

Шинжлэх Ухаан Технологийн Их Сургууль
Мэдээлэл, Холбооны Технологийн Сургууль



Бие даалт №1

F.CSM301: Алгоритмын шинжилгээ ба зохиомж

Сэдвийн нэр: Улаанбаатар хотын газрын зургийг ашиглан богино зам тооцох

Шалгасан багш: Д.Батмөнх

Гүйцэтгэсэн оюутан: Т.Мөнхцэцэн /B232270025/

Улаанбаатар хот
2025 он

Агуулга

1 Танилцуулга	2
1.1 Ашигласан технологиуд	2
2 Системийн Архитектур	2
2.1 Ерөнхий бүтэц	2
2.2 Графын загвар	2
2.3 Θгөгдөл боловсруулах	3
3 Хайлтын Алгоритмууд	3
3.1 BFS алгоритм	3
3.1.1 Хэрэгжилт	3
3.1.2 Цаг хугацааны нарийн төвөгтэй байдал	4
3.2 DFS алгоритм	4
3.2.1 Хэрэгжилт	4
3.2.2 Цаг хугацааны нарийн төвөгтэй байдал	5
3.3 Dijkstra алгоритм	5
3.3.1 Хэрэгжилт	5
3.3.2 Цаг хугацааны нарийн төвөгтэй байдал	6
4 Хэрэгжүүлэлт	6
4.1 Θгөгдөл боловсруулалт	6
4.2 Графын бүтэц	6
5 REST API Дизайн	6
5.1 API Endpoint-үүд	6
6 Туршилт ба Үр Дүн	7
6.1 Туршилтын орчин	7
6.2 Алгоритмуудын харьцуулалт	7
6.3 Frontend (Leaflet.js) ашиглалт	7
7 Дүгнэлт	8

1 Танилцуулга

Энэхүү бие даалтаар графын хайлтын алгоритмуудын (BFS, DFS болон Dijkstra) үндсэн санааг ойлгож, Улаанбаатар хотын газрын зургийг ашиглан богино зам тооцох даалгаврыг хэрэгжүүлсэн.

1.1 Ашигласан технологиуд

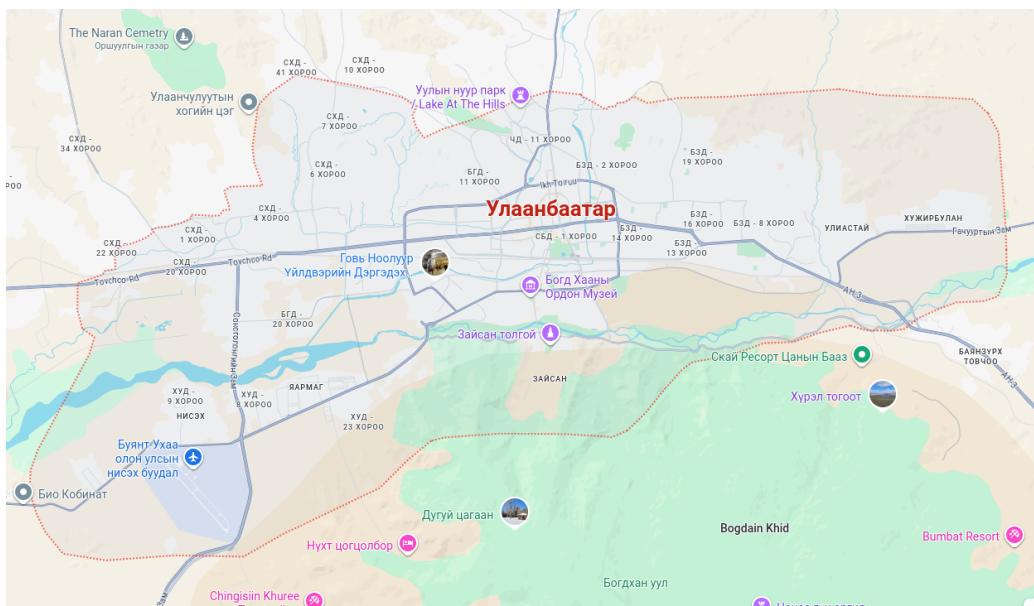
- Python Flask - Backend framework
- OpenStreetMap (OSM) - Газар зүйн өгөгдөл
- Leaflet.js - Газрын зураг дэлгэц
- GeoPandas - Газар зүйн өгөгдөл боловсруулах
- HTML/CSS/JavaScript - Frontend

