

**实 验 报 告**

**（2019 / 2020 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 离散数学 | | | | | |
| 实验名称 | 图的生成及欧拉回路的确定 | | | | | |
| 实验时间 | 2021 | 年 | 10 | 月 | 5 | 日 |
| 指导单位 | 计算机学院计算机科学与技术系 | | | | | |
| 指导教师 |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 |  | 班级学号 |  |
| 学院(系) | 计算机学院 | 专 业 |  |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | **图的生成及欧拉回路的确定** | | | **指导教师** |  |
| **实验类型** | **验证** | **实验学时** | **4** | **实验时间** | 2021-10-5 |
| 1. **实验目的和要求**   对给定n个节点的无向图，进行欧拉图与半欧拉图的判定，若是欧拉图或半欧拉图，则输出所有的欧拉回路。 | | | | | |
| 二、**实验环境(实验设备)**  硬件：微型计算机  软件：Fedora Linux 34操作系统, gcc 11.2.1编译器，代码编辑器等 | | | | | |
| **三、实验原理及内容**  1、图的生成：图的存储采用动态分配的邻接矩阵。由于是无向图，只需要下三角与对角线。定义一个查询函数，给出图与对应位置，获得/修改那个地方的连接状态。函数内部需要判断位置的合法性，如(1,2) (2,1)代表同一个位置，而(-1, -1)是非法的。  在此基础上添加由字符到数字位置的映射。无论节点叫做什么，都能以O(1)的复杂度获得它对应的位置。一对字符代表这两个节点之间有连接。  2、图的可视化：利用外部工具Graphviz图描述语言可以把字符转化为图片格式。程序输出图时把它输出为符合格式的语言。原图以无向图输出并添加是否是欧拉图的标注，欧拉回路以有向图输出，并在箭头上标注数字，可视化一笔画的画法。这样，即使不理解邻接矩阵，也可以通过输入字符创建图并生成可视化的一笔画方案。  3、欧拉路径的确：采用递归逐步深入的方法。在递归函数外建立一个堆栈，大小是图边的数目+1，里面存放到此次调用之前所有访问的节点。递归函数开始先把当前节点入栈，如果栈满，那么栈里的元素就是一笔画的路径。否则标注当前边为已使用，对所有当前节点有连接的边调用本函数，达到层层递进的效果。再把当前节点出栈，取消标注当前边为已使用。该函数被封装在一个源文件内，供另一个接口函数调用。这个函数对每一个可能是路径起始点的节点调用递归函数，返回存有递归函数获取的路径的结构指针。  程序的源代码及注释如下：  eularPath.h  #pragma once  #ifdef \_\_cplusplus  #error C CODE, NO CXX PERMITTED  #else  #include <stdbool.h>  #include <stdint.h>  // 用于实现节点名称和对应下标的O(1)转换  typedef struct AscMapper {  int8\_t ascToLbl[128];  char lblToAsc[128];  } AscMapper;  // 初始化映射器  AscMapper\* initMapper(AscMapper\* this, const char\* src);  // 存储无向图基本信息  typedef struct UndirGraphInfo {  bool \*\*adjMat; // 临接矩阵  uint8\_t \*degrees; // 每个节点的度数  unsigned edgeSize, nodeSize; // 边数、节点数  } GraphInfo;  // 存储一个无向图所有欧拉路径的信息  typedef struct EularData {  unsigned resSize, resMax, stepSize; // 结果数目与最大结果数目  uint8\_t\* stepStk; //临时数组，存储获取欧拉路径用到的栈的数据  uint8\_t\*\* results; // 存储所有结果  } EularData;  // 存储图信息函数的相关声明  GraphInfo\* initGraph(GraphInfo\* this, unsigned size);  void addLinks(GraphInfo\* this, const AscMapper\* map,  const char\* src1, const char\* src2);  void addLinksAlternate(GraphInfo \*this, const AscMapper \*map, const char \*src);  bool toggleLink(GraphInfo\* this, uint8\_t pos1, uint8\_t pos2);  bool hasLink(const GraphInfo \*this, uint8\_t pos1, uint8\_t pos2);  void freeGraph(GraphInfo \*this);  // 获取欧拉回路函数的相关声明  EularData\* getEular(const GraphInfo\* this, unsigned resMax);  void freeEular(EularData \*data);  void printGraph(const GraphInfo \*this, const AscMapper\* map,  const EularData\* data);  #endif  main.c  #include "eularPath.h"  #include <stdio.h>  #include <string.h>  #include <stdlib.h>  // 用于解析命令行参数选择合适模式初始化图和相关结构  int eulArg(int argc, char\*\* argv, AscMapper\* map, GraphInfo\* g,  unsigned\* maxRes) {  \*maxRes = 50;  switch (argc) {  case 3: \*maxRes = atoi(argv[2]); //[[fallthrough]]  case 2:  initMapper(map, argv[1]);  initGraph(g, strlen(map->lblToAsc));  addLinksAlternate(g, map, argv[1]);  break;  case 5: \*maxRes = atoi(argv[4]); //[[fallthrough]]  case 4:  initMapper(map, argv[1]);  initGraph(g, strlen(argv[1]));  addLinks(g, map, argv[2], argv[3]);  break;  default:  printf("Usage: %s ALLNODES FROMNODES TONODES [MAXRES]\n", argv[0]);  printf("Or: %s LINKPAIRS [MAXRES]", argv[0]);  return 1; //命令行输入错误返回1  }  return 0; //正确返回0  }  int main(int argc, char\*\* argv) {  unsigned maxSize; AscMapper myMap; GraphInfo myGraph;  if (eulArg(argc, argv, &myMap, &myGraph, &maxSize))  return 1; // 输入错误，退出程序  EularData\* dat = getEular(&myGraph, maxSize); //获取欧拉图信息  printGraph(&myGraph, &myMap, dat); // 输出原图与欧拉图  freeGraph(&myGraph);  if (dat) freeEular(dat); //若非欧拉图信息结构为空，需检验空值  return 0;  }  graphStore.c  #include "eularPath.h"  #include <stdio.h>  #include <memory.h>  #include <stdlib.h>  #include <assert.h>  #pragma GCC diagnostic ignored "-Wchar-subscripts"  // 初始化字符编号映射，src的字符作为节点名称  AscMapper \*initMapper(AscMapper \*this, const char \*src) {  unsigned sz = 0;  memset(this->lblToAsc, -1, 128);  memset(this->ascToLbl, -1, 128);  while (\*src) {  if (this->ascToLbl[\*src] == -1) {  this->ascToLbl[\*src] = sz;  this->lblToAsc[sz++] = \*src;  }  src++;  }  this->lblToAsc[sz] = '\0'; // lblToAsc按顺序为整个图的节点  return this;  }  // 初始化大小为size \* size的图  GraphInfo \*initGraph(GraphInfo \*this, unsigned size) {  this->nodeSize = size; this->edgeSize = 0;  this->degrees = calloc(this->nodeSize, sizeof(uint8\_t));  this->adjMat = calloc(this->nodeSize, sizeof(bool\*));  for (int i = 0; i < this->nodeSize; i++)  // 由于是无向图，只存储下三角与对角线，第i行动态数组长度i+1  this->adjMat[i] = calloc(i + 1, sizeof(bool));  return this;  }  // 改变pos1行pos2列的连接状态  bool toggleLink(GraphInfo \*this, uint8\_t pos1, uint8\_t pos2) {  // 先判断是否越界  assert(pos1 < this->nodeSize && pos2 < this->nodeSize);  // 由于只有下三角，若行大于列交换行列下标  bool\* item = pos1 < pos2 ?  &(this->adjMat[pos2][pos1]) : &(this->adjMat[pos1][pos2]);  if (\*item) // 未连接则连上，有连接则断开  this->degrees[pos1]--, this->degrees[pos2]--, this->edgeSize--;  // 改变对应节点度数，加减图的边数  else this->degrees[pos1]++, this->degrees[pos2]++, this->edgeSize++;  return \*item = !(\*item);  }  // 获取节点pos1 pos2之间是否有连接  bool hasLink(const GraphInfo \*this, uint8\_t pos1, uint8\_t pos2) {  assert(pos1 < this->nodeSize && pos2 < this->nodeSize); // 先判断是否越界  // 由于只有下三角，若行大于列交换行列下标  return pos1 < pos2 ? this->adjMat[pos2][pos1] : this->adjMat[pos1][pos2];  }  // 为一个图添加多个边，节点为src1 src2里的每一个字符  void addLinks(GraphInfo \*this, const AscMapper \*map,  const char \*src1, const char \*src2) {  while (\*src1 && \*src2)  toggleLink(this, map->ascToLbl[\*src1++], map->ascToLbl[\*src2++]);  }  // 交替添加src里的字符  void