Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра ПМиК

КУРСОВОЙ ПРОЕКТ

по дисциплине «Методы оптимизации»

Решение задачи о максимальном потоке

Выполнил студент Марков В.А.
Ф.И.О.

Группы МГ-165

Работу принял _____ доцент к.ф-м.н. А.А. Рубан
подпись

Защищена _____ Оценка _____

Новсибирск 2017 г.

СОДЕРЖАНИЕ

1. Цель курсового проекта

Программно реализовать решение задачи о максимальном потоке.

2. Транспортная задача

Транспортная задача – математическая задача линейного программирования специального вида. Её можно рассматривать как задачу об оптимальном плане перевозок грузов из пунктов отправления в пункты потребления, с минимальными затратами на перевозки.

Транспортная задача по теории сложности вычислений входит в класс сложности *P*. Когда суммарный объём предложений (грузов, имеющихся в пунктах отправления) не равен общему объёму спроса на товары (грузы), запрашиваемые пунктами потребления, транспортная задача называется несбалансированной (открытой).

Транспортная задача (классическая) – задача об оптимальном плане перевозок однородного продукта из однородных пунктов наличия в однородные пункты потребления на однородных транспортных средствах (предопределённом количестве) со статичными данными и линеарном подходе (это основные условия задачи).

Для классической транспортной задачи выделяют два типа задач: критерий стоимости (достижение минимума затрат на перевозку) или расстояний и критерий времени (затрачивается минимум времени на перевозку).

Под названием транспортная задача, определяется широкий круг задач с единой математической моделью, эти задачи относятся к задачам линейного программирования и могут быть решены оптимальным методом. Однако, специальный метод решения транспортной задачи позволяет существенно упростить её решение, поскольку транспортная задача разрабатывалась для минимизации стоимости перевозок.

3. Решение с помощью теории графов

Рассматривается двудольный граф, в котором пункты производства находятся в верхней доле, а пункты потребления – в нижней. Пункты производства и потребления попарно соединяются рёбрами бесконечной пропускной способности и цены за единицу потока $C_{i,j}$

К верхней доле искусственно присоединяется исток. Пропускная способность рёбер из истока в каждый пункт производства равна запасу продукта в этом пункте. Цена за единицу потока у этих рёбер равна 0.

Аналогично к нижней доле присоединяется сток. Пропускная способность рёбер из каждого пункта потребления в сток равна потребности в продукте в этом пункте. Цена за единицу потока у этих рёбер тоже равна 0.

Дальше решается задача нахождения максимального потока минимальной стоимости (mincost maxflow). Её решение аналогично нахождению максимального потока в алгоритме Форда-Фалкерсона. Только вместо кратчайшего дополняющего потока ищется самый дешёвый.

4. Задача о максимальном потоке

В теории оптимизации и теории графов, задача о максимальном потоке заключается в нахождении такого потока по транспортной сети, что сумма потоков из истока, или, что то же самое, сумма потоков в сток максимальна.

Задача о максимальном потоке является частным случаем более трудных задач, как например задача о циркуляции [1].

5. Теорема Форда-Фалкерсона

Звучит так – величина максимального потока в графе путей равна величине пропускной способности его минимального разреза.

Достаточность – любой поток между вершинами t и s меньше или равен величине любого сечения. Пусть дан некоторый поток и некоторое сечение. Величина данного потока складывается из величин «грузов», перевозимых по

всем возможным путям из вершины t в s. Каждый такой путь обязан иметь общее ребро с данным сечением. Так как по каждому ребру сечения суммарно нельзя перевести «груза» больше, чем его пропускная способность, поэтому сумма всех грузов меньше или равна сумме всех пропускных способностей рёбер данного сечения.

Отсюда следует, что любой поток меньше или равен величине минимального сечения, а значит и максимальный поток меньше или равен величине минимального сечения.

На этой теореме основан алгоритм Форда-Фалкерсона поиска максимального потока в графе [2].

6. Алгоритм Форда-Фалкерсона

Алгоритм Форда-Фалкерсона решает задачу нахождения максимального потока в транспортной сети.

