && entry\_list\_it != entry\_list.end()) {

if (query\_list\_it->first < entry\_list\_it->first) {

++query\_list\_it;

} else if (entry\_list\_it->first < query\_list\_it->first) {

++entry\_list\_it;

} else {

for (size\_t i = 0; i < (query\_list\_it->second).size(); ++i) {

struct edge\_t \*query\_edge = (query\_list\_it->second)[i];

size\_t query\_from = query\_edge->from;

size\_t query\_to = query\_edge->to;

for (size\_t j = 0; j < (entry\_list\_it->second).size(); ++j) {

struct edge\_t \*entry\_edge = (entry\_list\_it->second)[j];

size\_t entry\_from = entry\_edge->from;

size\_t entry\_to = entry\_edge->to;

\_m\_columns.set(query\_from, entry\_from, 1);

\_m\_columns.set(query\_to, entry\_to, 1);

}

}

++entry\_list\_it;

++query\_list\_it;

}

}

//prune : degree

for (size\_t i = 0; i < \_m\_columns.size().first; ++i) {

for (size\_t j = 0; j < \_m\_columns.size().second; ++j) {

if (\_m\_columns.get(i, j) == 1) {

size\_t degree\_from = query\_graph.get\_vertex(i).edges.size();

size\_t degree\_to = entry\_graph.get\_vertex(j).edges.size();

if (degree\_to < degree\_from)

\_m\_columns.set(i, j, 0);

}

}

}

//prune : refine

refine(\_m\_columns, 0);

#ifdef DEBUG

//printf("after refine\n%s\n", \_m\_columns.c\_str());

#endif

//prune : the mapping of zeros

for (size\_t i = 0; i < \_m\_columns.size().first; ++i) {

bool find = false;

for (size\_t j = 0; j < \_m\_columns.size().second; ++j) {

if (\_m\_columns.get(i, j) == 1) {

find = true;

break;

}

}

if (!find) {

return false;

}

}

return true;

}

### 3). 深度优先搜索:

void Isomorphism::dfs\_search(size\_t idx, Matrix<bool> matrix)

{

//prune 3: if only one of the mapping has found

if (\_m\_cur\_find)

return;

#ifdef DEBUG

//printf("%s\n", matrix.c\_str());

#endif

if (idx == matrix.size().first) {

Matrix<bool> matrix\_c = matrix \* ((matrix \* matrix\_b).transposition());

#ifdef DEBUG

#endif

if (judge(matrix\_c)) {

\_m\_cur\_find = true;

\_m\_output[\_m\_cur\_query\_id].push\_back(\_m\_cur\_graph\_id);

}

return ;

}

refine(matrix, idx);

#ifdef DEBUG

//printf("after refine\n%s\n", matrix.c\_str());

#endif

for (size\_t i = 0; i < matrix.size().second; ++i) {

if (\_m\_columns\_used[i] != -1 || !matrix.get(idx, i))

continue;

Matrix<bool> next\_matrix = matrix;

next\_matrix.fill(idx, 0);

next\_matrix.set(idx, i, 1);

\_m\_columns\_used[i] = idx;

dfs\_search(idx + 1, next\_matrix);

\_m\_columns\_used[i] = -1;

}

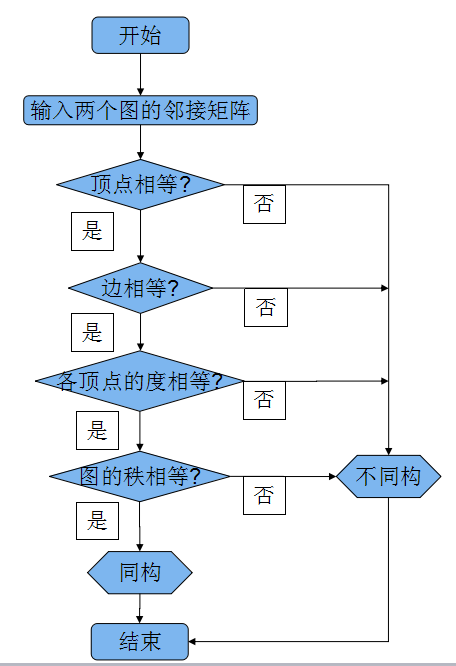
}

# 二. Ullmann算法的计算复杂度分析：

1. . 在最坏情况下, Ullmann算法的时间复杂度是: O(n^4).
2. . 算法中, 剪枝是影响算法性能的重要因素.
3. . Ullmann算法以深度优先的方式进行搜索, 搜索过程表示为一个bool矩 阵. 当节点不匹配时, 回溯到最近匹配的节点, 寻找其他方向.
4. . 算法会检查匹配点对的邻接点的匹配情况, 尽早地识别出不匹配的节点, 提高算法效率.

