Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Саратовский государственный технический университет
имени Гагарина Ю.А.»

Институт прикладных информационных
технологий и коммуникаций

Кафедра Информационная безопасность
автоматизированных систем

Направление подготовки 10.03.01 Информационная безопасность

Расчётно-графическая работа
по дисциплине «Языки программирования»

«Определение маршрута с учетом загрузки сети»

Выполнил: студент 1 курса
учебной группы с-ИБС11
очной формы обучения
Иванова Юлия Сергеевна
Проверил: ассистент каф. ИБС
Романчук С. П.

Саратов 2020

Дан файл с описанием карты города. Каждая связь имеет два веса – максимальная пропускная способность дороги и ее текущий уровень загрузки в процентах. Возможный вариант структуры файла приведен ниже:

N – число вершин.

X1 Y1 // координаты вершины

X2 Y2

…

XN YN

M – число связей.

N1 K1 P1 V1 // какие вершины связаны, пропускная способность и

загрузка

N2 K2 P2 V2

…

NM KM PM VM

С учетом пробок проложить наиболее быстрый маршрут из пункта A в пункт B.

Программа написана на C++ и включает в себя подсчёт графа, его сохранение, присвоение ребрам веса и поиск кратчайшего расстояния между A и B с помощью алгоритма Дейкстры. Более подробное решение рассмотрим далее.

Содержание

1. Введение ……стр.4
2. Теоретическая часть……...стр.5
3. Практическая часть…….стр.8
4. Заключение……стр.10
5. Приложения……...стр.11
6. Литература…….стр.16

Введение

Благодаря своему широкому применению, теория о нахождении кратчайших путей в последнее время интенсивно развивается. Нахождение кратчайшего пути – актуальная задача и используется практически везде: при нахождении оптимального маршрута между двумя объектами на местности, также используется в системах автопилота, используется для нахождения оптимального маршрута, в компьютерных сетях и тд.

Тема работы: «Определение маршрута с учетом загрузки сети». Сеть представляет из себя граф с ребрами различного веса, в котором необходимо найти кратчайшее расстояние между двумя вершинами.

Целью работы является разработка программы для решения задач поиска кратчайшего пути между вершинами графа.

Задачи работы:

1. рассмотрение основных сведений о графах;
2. рассмотрение алгоритма Дейкстры для нахождения кратчайшего пути между 2 вершинами;
3. описать представление графов на ЭВМ;
4. разработка программного продукта на языке программирования C++, реализующий алгоритм Дейкстры поиска кратчайшего пути между вершинами графа.

Теоретическая часть

Приложение написано на языке C++ и включает библиотеки <iosteram>, <fstream>, <vector>, <set>.

Iostream является частью стандартной библиотеки C++ и управляет вводом-выводом и использует объекты cin, cout, cerr и clog для передачи информации и из стандартных потоков ввода, вывода, ошибок без буферизации и ошибок с буферизацией соответственно.

Fstream – заголовочный файл из стандартной библиотеки C++, включающий набор классов, методов и функций, которые предоставляют интерфейс для чтения/записи данных из/в файл. Функции, включенные в данный файл, позволяют производить чтение из файлов.

Vector - замена стандартному динамическому массиву, память для которого выделяется вручную, с помощью оператора new. Его использование позволяет избежать утечек памяти и облегчает работу. Класс vector обладает стандартным набором методов для доступа к элементам, добавления и удаления элементов, а также получения количества хранимых элементов.

Set реализует шаблоны классов контейнеров std::set и std::multiset — сортированные ассоциативные контейнеры или множества. По сути это контейнеры, которые содержат некоторое количество отсортированных элементов; при добавлении нового элемента в множество он сразу становится на свое место так, чтобы не нарушать порядка сортировки. Порядок ввода элементов в множество никак не влияет на порядок хранения в множестве. Автоматическая сортировка элементов в множествах накладывает определенные ограничения. Например, в множествах нельзя изменить значение какого-то элемента напрямую, так как это могло бы сломать сортировку.

Классы в C++ — это абстракция описывающая методы, свойства, ещё не существующих объектов. Методы класса — это его функции. Свойства класса — его переменные. Также используются модификаторы public и private. Все функции и переменные, которые находятся после модификатора public,

становятся доступными из всех частей программы. Закрытые данные класса размещаются после модификатора доступа private.

Также в написании мы используем структуру. Структура - это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации. Объявление структуры приводит к образованию шаблона, используемого для создания объектов структуры. Переменные, образующие структуру, называются членами структуры. Обычно все члены структуры связаны друг с другом. Когда объявлена структурная переменная, компилятор автоматически выделяет необходимый участок памяти для размещения всех ее членов.

Графы. Алгоритм Дейкстры

граф - это математическая модель, представленная совокупностью множества вершин и связей между ними. Каждая пара вершин, имеющих связь, называется ребром графа.

Маршрут в графе - это чередующаяся последовательность вершин и рёбер, в которой любые два соседних элемента инцидентны. Если начальная и конечная вершины маршрута совпадают, то маршрут замкнут, иначе открыт. Маршрут называется цепью, если все его ребра различны, и простой цепью, если также различны и вершины.

