Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студентка: Е. А. Айрапетова
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №9

Задача: Разработать программу на языке C или C++, реализующую указанный алгоритм согласно заданию:

Задан неориентированный двудольный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти максимальное паросочетание в графе алгоритмом Куна. Для обеспечения однозначности ответа списки смежности графа следует предварительно отсортировать. Граф не содержит петель и кратных ребер.

1 Описание

Требуется написать алгоритм Куна для неориентированного двудольного графа.

Паросочетанием M называется набор попарно несмежных рёбер графа (иными словами, любой вершине графа должно быть инцидентно не более одного ребра из M).

Цепью длины k назовём некоторый простой путь (т.е. не содержащий повторяющихся вершин или рёбер), содержащий ровно k рёбер.

Чередующейся цепью относительно паросочетания назовём простой путь длины k в которой рёбра поочередно принадлежат/не принадлежат паросочетанию.

Увеличивающей цепью относительно паросочетания назовём чередующуюся цепь, у которой начальная и конечная вершины не принадлежат паросочетанию. [2].

Алгоритм Куна является эффективным алгоритмом для поиска максимального паросочетания. Он заключается в том, что мы будем искать увеличивающую цепь, пока ищется, и проводить чередование в ней, то есть убирать из паросочетания все рёбра, принадлежащие цепи, и, наоборот, добавлять все остальные.

2 Исходный код

Для хранения рёбер используется std::vector. Для того, чтобы данные были отсортированы, используем std::set.

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
#include <queue>

bool DFS(int v, std::set<int>& used, std::vector<std::vector<int>>& graph, std::vector
    <int>& matching) {
    if (used.count(v)) {
        return false;
    }
    used.insert(v);
    for (int& elem : graph[v]) {
        if (matching[elem] == -1 || DFS(matching[elem], used, graph, matching) {
            matching[elem] = v;
            return true;
        }
    }
    return false;
}

std::vector<std::pair<int, int>> KuhnAlgorithm(std::vector<std::vector<int>>& graph) {
    std::vector<int> matching(graph.size(), -1);
    std::set<int> used;
    for (int i = 0; i < graph.size(); ++i) {
        used.clear();
        DFS(i, used, graph, matching);
    }
    std::vector<std::pair<int, int>> answer;
    for (int i = 0; i < matching.size(); ++i) {
        if (matching[i] != -1) {
            answer.push_back(std::make_pair(std::min(i, matching[i]), std::max(i,
                matching[i])));
        }
    }
    std::sort(answer.begin(), answer.end(), [](std::pair<int, int> l, std::pair<int,
        int> r) { return l.first < r.first; });
    return answer;
}

std::vector<int> SplitForPart(std::vector<std::vector<int>>& graph) {
    std::vector<int> part(graph.size(), -1);
    std::vector<bool> used(graph.size(), false);
    std::queue<int> queue;
```

```
42      for (int i = 0; i < graph.size(); ++i) {
43          if (part[i] == -1) {
44              part[i] = 0;
45              queue.push(i);
46              used[i] = true;
47              while (!queue.empty()) {
48                  int curr = queue.front();
49                  queue.pop();
50                  int parent = curr;
51                  for (int j = 0; j < graph[curr].size(); ++j) {
52                      if (part[graph[curr][j]] == -1 && !used[graph[curr][j]]) {
53                          part[graph[curr][j]] = !part[parent];
54                          used[graph[curr][j]] = true;
55                          queue.push(graph[curr][j]);
56                      }
57                  }
58              }
59          }
60      }
61      return part;
62  }
63
64  int main() {
65      int n, m, begin, end;
66      std::cin >> n >> m;
67      std::vector<std::vector<int>> graph(n);
68      for (int i = 0; i < m; ++i) {
69          std::cin >> begin >> end;
70          graph[begin - 1].push_back(end - 1);
71          graph[end - 1].push_back(begin - 1);
72      }
73      std::vector<int> part = SplitForPart(graph);
74      std::vector<std::vector<int>> biGraph(graph.size());
75      for (size_t i = 0; i < graph.size(); ++i) {
76          if (!graph[i].empty())
77              std::sort(graph[i].begin(), graph[i].end());
78      }
79      for (int i = 0; i < graph.size(); ++i) {
80          if (!part[i]) {
81              biGraph[i] = graph[i];
82          }
83      }
84      std::vector<std::pair<int, int>> result = KuhnAlgorithm(biGraph);
85      std::cout << result.size() << '\n';
86      for (const std::pair<int, int>& pair : result) {
87          std::cout << pair.first + 1 << ' ' << pair.second + 1 << '\n';
88      }
89      return 0;
90  }
```

3 Консоль

```
jane@Evgenia:/mnt/c/Files/ДА/ЛР9/solution$ g++ lab9.cpp
jane@Evgenia:/mnt/c/Files/ДА/ЛР9/solution$ ./a.out
4 3
1 2
2 3
3 4
2
1 2
3 4
jane@Evgenia:/mnt/c/Files/ДА/ЛР9/solution$ ./a.out
5 4
1 2
2 3
1 3
4 5
2
1 2
4 5
```

4 Выводы

Выполнив девятую лабораторную работу по курсу «Дискретный анализ»,я изучила способы представления графов с помощью компьютера и реализовала алгоритм Куна. Также я освежила свои знания о графах из курса «Дискретной математики» и дополнила их.

Список литературы

[1]

Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн.

Алгоритмы: построение и анализ, 2-е издание. ---Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. ---1296 с. (ISBN 5-8459-0857-4 (рус.))

[2]

Паросочетания.
URL: https://algorithmica.org/ru/matching (дата обращения: 16.05.2021).