Идея алгоритма заключается в следующем. Изначально величине потока присваивается значение 0: $f(u,v) = 0$ для всех $u, v \in V$. Затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника s к стоку t, вдоль которого можно послать больший поток). Процесс повторяется, пока можно найти увеличивающий путь.

6.1. Формальное описание

Дан граф $G(V,E)$ с пропускной способностью $c(u,v)$ и потоком $f(u,v) = 0$ для рёбер из u в v. Необходимо найти максимальный поток из источника s в сток t. На каждом шаге алгоритма действуют те же условия, что и для всех потоков:

- $f(u,v) \leq c(u,v)$ Поток из u в v не превосходит пропускной способности.
- $f(u,v) = -f(u,v)$
- $\sum_v f(u,v) = 0 <=> f_{in}(u) = f_{out}(u)$ для всех узлов u, кроме s и t. Поток не изменяется при прохождении через узел.

Остаточная сеть $G_f(V, E_f)$ – сеть с пропускной способностью $c_f(u,v) = c(u,v) - f(u,v) - \mathrm{c}_f(p)$ и без потока.

Псевдокод алгоритма:

```
1  Вход: Граф G с пропускной способностью c , источник s и сток t
2  Выход: Максимальный поток f из s в t
3  f(u,v) ← 0 для всех рёбер (u,v)
4  Пока есть путь p  из s  в t в G_f, такой что c_f(p) > 0для всех рёбер
   (u,v) ∈ p :
5     Найти c_f(u,v) = min{c_f(u,v) | (u,v) ∈ p}
6     Для каждого ребра (u,v) ∈ p
7        f(u,v) ← f(u,v) + c_f(p)
8        f(u,v) ← f(u,v) − c_f(p)
```

Путь может быть найден, например, поиском в ширину (или поиском в глубину.

6.2. Сложность алгоритма

На каждом шаге алгоритм добавляет поток увеличивающего пути к уже имеющемуся потоку. Если пропускные способности всех рёбер – целые числа, легко доказать по индукции, что и потоки через все рёбра всегда будут целыми. Следовательно, на каждом шаге алгоритм увеличивает поток по крайней мере на единицу, следовательно, он сойдётся не более чем за $O(f)$ шагов, где f – максимальный поток в графе. Можно выполнить каждый шаг за время $O(E)$, где E – число рёбер в графе, тогда общее время работы алгоритма ограничено $O(E * f)$.

Если величина пропускной способности хотя бы одного из рёбер – иррациональное число, то алгоритм может работать бесконечно, даже не обязательно сходясь к правильному решению [3].

7. Реализация

Для решения задачи о максимальном потоке потребовалось реализовать алгоритм поиска в ширину (Breadth-first search) алгоритм Форда-Фалкерсона (Ford-Fulkerson algorithm). Реализации приведены на языке C++.

7.1. Реализация алгоритма поиска в ширину

Листинг алгоритма поиска в ширину

```
bool bfs(vector< vector<int>> rGraph, int s, int t, vector<int> &parent)
{
    const int V = rGraph.size();
    // Create a visited array and mark all vertices as not visited
    vector<bool> visited(V, false);
    /*
     * Create a queue, enqueue source vertex and mark source vertex
     * as visited
     */
    std::queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s]  = -1;
    while (!q.empty())// Standard BFS Loop
    {
        auto u = q.front();
        q.pop();
        for (auto v = 0; v < V; v++)
        {
            if (visited[v] == false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v]  = u;
                visited[v] = true;
            }
        }
    }
    /*
     * If we reached sink in BFS starting from source,
     * then return true, else false
     */
    return (visited[t] == true);
}
```

