Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: Л. В. Ивенкова
Преподаватель: Н. С. Капралов
Группа: М8О-208Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №8

Задача: Топологическая сортировка

Вариант №6

Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Дан набор из **N** объектов. Каждому из объектов присвоен свой номер: **1, 2, 3** и т.д. Кроме того, заданы **N** условий с ограничиениями на расположение вида «**A** должен находиться перед **B**». Необходимо найти такой минимальный набор правил, что все остальные ограничения будут выполняться.

Формат входных данных

На первой строке два числа, **N** и **M**, за которыми следует **M** строк с ограничениями вида «$\mathbf{A} < \mathbf{B}$» ($1 \leq A, B \leq N$) определяющими относительную последовательность объектов с номерами **A** и **B**.

Формат результата

-1, если расположить объекты в соответствии с требованиями невозможно, искомый набор правил в противном случае.

1 Описание

Требуется написать реализацию топологической сортировки.

В общем случае задача на топологическую сортировку выглядит так:

"Дан ориентированный граф с n вершинами и m рёбрами. Требуется **перенумеровать** его вершины таким образом, чтобы каждое рёбро вело из вершины с меньшим номером в вершину с большим.

Иными словами, требуется найти перестановку вершин (**топологический порядок**), соответствующую порядку, задаваемому всеми рёбрами графа.

Топологическая сортировка может быть **не единственной** (например, если граф — пустой; или если есть три такие вершины a, b, c, что из a есть пути в b и в c, но ни из b в c, ни из c в b добраться нельзя).

Топологической сортировки может **не существовать вовсе** — если граф содержит циклы (поскольку при этом возникает противоречие: есть путь и из одной вершины в другую, и наоборот)."[1]

В нашем случае заданы n объектов, и m отношений между ними — что один объект стоит впереди другого (или же, по аналогии с числами, одна вершина имеет меньший номер). Требуется проверить, не противоречивы ли эти отношения, и если нет, выдать объекты в порядке их "возрастания"(если решений несколько — выдать любое). Легко заметить, что это в точности и есть задача о поиске топологической сортировки в графе из n вершин.

Для топологической сортировки используется обход в глубину.

"Предположим, что граф ацикличен, т.е. решение существует. Что делает обход в глубину? При запуске из какой-то вершины v он пытается запуститься вдоль всех рёбер, исходящих из v. Вдоль тех рёбер, концы которых уже были посещены ранее, он не проходит, а вдоль всех остальных — проходит и вызывает себя от их концов.

Таким образом, к моменту выхода из вызова $\mathrm{dfs}(v)$ все вершины, достижимые из v как непосредственно (по одному ребру), так и косвенно (по пути) — все такие вершины уже посещены обходом. Следовательно, если мы будем в момент выхода из $\mathrm{dfs}(v)$ добавлять нашу вершину в начало некоего списка, то в конце концов в этом списке получится топологическая сортировка."[1] И остаётся лишь сделать реверс этого списка.

Чтобы топологическая сортировка существовала, нужно проверить граф на надичие циклов. Это можно сделать с помощью обхода в глубину [2]. Для этого непосещённые

вершины красим в белый цвет, посещенные в чёрный, а вершины, из которых ещё может быть продолжен обход в глубину, серым. Если при обходе мы встретим серую вершину, то в графе присутсятвует цикл.

Подход топологической сортировки использует идею жадного алгоритма: мы не перебираем все возможные варианты перестановок даже между элементами на одинаковой глубине. Оптимальный выбор запоминать самые глубокие вершины, а затем менее глубокие, даёт оптимальное решение задачи в целом.

2 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>

using namespace std;

void DFS(int v, const vector<vector<int>>& G, vector<bool>& used, vector<int>& res) {

    used[v] = 1;
    for (int to: G[v]) {
        if (!used[to]) {
            DFS(to, G, used, res);
        }
    }
    res.push_back(v);
}

bool Circle(int v, const vector<vector<int>>& G, vector<string>& color) {
    color[v] = "grey";
    for (int to: G[v]) {
        if (color[to] == "white")
            Circle(to, G, color);
        if (color[to] == "grey")
            return false;
    }
    color[v] = "black";
    return true;
}

int main() {

    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int n, m;
    cin >> n >> m;

    int a, b;
    vector<vector<int>> G(n);
    for (int i = 0; i < m; ++i) {
        cin >> a >> b;
        a--;
        b--;
        G[a].push_back(b);
```

```
47      }
48
49
50      vector<string> color(n, "white");
51      for (int i = 0; i < n; ++i) {
52          if(!Circle(i, G, color)) {
53              cout << "-1\n";
54              return 0;
55          }
56      }
57
58      vector<bool> used(n);
59      vector<int> res;
60      for (int i = 0; i < n; ++i) {
61          if (!used[i]) {
62              DFS(i, G, used, res);
63          }
64      }
65
66      reverse (res.begin(), res.end());
67
68      for (size_t i = 0; i < res.size() - 1; ++i) {
69          cout << res[i] + 1 << " " << res[i + 1] + 1 << "\n";
70      }
71
72      return 0;
73  }
```

3 Консоль

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ g++ 1.cpp -o Lab8
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ cat test.txt
5 4
1 2
2 3
1 3
4 5
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ ./Lab8 <test.txt
4 5
5 1
1 2
2 3
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ cat test1.txt
3 3
1 2
2 3
3 1
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ ./Lab8 <test1.txt
-1
```

4 Тест производительности

Обход в глубину имеет сложность $O(|V| + |E|)$, где V - множесство вершин, а - множество рёбер графа. Два обхода в глубину будут иметь сложность $O(n + m)$. Объём затраченной памяти - $O(n)$.

Сравним жадный алгоритм с наивным, который перебирает все перестановки элементов. Сложность наивного алгоритма - $O(n!)$.

```
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ ./Lab8 <test0.txt
3 3
1 2
2 3
1 3
Топологическая сорт.: 25 mcs
Наивный алгоритм: 22 mcs
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ ./Lab8 <test1.txt
5 4
1 2
2 3
1 3
4 5
Топологическая сорт.: 26 mcs
Наивный алгоритм: 45 mcs
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ ./Lab8 <test2.txt
8 8
6 4
8 5
2 5
1 7
3 1
7 2
2 8
3 6
Топологическая сорт.: 27 mcs
Наивный алгоритм: 7277 mcs
parsifal@DESKTOP-3G70RV4:~/DA/Lab8$ ./Lab8 <test3.txt
10 11
4 6
4 5
```

```
4 3
3 2
3 10
10 1
2 7
5 2
7 8
9 8
5 9
Топологическая сорт.: 29 mcs
Наивный алгоритм: 569637 mcs
```

5 Выводы

Выполнив восьмую лабораторную работу по курсу «Дискретный анализ», я изучила различные алгоритмы, использующие идею жадных алгоритмов. В отличие от динамического программирования, жадные алгоритмы предполагают, что задача имеет оптимальное решение, которое строится из оптимальных решений подзадач с заранее определённым выбором, а не переборов всех вариантов перехода. Такой подход уменьшает временные и простроа=нственные ресурсы, нужные для решения задачи.

Список литературы

[1] *Топологическая сортировка — e-maxx.*
URL: `https://e-maxx.ru/algo/topological_sort` (дата обращения: 23.06.2021).

[2] *Использование обхода в глубину для поиска цикла - neerc.ifmo*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Использование_обхода_в_глубину_для`
(дата обращения: 23.06.2021).

[3] *Топологическая сортировкаа - brestprog*
URL: `https://brestprog.by/topics/topsort/` (дата обращения: 23.06.2021).