Граф может как невзвешенный, так и взвешенный. Для первого типа все ребра графа соединяют вершины одинаковыми дугами. Для взвешенных графов каждая дуга имеет вес — числовое значение, которое характеризует ребро.

В данной задаче граф задан множеством точек на плоскости и множеством рёбер с различной загруженностью. Данный способ задания удобен для хранения графа в виде списков смежности — для каждой вершины хранить список всех её соседей вместе с весами рёбер.

Основной используемый алгоритм в программе – алгоритм Дейкстры. Алгоритм Дейкстры - алгоритм на графах, который находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса. Алгоритм работает пошагово — на

каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены. Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Мы рассматриваем всевозможные маршруты, в которых u является предпоследним пунктом. Вершины, в которые ведут рёбра из u, назовём соседями этой вершины. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещённую и повторим шаг алгоритма.

Практическая часть

Для решения поставленной задачи был реализован алгоритм, который находит кратчайшее расстояние между двумя вершинами в данном графе. Интерфейс программы реализован в виде консольного приложения я на языке C++.

Основной класс, реализующий поставленную задачу – PathFinder. Конструктор класса принимает строку, в которой записан путь к текстовому файлу с заданным графом. Его структура описана в теме работы.

Для считывания входного графа используется библиотека fstream, которая открывает поток считывания данных из текстового файла. Сначала для каждой вершины в массиве сохраняется её координаты на числовой плоскости, все вершины нумеруются в порядке их вхождения во входном файле.

После считывания всех координат происходит считывание ребер графа. В задаче описывается лишь загруженность дорог, следовательно, вес рёбер мы должны вычислять сами, в зависимости от расстояния между вершинами. В ходе выполнения работы была выведена формула, которая наиболее подходит для подсчета веса ребра, соединяющего две вершины. Эта формула описана в функции calc и зависит от расстояния между двумя вершинами (вычисляется как расстояние на плоскости между двумя точками) и загруженностью дороги. Если загруженность дороги 100%, то вес ребра присваивается очень большому числу так, чтобы при работе программы — это ребро воспринималось, как отсутствие ребра между вершинами.

После считывания графа пользователю предоставляется ввести номера вершин, между которыми необходимо найти кратчайшее расстояние. Введённые данные проверяются, чтобы эти вершины присутствовали в исходном графе.

После проверки введенных данных начинается вычисление кратчайшего пути между вершинами. Для решения данной задачи используется алгоритм Дейкстры, описанный выше, но с небольшой модернизацией.

Поскольку в контейнере нам надо хранить вершины, упорядоченные по их величинам, то удобно в set помещать пары: первый элемент пары — расстояние, а второй — номер вершины. В результате в set будут храниться пары, автоматически упорядоченные по расстояниям. Изначально в него помещаем стартовую вершину s с её расстоянием. Основной цикл алгоритма выполняется, пока в очереди есть хоть одна вершина. Из очереди извлекается вершина с наименьшим расстоянием, и затем из неё выполняются релаксации. Перед выполнением каждой успешной релаксации мы сначала удаляем из set старую пару, а затем, после выполнения релаксации, добавляем обратно новую пару с новым расстоянием.

Это оптимизация ускоряет решения, поскольку set позволяет хранить отсортированные данные и позволяет добавлять и извлекать элементы за логарифмическое время.

Результат работы функции нахождения кратчайшего пути помещается в структуру path, которая хранит в себе вершины начали и окончания пути, значение кратчайшего расстояния, а также сам путь.

Для восстановления пути при подсчете кратчайшего расстояния для каждой вершины сохраняется её предок — вершина, из которой мы посещаем текущую по кратчайшему пути. Для стартовой вершины данное значение равно -1. После работы алгоритма мы просто рекурсивно перебираем все вершины в пути, начиная с финальной, и сохраняем весь путь в массив. В результате в списке окажется весь путь в перевернутом формате, так как начинами с последней вершины. Перевернув данный массив, мы получим путь от стартовой вершины до конечной.

Получив в основной программе структуру с ответом, мы выводим пользователю весь путь и повторяем все действия заново для новых вершин поиска.

Заключение

Протестировав данный алгоритм задачи, мы создали интерфейс, интуитивно понятный для пользователя для подсчета кратчайшего пути из одной точки в другую с учетом таким факторов, как загруженность и пропускная способность.

