Университет ИТМО

Факультет ПИиКТ

Лабораторная работа №2 по дисциплине
“Системы искусственного интеллекта”

(3ий курс бакалавриата ФПИиКТ)

Студент:

Иванов Евгений Дмитриевич

Группа P33111

Преподаватель:

Авдюшина Анна Евгеньевна

Цель задания: Исследование алгоритмов решения задач методом поиска.

Описание предметной области. Имеется транспортная сеть, связывающая города СНГ. Сеть представлена в виде таблицы связей между городами. Связи являются двусторонними, т.е. допускают движение в обоих направлениях. Необходимо проложить маршрут из одной заданной точки в другую.

Этап 1. Неинформированный поиск.

На этом этапе известна только топология связей между городами.

Выполнить:

1) поиск в ширину;
2) поиск глубину;
3) поиск с ограничением глубины;
4) поиск с итеративным углублением;
5) двунаправленный поиск.

Отобразить движение по дереву на его графе с указанием сложности каждого вида поиска. Сделать выводы.

Этап 2. Информированный поиск.

Воспользовавшись информацией о протяженности связей от текущего узла, **выполнить:**

1) жадный поиск по первому наилучшему соответствию;
2) затем, используя информацию о расстоянии до цели по прямой от каждого узла, выполнить поиск методом минимизации суммарной оценки A*.

Отобразить на графе выбранный маршрут и сравнить его сложность с неинформированным поиском. Сделать выводы.

Таблица связей между городами

Город 1	Город 2	Расстояние, км
Вильнюс	Брест	531
Витебск	Брест	638
Витебск	Вильюс	360
Воронеж	Витебск	869
Воронеж	Волгоград	581
Волгоград	Витебск	1455
Витебск	Ниж.Новгород	911
Вильнюс	Даугавпилс	211
Калининград	Брест	699
Калиниград	Вильнюс	333
Каунас	Вильнюс	102
Киев	Вильнюс	734
Киев	Житомир	131
Житомир	Донецк	863

Житомир	Волгоград	1493
Кишинев	Киев	467
Кишинев	Донецк	812
С.Петербург	Витебск	602
С.Петербург	Калининград	739
С.Петербург	Рига	641
Москва	Казань	815
Москва	Ниж.Новгород	411
Москва	Минск	690
Москва	Донецк	1084
Москва	С.Петербург	664
Мурманск	С.Петербург	1412
Мурманск	Минск	2238
Орел	Витебск	522
Орел	Донецк	709
Орел	Москва	368
Одесса	Киев	487
Рига	Каунас	267
Таллинн	Рига	308
Харьков	Киев	471
Харьков	Симферополь	639
Ярославль	Воронеж	739
Ярославль	Минск	940
Уфа	Казань	525
Уфа	Самара	461

Номер варианта	Исходный пункт	Пункт назначения
1	Мурманск	Одесса
2	С.Петербург	Житомир
3	Самара	Ярославль
4	Рига	Уфа
5	Казань	Таллин
6	Симферополь	Мурманск
7	Рига	Одесса
8	Вильнюс	Одесса
9	Брест	Казань
10	Харьков	Ниж.Новгород