# 三. 利用图同构充要条件的算法:

## 1. 流程图



## 2.关键代码

### 1). 求各顶点的度

#include "graph\_degree.h"

static int

compare(const void \*a, const void \*b)

{

return \*(int \*)b - \*(int \*)a;

}

void graph\_degree(int \*\*graph, int \*degree, int v\_num)

{

//求度

int i = 0;

int j = 0;

int degree\_num = 0;

int \*\*temp\_graph = graph;

int \*temp\_degree = degree;

for (i = 0; i < v\_num; i++)

{

for (j = 0; j < v\_num; j++)

{

if (temp\_graph[i][j] == 1)

{

degree\_num++;

}

}

temp\_degree[i] = degree\_num;

degree\_num =0;

}

//排序

qsort(temp\_degree, v\_num, sizeof(int), compare);

/\* //试试输出

printf("测试一下排序后的度: \n");

for (i = 0; i < v\_num; i++)

{

printf("%d ", temp\_degree[i]);

}

printf("\n");

\*/

}

### 2).邻接矩阵的秩

#include "graph\_rand.h"

int graph\_rand(int \*\*graph, int v\_num)

{

int \*\*temp\_graph = graph; //注意这里!

int i = 0;

int j = 0;

int k = 0;

int flag = 0;

int flag2 = 0;

int rand\_num = v\_num; //存图的秩

//计算相等且不为0的行

//计算前n行中为0的行

for (i = 0; i < v\_num-1; i++)

{

for (j = i+1; j < v\_num; j++)

{

for (k = 0; k < v\_num; k++)

{

if (temp\_graph[i][k] != temp\_graph[j][k])

{

flag = 1;

break;

}

//用于判断为0的行

if (temp\_graph[i][k] > 0)

{

flag2 = 1;

}

}

//行不全为0, 但相同

if ((flag == 0) && (flag2 == 1) &&(k == v\_num))//这里k要注意!

{

rand\_num--; //行相同,秩减一

}

else if ((flag2 == 0) &&(k == v\_num)) //整行为0

{

rand\_num--; //空行

}

// printf("shishi %d\n", rand\_num);

flag = 0;

flag2 = 0;

}

}

//判断图的最后一行是否全为0

flag2 = 0;

for (k = 0; k < v\_num; k++)

{

if(temp\_graph[i][k] > 0)

{

flag2 = 1;

}

}

if (flag2 == 0 && k == v\_num)

{

rand\_num--;

}

// printf("秩为:%d\n", rand\_num);

return rand\_num;

}

## 算法的计算复杂度分析:

1. . 时间复杂度: O(n^3);

计算领结矩阵的秩用到的3重循环.

# 四. 总结综合设计心得体会：

这个课程设计好难,

刚开始,把别人github上的算法打印下来, 看了好久都不懂.

百度上资料很少, google上才有一些, 才查了好些资料才开始看懂一点点.

后来无意中百度了一下”图同构的充要条件”, 看到了一个感觉自己能实现的算法, 就自己实现了.

# 五. 附录

## 1. 自己写的算法的全部源码

### 1). Main.c

/\*==============================================================================\

\* Copyright(C)2015 Chudai.

\*

\* File name : main.c

\* Version : v1.0.0

\* Author : 初代

\* Date : 2015/11/22

\* Description :

\* Function list: 1.

\* 2.

\* 3.

\* History :

\\*==============================================================================\*/

#include "main.h"

int main(void)

{

int \*\*graph\_a = NULL; //指向a图

int \*\*graph\_b = NULL; //指向b图

int \*ga\_degree = NULL; //指向"存储a图的度的数组"

int \*gb\_degree = NULL; //指向"存储b图的度的数组"

int va\_num = 0; //a图的顶点个数

int vb\_num = 0; //b图的顶点个数

int ea\_num = 0; //a图的边数

int eb\_num = 0; //b图的边数

int ga\_rand = 0; //a图的秩

int gb\_rand = 0; //b图的秩

int i = 0;

printf("-----本程序判断两个无向图是否同构-----\n");

printf("请输入第一个图的顶点数: ");

scanf("%d", &va\_num);

printf("请输入第二个图的顶点数: ");

scanf("%d", &vb\_num);

if (va\_num != vb\_num)

{

printf("不同构! 顶点数不相等!\n");

return 0;

