Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 7 по курсу «Дискретный анализ»

Студент: Бачурин П.Д.
Преподаватель: Макаров Н.К
Даты:
Оценка:
Подпись:

Москва, 2024

Задача

Заданы N объектов с ограничиениями на расположение вида «A должен находиться перед B». Необходимо найти такой порядок расположения объектов, что все ограничения будут выполняться.

Формат ввода

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

Формат вывода

−1 если расположить объекты в соответствии с требованиями невозможно, последовательность номеров объектов в противном случае.

Алгоритм

1. Инициализация данных:

- Считываем число объектов N и число ограничений M. Затем считываем M ограничений в вектор пар.

2. Алгоритм топологической сортировки:

- По данным ограничениям создаем граф, где каждое ограничение типа A перед B обозначает ребро из вершины A в вершину B.
- При построении графа подсчитываем сколько ребер входит в каждую вершину.
- Каждую вершину, в которую входит 0 ребер, добавляем в очередь для обхода в ширину.
- При обходе добавляем вершину в результирующий вектор и смотрим соседей. Для каждого соседа уменьшаем кол-во входящих в него ребер, если это кол-во стало равно нулю, то добавляем его в очередь для обхода.

5. Вывод результата:

- Если в результирующем векторе кол-во элементов не равно числу вершин, то значит, что хотя бы одно из ограничений невозможно выполнить.
- В ином случае выводим элементы результирующего вектора вектора.

Исходный код

```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>

std::vector<int> topological_sort(const int n,
std::vector<std::pair<int, int>> &constraints)
{
    std::vector<std::vector<int>> graph(n +
    1); std::vector in_degree(n + 1, 0);

    for (const auto &[a, b] : constraints)
        { graph[a].push_back(b);
        in_degree[b]++;
    }

    std::queue<int> q;
    for (int i = 1; i <= n; ++i) {
        if (in_degree[i] == 0) {
            q.push(i);
        }
    }

    std::vector<int> result;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        result.push_back(node);

        for (int neighbor : graph[node]) {
            in_degree[neighbor]--;
            if (in_degree[neighbor] == 0) {
                q.push(neighbor);
            }
        }
    }

    if (result.size() != n) {
        return {-1};
    }

    return result;
}

int main() {
    int n, m;
    std::cin >> n >> m;
    std::vector<std::pair<int, int>>
    constraints(m); for (int i = 0; i < m; ++i) {
        std::cin >> constraints[i].first >> constraints[i].second;
    }
```

```
    std::vector<int> sorted_order = topological_sort(n, constraints);if
    (sorted_order.size() == 1 && sorted_order[0] == -1) { std::cout <<
    -1 << "\n";
    } else {
        for (const int num : sorted_order) {
            std::cout << num << " ";
        }
        std::cout << "\n";
    }
}
```

Тесты

1.
ввод:
3 2
1 2
2 3
вывод:
1 2 3

2.
ввод:
5 6
1 2
1 3
1 4
1 5
2 5
3 2
вывод:
13425

3.
ввод:
2 2
1 2
2 1
вывод:
-1

4.
ввод:
10 2
1 2
2 10
вывод:13456789210

Вывод

В результате выполнения лабораторной работы я изучил основные алгортмы, использующие идею жадных алгоритмов, составил и отладил программу для своего варианта задания, работающую за O(n + m). В отличие от динамического программирования жадные алгоритмы предполагают, что задача имеет оптимальное решение, которое строится из оптимальных решений для подзадач с заранее определённым выбором, а не перебором всех вариантов перехода. Такой подход уменьшает временные и пространственные ресурсы, нужные для решения задачи.