Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Дискретный анализ»

Студент: А. Ю. Голов
Преподаватель: С. А. Михайлова
Группа: М8О-301Б-21

Дата:
Оценка:
Подпись:

Москва, 2023

Лабораторная работа №9

Формулировка задания: Задан взвешенный ориентированный граф, состоящий из n вершин и m ребер. Вершины пронумерованы целыми числами от 1 до n. Необходимо найти величину максимального потока в графе при помощи алгоритма Форда-Фалкерсона. Для достижения приемлемой производительности в алгоритме рекомендуется использовать поиск в ширину, а не в глубину. Истоком является вершина с номером 1, стоком – вершина с номером n. Вес ребра равен его пропускной способности. Граф не содержит петель и кратных ребер.

Формат ввода:

В первой строке заданы $1 \leq n \leq 2000$ и $1 \leq m \leq 10000$. В следующих m строках записаны ребра. Каждая строка содержит три числа – номера вершин, соединенных ребром, и вес данного ребра. Вес ребра – целое число от 0 до 10^9.

Формат вывода: Необходимо вывести одно число – искомую величину максимального потока. Если пути из истока в сток не существует, данная величина равна нулю.

1 Описание

Задача решается вполне тривиально, поэтому можно ограничиться кратким описанием алгоритма. Обращаясь же к коду, важно отметить, что был реализован абстрактный класс с приватными полями, необходимыми для имплементации графа, а так же методом поиска - использовать поиск в глубину, в ширину или с рекурсивным заглублением - решается при реализации интерфейса.

Алгоритм Форда-Фалкерсона использует поиск увеличивающих путей в остаточной сети для нахождения максимального потока в сети. Процесс повторяется до тех пор, пока увеличивающие пути существуют. Инициализация: Создать остаточную сеть, где пропускные способности ребер обновляются после каждого увеличивающего пути. Поиск увеличивающего пути: Используя поиск в глубину или ширину, найти путь от источника к стоку в остаточной сети. Нахождение пропускной способности пути: Найти минимальную пропускную способность ребер на увеличивающем пути. Обновление остаточной сети: Уменьшить пропускные способности по ребрам на увеличивающем пути и увеличить их обратно. Повторение: Повторять шаги 2-4, пока увеличивающие пути существуют. Максимальный поток: Суммировать пропускные способности всех увеличивающих путей. Алгоритм завершается, когда нет больше увеличивающих путей в остаточной сети.

2 Исходный код

```
1
2  #include <bits/stdc++.h>
3
4  int INF = 2e9;
5
6  class IGraph
7  {
8    public:
9    int Size() {
10     return this->data.size();
11   }
12
13   void SetVertex(int i, int j, int value) {
14     this->data[i][j] = value;
15   }
16
17   int GetVertex(int i, int j) {
18     return this->data[i][j];
19   }
20
21   bool Search(int source, int destination, std::vector<int> &parents);
22
23   protected:
24   std::vector<std::vector<int>> data;
25 };
26
27 class TGraph : public IGraph
28 {
29   public:
30   TGraph(int n) {
31     this->data = std::vector<std::vector<int>>(n + 1, std::vector<int>(n + 1));
32   }
33
34   void SetVertex(int i, int j, int value) {
35     this->data[i][j] = value;
36   }
37
38   int GetVertex(int i, int j) {
39     return this->data[i][j];
40   }
41
42   bool Search(int source, int destination, std::vector<int> &parents)
43   {
44     std::vector<bool> visited(this->Size());
45     std::queue<int> mem;
46
```

```
47      mem.push(source);
48      visited[source] = true;
49      parents[source] = -1;
50
51      while (!mem.empty())
52      {
53        int lastVer = mem.front();
54        mem.pop();
55
56        for (int curVer = 0; curVer < this->Size(); ++curVer)
57        {
58          if (!visited[curVer] && this->data[lastVer][curVer])
59          {
60            mem.push(curVer);
61            parents[curVer] = lastVer;
62            visited[curVer] = true;
63
64            if (curVer == destination) {
65              return true;
66            }
67          }
68        }
69      }
70
71      return false;
72    }
73  };
74
75  int64_t GetMaxFlow(TGraph &graph, int source, int destination)
76  {
77    int64_t result = 0;
78
79    TGraph resGraph = graph;
80    std::vector<int> parents(graph.Size());
81
82    while (resGraph.Search(source, destination, parents))
83    {
84      int curFlow = INF;
85
86      for (int curVertex = destination; curVertex != source; curVertex = parents[
          curVertex]) {
87        curFlow = std::min(curFlow, graph.GetVertex(parents[curVertex], curVertex));
88      }
89
90      for (int curVertex = destination; curVertex != source; curVertex = parents[
          curVertex])
91      {
92        resGraph.SetVertex(parents[curVertex], curVertex, resGraph.GetVertex(parents[
            curVertex], curVertex) - curFlow);
```

```
93        resGraph.SetVertex(curVertex, parents[curVertex], resGraph.GetVertex(curVertex,
              parents[curVertex]) + curFlow);
94      }
95
96      result += curFlow;
97    }
98
99    return result;
100 }
101
102 int main(void)
103 {
104   int n, m;
105   std::cin >> n >> m;
106
107   TGraph graph(n);
108
109   int src, dst, value;
110
111   for (int i = 0; i < m; ++i)
112   {
113     std::cin >> src >> dst >> value;
114     graph.SetVertex(src, dst, value);
115   }
116
117   std::cout << GetMaxFlow(graph, 1, n) << '\n';
118
119   return 0;
120 }
```

3 Тест производительности

В качестве тестирующего инструмента были выбраны гугл-тесты, интегрированные в программу при помощи CMake. Были применены методы «table-driven tests» и «fuzzy-test», ниже приведены табличные тестовые случаи.

Ввод	Вывод
5 6 1 2 4 1 3 3 1 4 1 2 5 3 3 5 3 4 5 10	7
4 4 1 2 4 1 3 3 2 3 1 3 4 5	4

4 Выводы

По итогам выполнения девятой лабораторной работы по курсу «Дискретный анализ», стало очевидно, красивое и простое решение задачи часто лежит на поверхности.

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] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 16.12.2013).

[3] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008