2 Системийн Архитектур

2.1 Ерөнхий бүтэц

Систем нь дараах 3 үндсэн хэсгээс бүрдэнэ:

1. **Backend:** Python Flask REST API
2. **Frontend:** HTML/JavaScript веб интерфейс
3. **Өгөгдлийн сан:** OSM shapefile өгөгдөл



Зураг 1: Улаанбаатар хотын газрын зургийн тойм

2.2 Графын загвар

OSM өгөгдлийг граф хэлбэрт оруулахдаа дараах бүтцийг ашигласан:

Listing 1: Графын бүтэц

```
1 class GraphBuilder:  
2     def __init__(self):
```

```

3     self.nodes = {} # node_id -> (lat, lon)
4     self.adjacency = {} # node_id -> [(neighbor, weight)]
5     self.node_count = 0

```

2.3 Өгөгдөл боловсруулах

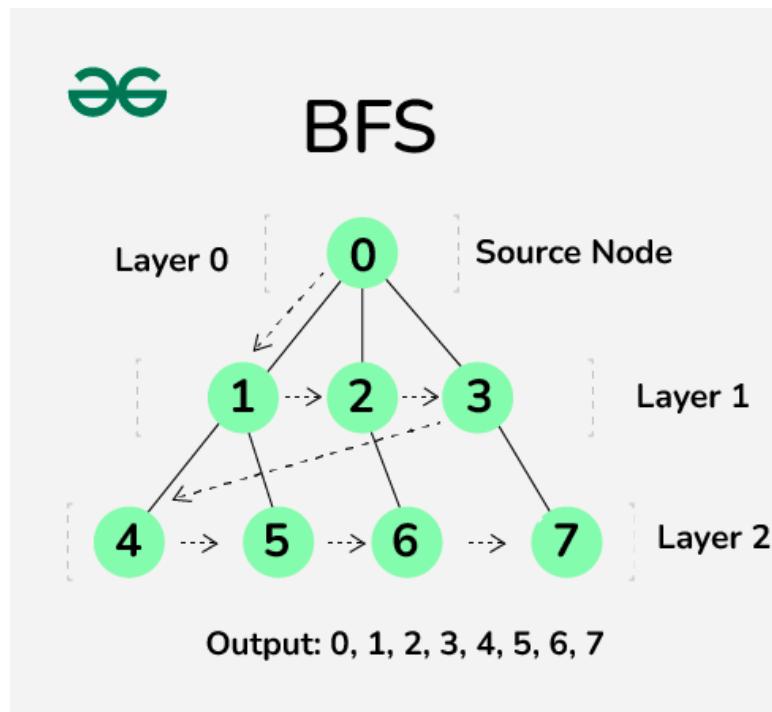
OSM shapefile файлаас дараах мэдээллүүдийг гарган авсан:

- Замын геометр (LineString)
- Замын төрөл (fclass)
- Нэг чигийн зам (oneway)
- Координатууд

3 Хайлтын Алгоритмууд

3.1 BFS алгоритм

BFS (Breadth-First Search) алгоритм нь графын бүх оройг төвшин төвшнөөр нь эрэмбэлж хайдаг. Энэ нь хамгийн богино замыг олоход тохиромжтой.