addLinksAlternate(GraphInfo \*this, const AscMapper \*map, const char \*src) {  char from[256] = "", to[256] = ""; uint8\_t i = 0;  while (src[i])  if (i & 1) to[i >> 1] = src[i], i++;  else from[i >> 1] = src[i], i++;  addLinks(this, map, from, to);  }  // 释放图的内存  void freeGraph(GraphInfo \*this) {  free(this->degrees);  for (unsigned i = 0; i < this->nodeSize; i++)  free(this->adjMat[i]); // 先释放矩阵  free(this->adjMat); // 再释放指向矩阵的指针  }  getEular.c  #include "eularPath.h"  #include <stdio.h>  #include <stdlib.h>  #include <memory.h>  // 打印图以及所有的欧拉回路（如果存在）  // 格式是Graphviz图格式，便于可视化输出  void printGraph(const GraphInfo \*this, const AscMapper\* map,  const EularData\* data) {  if (!data || data->resSize == 0) // 不是欧拉图，图下方显示“不是欧拉图”  puts("graph original {\n\tlabel=\"Not an Eularian Graph!\"");  // 否则显示“是欧拉图”  else puts("graph original {\n\tlabel=\"Eularian Graph!\"");  for (unsigned r = 0; r < this->nodeSize; r++)  for (unsigned j = 0; j <= r; j++)  if (hasLink(this, r, j)) // 如果有连接，就输出  printf("\t%c -- %c;\n", map->lblToAsc[j], map->lblToAsc[r]);  puts("}");  if (!data || data->resSize == 0) return; // 不是欧拉图，到此为止  for (unsigned i = 0; i < data->resSize; i++) {  unsigned step = 0; // 用于标识当前步数，输出的图可以看到具体走法  printf("digraph solution%d {\n\tlabel=\"Solution %d\"\n", i, i);  for (unsigned j = 0; j < this->edgeSize; j++)  // 按照每一个res里记录的点走的路径是欧拉回路  printf("\t%c -> %c [label=\"%d\"];\n", // 画有向图箭头  map->lblToAsc[data->results[i][j]], // 标注步数编号  map->lblToAsc[data->results[i][j + 1]], ++step);  puts("}");  }  }  #pragma GCC diagnostic push  #pragma GCC diagnostic ignored "-Wdiscarded-qualifiers"  // 获取欧拉路径的主体递归函数。  void getEularImpl(EularData \*dat, const GraphInfo\* g, unsigned nodeLbl) {  for (unsigned i = 0; i < g->nodeSize && dat->resSize != dat->resMax; i++)  /\* 对于每个点，如果有连接，则走入这个点  \* 走入、走出一个点前后，图的内容是不变的，因此声明\*g为常量  \* 但是在过程中，需要改变图的内容，把尝试走的路径标为无连接  \* 走出去后，会把图的内容恢复，因此需要暂时去除常量标识符 \*/  if (hasLink(g, i, nodeLbl)) {  GraphInfo \*constCast = g; // 把图中走过的边去掉  toggleLink(constCast, i, nodeLbl); constCast->edgeSize++;  dat->stepStk[dat->stepSize++] = i; // 当前正在走的点压入栈  if (dat->stepSize == g->edgeSize + 1) // 判断是否所有边都走过  // 入栈的点是行走顺序，把它复制到对应结果数组里  dat->results[dat->resSize++] = memcpy(malloc(sizeof(uint8\_t) \* (  g->edgeSize + 1)), dat->stepStk, sizeof(uint8\_t) \* (g->edgeSize + 1));  else getEularImpl(dat, g, i); // 未完成，再以下一个点为起点，重复调用此函数继续下一步  dat->stepSize--; // 这个点走完了，出栈，把去掉的边恢复  toggleLink(constCast, i, nodeLbl); constCast->edgeSize--;  }  }  #pragma GCC diagnostic pop  // 用户调用的获取欧拉回路的函数。图g在进出函数前后无变化，返回数据写入的指针。  // 如果不是（半）欧拉图，返回空指针。  