Вариант: 8 + 1 = **9**

Код алгоритмов:

```
bool depth_first_search(const string &a, const string &b) {
    if (a == b) { return true; }

    map_visited[a] = true;

    for (auto i = adj[a].begin(); i != adj[a].end(); i++) {
        const string s = i->first;
        // проверка на следующую вершину, если она не посещалась
при обходе
        if (!map_visited[s]) {
            lhs[s] = a;
            if (depth_first_search(s, b)) {
                return true; // вернулись если путь верный
            }
        }
    }
    return false;
}

bool breadth_first_search(const string &a, const string &b) {
```

```
    // очередь для обхода в ширину
    queue<string> que;
    que.push(a);

    map_visited[a] = true;
    while (!que.empty()) {
        const string u = que.front();
        que.pop();

        // условие выхода(нашли конечный путь)
        if (u == b) { return true; }

        for (auto i = adj[u].begin(); i != adj[u].end(); i++) {
            const string v = i->first;

            // проверяем на посещения все оставшиеся вершины для
данной, которые ещё не были посещены
            if (!map_visited[v]) {
                map_visited[v] = true;
                lhs[v] = u;
                que.push(v);        // поставили в очередь на обход
            }
        }
    }
    return false;
}

bool depth_limited_search(const string &a, const string &b, const
int limit) {
    if (a == b) { return true; }
    if (limit == 0) { return false; }

    map_visited[a] = true;

    for (auto i = adj[a].begin(); i != adj[a].end(); i++) {
        const string s = i->first;
        // проверка на следующую вершину, если она не посещалась
при обходе
        if (!map_visited[s]) {
            lhs[s] = a;
            if (depth_limited_search(s, b, limit - 1)) {
```

```
                return true; // вернулись если путь верный
            }
        }
    }
    return false;
}

bool iterative_depth_first_search(const string &a, const string &b)
{
    for (int i = 1; i < adj.size(); i++) {
        map_visited.clear();
        if (depth_limited_search(a, b, i)) {
            return true;
        }
    }
    return false;
}

bool bidirectional_search(const string &a, const string &b) {
    queue<string> que, pue;
    que.push(a);
    pue.push(b);

    map_visited[a] = true;
    map_visited_2[b] = true;
    while (!que.empty() && !pue.empty()) {

        const string q = que.front();
        que.pop();

        for (auto i = adj[q].begin(); i != adj[q].end(); i++) {
            const string s = i->first;
            // проверка если вершина ещё не посещалась
            if (!map_visited[s]) {
                map_visited[s] = true;
                lhs[s] = q;
                que.push(s);        // положили в очередь на обход
            }
        }
```

```
        // аналогичный обход с обратной стороны
        const string p = pue.front();
        pue.pop();

        for (auto i = adj[p].begin(); i != adj[p].end(); i++) {
            const string s = i->first;

            if (!map_visited_2[s]) {
                map_visited_2[s] = true;
                rhs[s] = p;
                pue.push(s);
            }
        }
        // условие выхода для двустороннего обхода
        if (check_set(map_visited, map_visited_2)) { return true; }
    }
    return false;
}

bool best_first_search(const string &a, const string &b) {
    if (a == b) { return true; }

    map_visited[a] = true;

    for (auto i = adj[a].begin(); i != adj[a].end(); i++) {
        string u = i->first;
        for (auto j = adj[a].begin(); j != adj[a].end(); j++) {
            const string v = j->first;
            // проверяем на новую минимальную эвристику
            if (!map_visited[v] && paths[u] > paths[v]) {
                u = v;
            }
        }

        if (!map_visited[u]) {
            lhs[u] = a;
            if (best_first_search(u, b)) {
                return true;
            }
        }
    }
```

```
    return false;
}

bool sum_path_and_value_search(const string &a, const string &b) {
    map<const string, int> weight;

    using pair = pair<int, string>;
    priority_queue<pair, vector<pair>, greater<pair>> que;
    que.push(make_pair(weight[a] + paths[a], a));

    map_visited[a] = true;
    while (!que.empty()) {
        const string u = que.top().second;
        que.pop();

        if (u == b) { return true; }

        for (auto i = adj[u].begin(); i != adj[u].end(); i++) {
            const string v = i->first;
            const int w = i->second;
            if (!map_visited[v]) {
                map_visited[v] = true;
                lhs[v] = u;
                weight[v] = weight[u] + w;

                que.push(make_pair(weight[v] + paths[v], v));
            }
        }
    }
    return false;
}
```

Результат работы программы:

```
DFS:
Брест ---> Вильнюс ---> Витебск ---> Воронеж ---> Волгоград ---> Житомир ---> Киев ---> Кишинев ---> Донецк ---> Москва
---> Казань

BFS:
Брест ---> Витебск ---> Ниж.Новгород ---> Москва ---> Казань

iterative deepening search:
Брест ---> Вильнюс ---> Витебск ---> Ниж.Новгород ---> Москва ---> Казань

bidirectional search:
```

Вывод: Я научился работать с методами информативного и неинформативного поисков. Прописал алгоритмы поиска в глубину, обход в ширину, обход и итеративным углублением, двунаправленный обход, обход по лучшему совпадению, поиск методом по минимальной сумме.