7.2. Реализация алгоритма Форда-Фалкерсона

Листинг алгоритма Форда-Фалкерсона

```
int fordFulkerson(vector<vector<int>> graph, int s, int t)
{
    const int V = graph.size();
    /*
     * Create a residual graph and fill the residual graph with
     * given capacities in the original graph as residual capacities
     * in residual graph
```

```
     */
    vector<vector<int>> rGraph = graph;
    /*
     * Residual graph where rGraph[i][j] indicates
     * residual capacity of edge from i to j (if there
     * is an edge. If rGraph[i][j] is 0, then there is not)
     */
    // This vector is filled by BFS and to store path
    vector<int> parent(V, 0);
    // There is no flow initially
    auto maxFlow = 0;
    // Augment the flow while tere is path from source to sink
    while (bfs(rGraph, s, t, parent))
    {
        /*
         * Find minimum residual capacity of the edges along the
         * path filled by BFS. Or we can say find the maximum flow
         * through the path found.
         */
        auto path_flow = INT_MAX;
        for (auto v = t; v != s; v = parent[v])
        {
            auto u    = parent[v];
            path_flow = std::min(path_flow, rGraph[u][v]);
        }
        /*
         * Update residual capacities of the edges and reverse edges
         * along the path
         */
        for (auto v = t; v != s; v = parent[v])
        {
            auto u        = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
        // Add path flow to overall flow
        maxFlow += path_flow;
    }
    return maxFlow; // Return the overall flow
}
```

7.3. Исходный граф

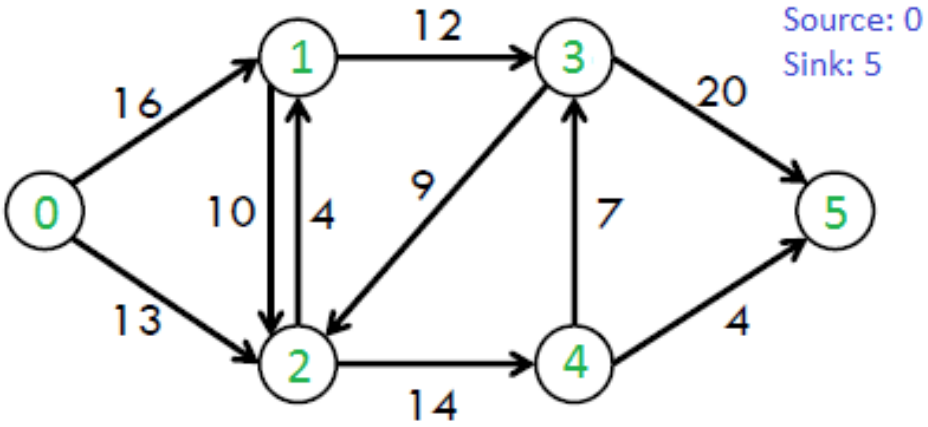

Рисунок 1 – Исходный граф, взятый из CLRS [4]

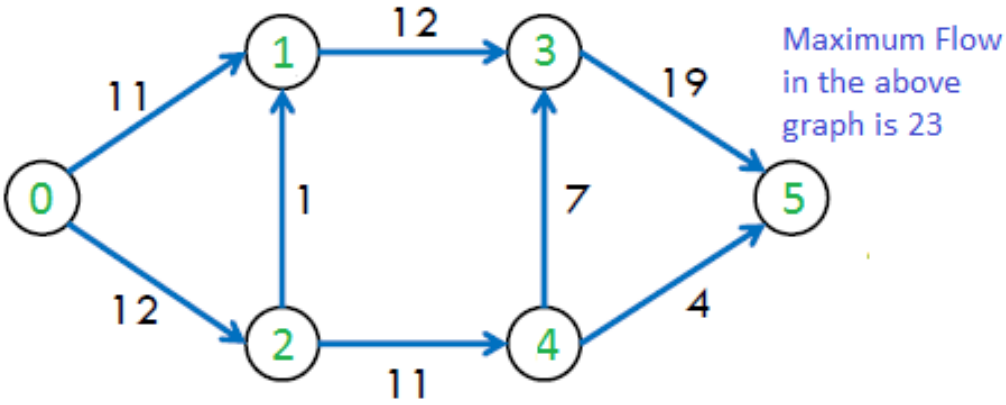

Рисунок 2 – Максимальный поток для исходного графа

7.4. Результат выполнения

```
Maximum-Flow-Problem$ build/max-flow-problem
The maximum possible flow is 23
```