EularData \*getEular(const GraphInfo \*this, unsigned int resMax) {  unsigned oddDeg = 0;  for (unsigned i = 0; i < this->nodeSize; i++)  if (this->degrees[i] & 1)  oddDeg++;  if (oddDeg != 0 && oddDeg != 2) // 统计度为奇数节点个数  return NULL; // 度为奇数节点个数不是0或2,无欧拉路径  EularData \*dat = malloc(sizeof(EularData)); // 有欧拉路径才分配空间  dat->stepStk = calloc(sizeof(uint8\_t), this->edgeSize + 1);  dat->results = calloc(sizeof(uint8\_t\*), resMax);  dat->resMax = resMax, dat->resSize = 0, dat->stepSize = 1;  if (oddDeg == 0) { // 全欧拉图，每一个点都有欧拉回路  for (unsigned i = 0; i < this->nodeSize; i++)  // 走入栈的栈底元素是第一个点i，提前赋值  dat->stepStk[0] = i, getEularImpl(dat, this, i);  return dat;  }  for (unsigned i = 0; i < this->nodeSize; i++)  if (this->degrees[i] & 1) // 半欧拉图，只有度为鸡的点有欧拉路径  dat->stepStk[0] = i, getEularImpl(dat, this, i);  return dat;  }  // 释放欧拉路径记录的空间  void freeEular(EularData \*data) {  free(data->stepStk);  for (unsigned i = 0; i < data->resSize; i++)  free(data->results[i]);  free(data->results);  free(data);  }  Makefile  CFLAGS = -O2 -Wall -Werror  VISDIR = vis  SRC = $(wildcard \*.c)  OBJ = $(SRC:.c=.o)  BIN = Exp4  # 默认图的参数（即右图）  ARGS = abcdefgh aabbcddeefg bdceefgghgh 128  %.o: %.c  $(CC) $(CFLAGS) -c $^ -o $@  $(BIN): $(OBJ)  $(CC) $(CFLAGS) $^ -o $@  # 生成并可视化图的所有欧拉回路，保存为svg矢量图  vis: $(BIN) cleanVis  mkdir -p $(VISDIR)  cd $(VISDIR); ../$(BIN) $(ARGS) | dot -Tsvg -O  mv $(VISDIR)/noname.gv.svg $(VISDIR)/Original.svg  cd $(VISDIR); rename 'noname.gv.' '' \* >/dev/null 2>&1  # 清理所有东西  clean: cleanBin cleanVis  # 清理生成的可执行文件  cleanBin:  $(RM) $(BIN) $(OBJ)  # 清理欧拉回路矢量图  cleanVis:  $(RM) -r $(VISDIR)  all: $(BIN)  运行可视化命令make vis,会自动生成前页的默认图的80个欧拉路径，标记为0～79。可以通过改变命令行参数make vis ARGS=...获取其他图的欧拉路径。前面的图的最后一种画法如下： | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  (一)实验中遇到的主要问题及解决方法  写求欧拉路径的递归函数起初在函数内定义一个整数存储当前所在节点，但在找到路径保存的时候，由于每一个函数所占空间相互独立，难以互相访问。联想到函数是以栈的形式储存的，选择建立假栈数组，即占用在函数之外的堆空间模拟入栈出栈的功能，但仍然能随机访问读取。  （二）实验心得  数据结构实验里也有一道和图相关的问题，求取的是有向图里任意两点的最短距离。尽管图的种类不同、问题不同，但有相似的思路：每走一步，更新相邻节点的状态，从更新的状态里继续寻找。以后遇到图论问题，也能以这种方式思考。  在实验中利用外部工具GraphViz可视化生成的图。虽然图的输出麻烦了许多，可是一目了然的图像结果，让我体会到今后各种软件、工具相互适配交融的强大力量。  （三）意见与建议（没有可省略） | | | | | |
| 1. **支撑毕业要求指标点**   支撑毕业要求的指标点为：   * 1-4掌握计算机科学与技术领域的专业知识，能将专业知识用于分析和解决计算机领域复杂工程问题。   √   * 2-1能够应用数学、自然科学和工程科学的基本知识，识别和分析计算机领域复杂工程问题的特征。 | | | | | |
| **六、指导教师评语 (含学生能力达成度的评价)** | | | | | |
| **成 绩** |  | **批阅人** |  | **日 期** |  |

如果不太想写太多字，“指导教师评语”也可以设计为如下的各选择项用打勾形式（仅仅作为一个简单示例，请各课程负责人根据课程和实验情况以及支撑的指标点来自行设定选择项，同一门课程的不同实验评分细则项允许存在不同）：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **评 分 细 则** | **评分项** | **优秀** | **良好** | **中等** | **合格** | **不合格** |
| **遵守实验室规章制度** |  |  |  |  |  |
| **学习态度** |  |  |  |  |  |
| **算法思想准备情况** |  |  |  |  |  |
| **程序设计能力** |  |  |  |  |  |
| **解决问题能力** |  |  |  |  |  |
| **课题功能实现情况** |  |  |  |  |  |
| **算法设计合理性** |  |  |  |  |  |
| **算法效能评价** |  |  |  |  |  |
| **回答问题准确度** |  |  |  |  |  |
| **报告书写认真程度** |  |  |  |  |  |
| **内容详实程度** |  |  |  |  |  |
| **文字表达熟练程度** |  |  |  |  |  |
| **其它评价意见** |  | | | | |
| **本次实验能力达成评价（总成绩）** |  |  |  |  |  |