Зураг 2: BFS алгоритмын ажиллах зарчим

3.1.1 Хэрэгжилт

Listing 2: BFS алгоритм

```

1 def bfs(self, start, end):
2     visited = set([start])
3     queue = deque([(start, [start])])
4
5     while queue:
6         node, path = queue.popleft()
7         if node == end:

```

```

8     return path
9     for neigh, w in self.graph.adjacency.get(node, []):
10        if neigh not in visited:
11            visited.add(neigh)
12            queue.append((neigh, path + [neigh]))

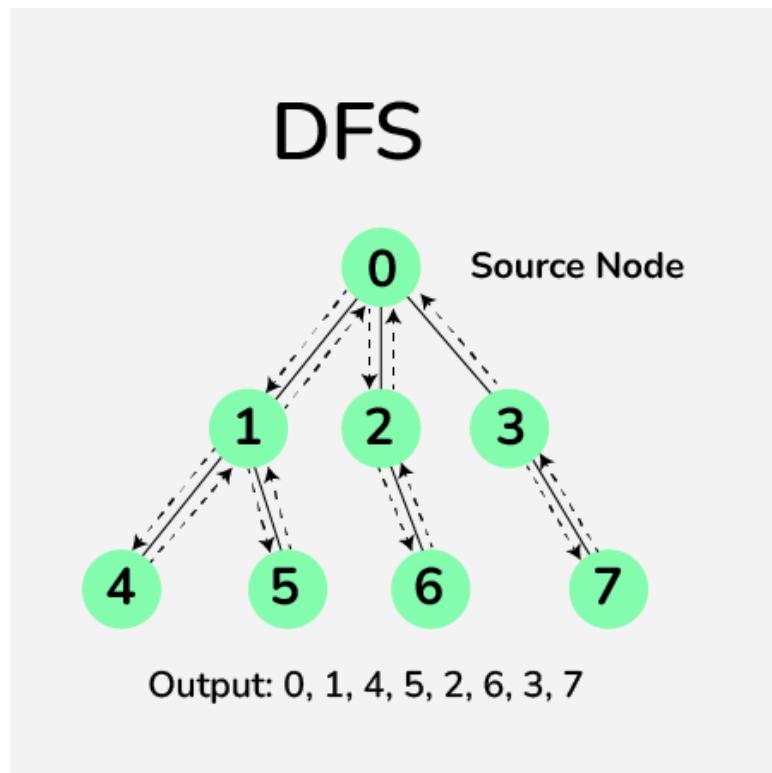
```

3.1.2 Цаг хугацааны нарийн төвөгтэй байдал

- Оролт: $O(V + E)$
- Санах ой: $O(V)$

3.2 DFS алгоритм

DFS (Depth-First Search) алгоритм нь гүнзгийрүүлэн хайлт хийж, боломжит бүх замыг олоход тохиромжтой.



Зураг 3: DFS алгоритмын ажиллах зарчим

3.2.1 Хэрэгжилт

Listing 3: DFS алгоритм

```

1 def dfs(self, start, end, max_depth=5000):
2     best_path = None
3     best_distance = float('inf')
4
5     def dfs_recursive(current, path, visited, depth):
6         if current == end:
7             current_distance = self.path_length(path)
8             if current_distance < best_distance:
9                 best_path = path.copy()
10                best_distance = current_distance
11
12

```

```

12     visited.add(current)
13     for neighbor, weight in self.graph.adjacency.get(current, []):
14         if neighbor not in visited:
15             dfs_recursive(neighbor, path + [neighbor],
16                           visited.copy(), depth + 1)

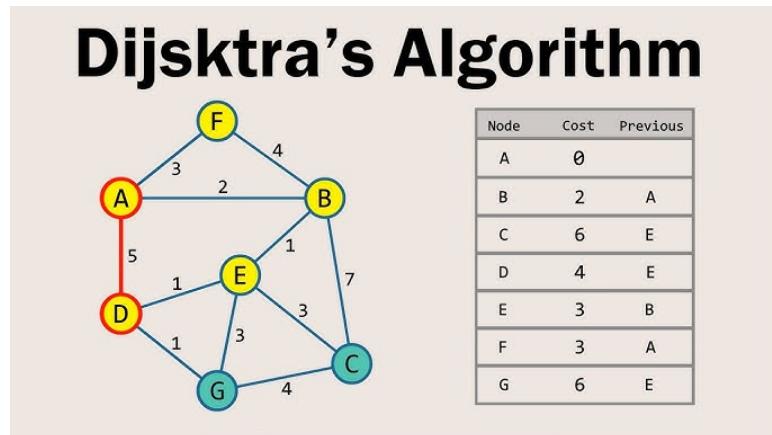
```