Рисунок 3 – Результат выполнения

Заключение

В результате выполнения курсового проекта был реализована задача о нахождении максимального потока.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задача о максимальном потоке URL:
   https://ru.wikipedia.org/wiki/Задача_о_максимальном_потоке
2. Теорема Форда-Фалкерсона URL:
   https://ru.wikipedia.org/wiki/Теорема_Форда_—_Фалкерсона
3. Алгоритм Форда-Фалкерсона URL:
   https://ru.wikipedia.org/wiki/Алгоритм_Форда_—_Фалкерсона
4. Максимальный поток минимальной стоимости URL:
   https://habrahabr.ru/post/61884/
5. Introduction to Algorithms URL:
   https://cyberzhg.gitbooks.io/clrs/content/Chapter_26_Maximum_Flow/

ПРИЛОЖЕНИЕ

main.cpp

```
/*
 * C++ program for implementation of Ford Fulkerson algorithm
 */
#include <iostream>
#include <limits.h>
#include <string>
#include <queue>
#include <vector>

/*
 * Returns true if there is a path from source 's' to sink 't' in
 * residual graph. Also fills parent[] to store the path
 */
bool bfs(std::vector<std::vector<int>> rGraph, int s, int t, std::vector<int>
&parent)
{
    const int V = rGraph.size();
    // Create a visited array and mark all vertices as not visited
    std::vector<bool> visited(V, false);
    /*
     * Create a queue, enqueue source vertex and mark source vertex
     * as visited
     */
    std::queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s]  = -1;
    // Standard BFS Loop
    while (!q.empty())
    {
        auto u = q.front();
        q.pop();
        for (auto v = 0; v < V; v++)
        {
            if (visited[v] == false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v]  = u;
                visited[v] = true;
            }
        }
    }
    /*
     * If we reached sink in BFS starting from source, then return
     * true, else false
     */
    return (visited[t] == true);
}
/*
 * Returns the maximum flow from s to t in the given graph
 */
int fordFulkerson(std::vector<std::vector<int>> graph, int s, int t)
{
    const int V = graph.size();
    /*
     * Create a residual graph and fill the residual graph with
     * given capacities in the original graph as residual capacities
     * in residual graph
     */
    std::vector<std::vector<int>> rGraph = graph;
```

```
    /*
     * Residual graph where rGraph[i][j] indicates
     * residual capacity of edge from i to j (if there
     * is an edge. If rGraph[i][j] is 0, then there is not)
     */
    // This vector is filled by BFS and to store path
    std::vector<int> parent(V, 0);
    // There is no flow initially
    auto maxFlow = 0;
    // Augment the flow while tere is path from source to sink
    while (bfs(rGraph, s, t, parent))
    {
        /*
         * Find minimum residual capacity of the edges along the
         * path filled by BFS. Or we can say find the maximum flow
         * through the path found.
         */
        auto path_flow = INT_MAX;
        for (auto v = t; v != s; v = parent[v])
        {
            auto u    = parent[v];
            path_flow = std::min(path_flow, rGraph[u][v]);
        }
        /*
         * Update residual capacities of the edges and reverse edges
         * along the path
         */
        for (auto v = t; v != s; v = parent[v])
        {
            auto u        = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
        // Add path flow to overall flow
        maxFlow += path_flow;
    }
    // Return the overall flow
    return maxFlow;
}

int main(int argc, char const *argv[])
{
    const int source = 0;
    const int sink   = 5;
    std::vector<std::vector<int>> graph;
    graph.push_back({ 0, 16, 13, 0, 0, 0 });
    graph.push_back({ 0, 0, 10, 12, 0, 0 });
    graph.push_back({ 0, 4, 0, 0, 14, 0 });
    graph.push_back({ 0, 0, 9, 0, 0, 20 });
    graph.push_back({ 0, 0, 0, 7, 0, 4 });
    graph.push_back({ 0, 0, 0, 0, 0, 0 });
    std::cout << "The maximum possible flow is "
              << fordFulkerson(graph, source, sink)
              << std::endl;
     return 0;
}
```