}

/\*

////////////////////////试试放在函数里//////////////

//分配内存

//第一个图

graph\_a = (int \*\*)malloc(va\_num \* sizeof(int \*));

for (i = 0; i < va\_num; i++)

{

graph\_a[i] = (int \*) malloc(va\_num \* sizeof(int));

}

//第一个图的度

ga\_degree = (int \*)malloc(va\_num \* sizeof(int));

//分配第二个图

graph\_b = (int \*\*)malloc(vb\_num \* sizeof(int \*));

for (i = 0; i < vb\_num; i++)

{

graph\_b[i] = (int \*) malloc(vb\_num \* sizeof(int));

}

//第二图的度

gb\_degree = (int \*) malloc(vb\_num \*sizeof(int));

\*/

graph\_a = graph\_malloc(va\_num);

ga\_degree = degree\_malloc(va\_num);

graph\_b = graph\_malloc(vb\_num);

gb\_degree = degree\_malloc(vb\_num);

//输入数据(邻接矩阵) & 检查是否输入有误

printf("输入第一个图的邻接矩阵:\n");

ea\_num = graph\_input(graph\_a, va\_num);

printf("输入第二个图的邻接矩阵:\n");

eb\_num = graph\_input(graph\_b, vb\_num);

//判断边数是否相等

if(ea\_num != eb\_num)

{

printf("不同构! 边数不相等!\n");

return 0;

}

//求两图每个点的度, 排序后判断每个顶点的度

//ga\_degree 是值-结果参数

graph\_degree(graph\_a, ga\_degree, va\_num);

graph\_degree(graph\_b, gb\_degree, vb\_num);

//判断

for (i = 0; i < va\_num; i++)

{

if (ga\_degree[i] != gb\_degree[i])

{

printf("不同构! 各个顶点的度不对应!\n");

return 0;

}

}

//求矩阵的秩

ga\_rand = graph\_rand(graph\_a, va\_num);

gb\_rand = graph\_rand(graph\_b, vb\_num);

//判断矩阵的4是否相等

if (ga\_rand != gb\_rand)

{

// printf("%d %d\n", ga\_rand, gb\_rand);

printf("不同构! 秩不相等!\n");

}

else

{

printf("同构!\n");

}

/\*

//释放内存

for (i = 0; i < va\_num; i++)

{

free(graph\_a[i]);

}

free(graph\_a);

free(ga\_degree);

for (i = 0; i < vb\_num; i++)

{

free(graph\_b[i]);

}

free(graph\_b);

free(gb\_degree);

\*/

graph\_free(graph\_a, ga\_degree, va\_num);

graph\_free(graph\_b, gb\_degree, vb\_num);

return 0;

}

### 2).graph\_malloc.c

/\*==============================================================================\

\* Copyright(C)2015 Chudai.

\*

\* File name : graph\_malloc.c

\* Version : v1.0.0

\* Author : 初代

\* Date : 2015/11/23

\* Description :

\* Function list: 1.

\* 2.

\* 3.

\* History :

\\*==============================================================================\*/

#include "graph\_malloc.h"

//分配图的内存

int \*\*graph\_malloc(int v\_num)

{

int i = 0;

int \*\*temp\_graph = NULL;

//分配图

temp\_graph = (int \*\*) malloc(v\_num \*sizeof(int \*));

if (temp\_graph == NULL)

{

printf("图内存分配错误: 一!\n");

exit(1);

}

for (i = 0; i < v\_num; i++)

{

temp\_graph[i] = (int \*) malloc(v\_num \* sizeof(int));

if (temp\_graph[i] == NULL)

{

printf("图内存分配错误: 二!\n");

exit(1);

}

}

return temp\_graph;

}

//分配度的内存

int \*degree\_malloc(int v\_num)

{

int \*temp\_degree = NULL;

//分配度

temp\_degree = (int \*)malloc(v\_num \*sizeof(int));

if (temp\_degree == NULL)

{

printf("度内存分配错误!\n");

exit(1);

}

return temp\_degree;

// graph = temp\_graph;

// degree = temp\_degree;

}

### 3).graph\_input.c

/\*==============================================================================\

\* Copyright(C)2015 Chudai.

\*

\* File name : graph\_input.c

\* Version : v1.0.0

\* Author : 初代

\* Date : 2015/11/22

\* Description :

\* Function list: 1.

\* 2.

\* 3.