Приложение

PathFinder.cpp

```
#include "PathFinder.h"

PathFinder::PathFinder()
{
	city.clear();
	roads.clear();
}

PathFinder::PathFinder(string path)
{
	city.clear();
	roads.clear();

	ifstream in(path);
	if (in.fail()) {
		throw exception("Файл не существует");
	}

	int n, m;
	in >> n;
	for (int i = 0; i < n; i++) {
		int x, y;
		in >> x >> y;
		city.push_back({x, y});
		roads.push_back(vector<pair<int, double> >(0));
	}
	in >> m;
	for (int i = 0; i < m; i++) {
		int a, b, s, p;
		in >> a >> b >> s >> p;
		a--;
		b--;
		auto distance = calc(s, p, dist(city[a], city[b]));
		if (distance != 1000000000) {
			roads[a].push_back({ b, distance });
			roads[b].push_back({ a, distance });
		}
	}
	in.close();
}

PathFinder::path PathFinder::FindPath(int a, int b)
{
	a--;
	b--;
	const int INF = 1000000000;
	PathFinder::path result(a, b);
```

```
	set<pair<double, int> > q;
	vector<double> dist(city.size(), INF);
	vector<int> p(city.size());
	dist[a] = 0;
	for (int i = 0; i < dist.size(); i++) {
		q.insert({ dist[i], i });
		p[i] = -1;
	}

	// Поиск пути наименьшего веса
	while (q.size() > 0) {
		auto f = (*q.begin()).second;
		q.erase(*q.begin());
		for (int j = 0; j < roads[f].size(); j++) {
			int c = roads[f][j].first;
			if (dist[c] > dist[f] + roads[f][j].second) {
				q.erase({ dist[c], c });
				dist[c] = dist[f] + roads[f][j].second;
				q.insert({ dist[c], c });
				p[c] = f;
			}
		}
	}

	// Восстановление пути
	if (dist[b] != INF) {
		result.time = -1;
		int t = b;
		while (p[t] != -1) {
			result.way.push_back(t);
			t = p[t];
		}
		result.way.push_back(a);
		result.time = dist[b];

		reverse(result.way.begin(), result.way.end());
	}
	else {
		result.time = dist[b];
	}
	return result;
}

bool PathFinder::exist(int x)
{
	return x <= city.size() && x > 0;
}

double PathFinder::calc(int s, int p, double dist)
{
	if (p == 100) {
```

```
		return 1000000000;
	}
	if (p == 0) {
		return dist / s;
	}
	return (dist / s) * (p / 10);
}
// (Расстояние / [пропускная способность]) * ([загруженность] / 10)

double PathFinder::dist(pair<int, int> a, pair<int, int> b)
{
	return sqrt(double((a.first - b.first) * (a.first - b.first) + (a.second - b.second) * (a.second -
b.second)));
	}
```

Source.cpp

```
#include <iostream>

#include "PathFinder.h"

using namespace std;

int main(){
	setlocale(LC_ALL, "Russian");
	PathFinder solution;
	try {
		solution = PathFinder("data.txt");
	}
	catch (exception e){
		cout << e.what();
	}

	cout << "Данные успешно загружены\n";
	cout << "Для выхода из программы введите 0\n";


	for (;;) {
		cout << "Введите номера вершин, между которыми необходимо найти путь: \n";
		int a, b;
		cin >> a;
		if (a == 0) {
			return 0;
		}
		if (!solution.exist(a)) {
			printf("Вершины с номером %d в городе нет\n", a);
			continue;
		}
		cin >> b;
		if (!solution.exist(b)) {
			printf("Вершины с номером %d в городе нет\n", b);
			continue;
```

```
                }
                auto res = solution.FindPath(a, b);
                if (res.time != -1) {
                        cout << "Кратчайший путь между вершинами с текущей загруженностью
дорог: \n";
                        for (int i = 0; i < res.way.size(); i++) {
                                cout << res.way[i] + 1;
                                if (i != res.way.size() - 1) {
                                        cout << " -> ";
                                }
                                else {
                                        cout << "\n";
                                }
                        }
                }
                else {
                        cout << "Между выбранными вершинами нет пути.\n";
                }
        }
        }
```

PathFinder.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <set>


using namespace std;


class PathFinder
{
public:
        // Структура для возвращения пути
        struct path{
                // Точки начала и конца маршрута
                int A, B;

                // Итоговое время (1000000000), если нет пути
                int time;

                // Вектор для хранения пути
                vector<int> way;

                path(int a, int b) {
                        A = a;
                        B = b;
                        time = -1;
                        way.clear();
```

```
		}

		// Возвращает ответ на задачу
		int getTime() {
			return time;
		}
	};

	PathFinder();
	// Конструктор класса, который принимает название файла, в котором хранятся данные
	PathFinder(string path);

	// Нахождение кратчайшего пути с помощью алгоритма Дейкстры
	path FindPath(int a, int b);

	// Есть ли город с данным номером в сети
	bool exist(int x);
private:
	// Подсчет веса ребра
	double calc(int s, int p, double dist);

	// Подсчет дистанции между городами
	double dist(pair<int, int> a, pair<int, int> b);

	// Хранение координат вершин
	vector<pair<int, int> > city;

	// Список смежности вершин графа
	vector<vector<pair<int, double> > > roads;
	};
```

Data.txt

```
6
1 1
1 2
2 3
3 4
4 5
5 6
8
1 2 10 50
1 3 10 90
2 3 10 10
2 5 10 9
3 5 10 0
3 4 10 9
5 6 10 9
4 5 10 9
```

Литература

1. Герберт Шилдт. Полный справочник по C++ — 4-е изд. — М.: Вильямс, 2011.
2. Бьёрн Страуструп. Язык программирования C++ Пер. с англ. — 3-е изд.
3. https://e-maxx.ru/algo/dijkstra_sparse
4. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ — 2-е изд.