\* History :

\\*==============================================================================\*/

#include "graph\_input.h"

int graph\_input(int \*\*graph, int v\_num)

{

int i = 0;

int j = 0;

int edge\_num = 0;

int \*\*temp\_graph = graph;

//输入数据

for (i = 0; i < v\_num; i++)

{

for (j = 0; j < v\_num; j++)

{

scanf("%d", &temp\_graph[i][j]);

if (temp\_graph[i][j] != 0 && temp\_graph[i][j] != 1)

{

printf("输入数据不为0或1!\n");

exit(1);

}

if (temp\_graph[i][j] == 1)

{

edge\_num++; //再想想!

}

}

}

//判断输入是否有错

for (i = 0; i < v\_num; i++)

{

for (j = 0; j < v\_num; j++)

{

if(temp\_graph[i][j] != temp\_graph[j][i])

{

printf("输入的图有错!\n");

exit(1);

}

}

}

/\*

/////debug

printf("\n");

for (i = 0; i < v\_num; i++)

{

for (j = 0; j < v\_num; j++)

{

printf("%d ", temp\_graph[i][j]);

}

printf("\n");

}

\*/

edge\_num /= 2; ////

return edge\_num;

}

### 4). graph\_degree.c

/\*==============================================================================\

\* Copyright(C)2015 Chudai.

\*

\* File name : graph\_degree.c

\* Version : v1.0.0

\* Author : 初代

\* Date : 2015/11/22

\* Description :

\* Function list: 1.

\* 2.

\* 3.

\* History :

\\*==============================================================================\*/

#include "graph\_degree.h"

static int

compare(const void \*a, const void \*b)

{

return \*(int \*)b - \*(int \*)a;

}

void graph\_degree(int \*\*graph, int \*degree, int v\_num)

{

//求度

int i = 0;

int j = 0;

int degree\_num = 0;

int \*\*temp\_graph = graph;

int \*temp\_degree = degree;

for (i = 0; i < v\_num; i++)

{

for (j = 0; j < v\_num; j++)

{

if (temp\_graph[i][j] == 1)

{

degree\_num++;

}

}

temp\_degree[i] = degree\_num;

degree\_num =0;

}

//排序

qsort(temp\_degree, v\_num, sizeof(int), compare);

/\* //试试输出

printf("测试一下排序后的度: \n");

for (i = 0; i < v\_num; i++)

{

printf("%d ", temp\_degree[i]);

}

printf("\n");

\*/

}

### 5). Graph\_rand.c

/\*==============================================================================\

\* Copyright(C)2015 Chudai.

\*

\* File name : graph\_rand.c

\* Version : v1.0.0

\* Author : 初代

\* Date : 2015/11/22

\* Description :

\* Function list: 1.

\* 2.

\* 3.

\* History :

\\*==============================================================================\*/

#include "graph\_rand.h"

int graph\_rand(int \*\*graph, int v\_num)

{

int \*\*temp\_graph = graph; //注意这里!

int i = 0;

int j = 0;

int k = 0;

int flag = 0;

int flag2 = 0;

int rand\_num = v\_num; //存图的秩

//计算相等且不为0的行

//计算前n行中为0的行

for (i = 0; i < v\_num-1; i++)

{

for (j = i+1; j < v\_num; j++)

{

for (k = 0; k < v\_num; k++)

{

if (temp\_graph[i][k] != temp\_graph[j][k])

{

flag = 1;

break;

}

//用于判断为0的行

if (temp\_graph[i][k] > 0)

{

flag2 = 1;

}

}

//行不全为0, 但相同

if ((flag == 0) && (flag2 == 1) &&(k == v\_num))//这里k要注意!

{

rand\_num--; //行相同,秩减一

}

else if ((flag2 == 0) &&(k == v\_num)) //整行为0

{

rand\_num--; //空行

}

// printf("shishi %d\n", rand\_num);

flag = 0;

flag2 = 0;

}

}

//判断图的最后一行是否全为0

flag2 = 0;

for (k = 0; k < v\_num; k++)

{

if(temp\_graph[i][k] > 0)

{

flag2 = 1;

}

}

if (flag2 == 0 && k == v\_num)

{

rand\_num--;

}

// printf("秩为:%d\n", rand\_num);

return rand\_num;

}

### 6). Graph\_free.c

/\*==============================================================================\

\* Copyright(C)2015 Chudai.

\*

\* File name : graph\_free.c

\* Version : v1.0.0

\* Author : 初代

\* Date : 2015/11/23

\* Description :

\* Function list: 1.

\* 2.

\* 3.

\* History :

\\*==============================================================================\*/

#include "graph\_free.h"

void graph\_free(int \*\*graph, int \*degree, int v\_num)

{

int i = 0;

//释放图

for (i = 0; i < v\_num; i++)

{

free(graph[i]);

}

free(graph);

//释放度

free(degree);

}