3.2.2 Цаг хугацааны нарийн төвөгтэй байдал

- Оролт: $O(V + E)$
- Санах ой: $O(V)$ (Өөрөө өөрийгөө давтан дуудах стек)

3.3 Dijkstra алгоритм

Dijkstra алгоритм нь эхлэх цэгээс бусад бүх цэг хүртэлх хамгийн богино замыг олдог greedy алгоритм юм.



Зураг 4: Dijkstra алгоритмын ажиллах зарчим

3.3.1 Хэрэгжилт

Listing 4: Dijkstra алгоритм

```

1 def dijkstra(self, start, end):
2     dist = {n: float("inf") for n in self.graph.nodes}
3     prev = {n: None for n in self.graph.nodes}
4     dist[start] = 0
5     pq = [(0, start)]
6
7     while pq:
8         d, u = heapq.heappop(pq)
9         if u == end:
10            break
11        for v, w in self.graph.adjacency.get(u, []):
12            nd = d + w
13            if nd < dist[v]:
14                dist[v] = nd
15                prev[v] = u
16                heapq.heappush(pq, (nd, v))

```

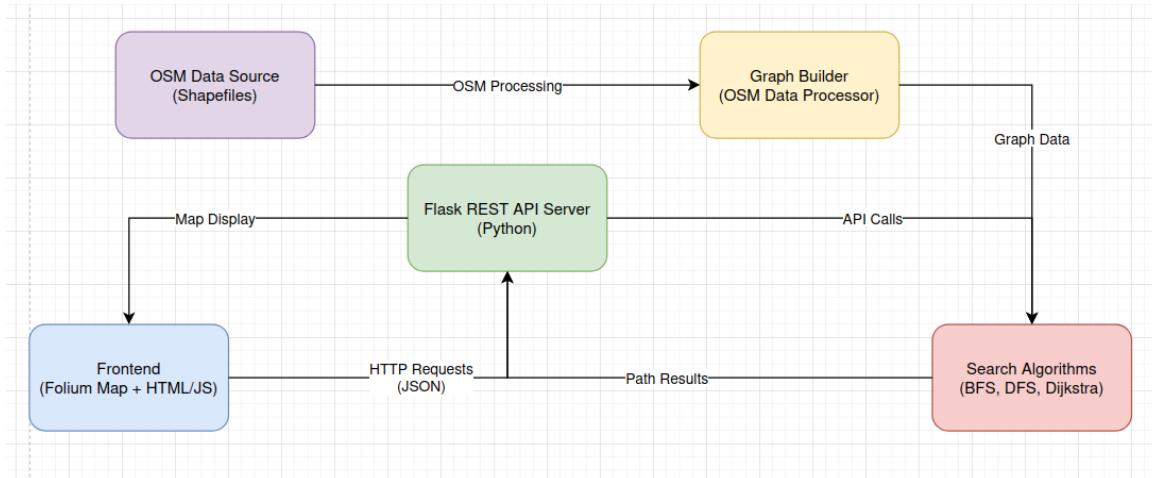
3.3.2 Цаг хугацааны нарийн төвөгтэй байдал

- Оролт: $O((V + E) \log V)$
- Санах ой: $O(V)$

4 Хэрэгжүүлэлт

4.1 Өгөгдөл боловсруулалт

Улаанбаатар хотын замын мэдээллийг OpenStreetMap-ээс татаж авч, GeoPandas сан ашиглан боловсруулсан. Дараах мэдээллүүдийг ашигласан:



Зураг 5: Өгөгдөл боловсруулалтын үйл явцын диаграмм

- node - замын уулзвар
- edge - замын шугам
- weight - замын урт
- oneway - нэг чигийн зам

4.2 Графын бүтэц

Python-д GraphBuilder ангийг ашиглан замын графыг үүсгэсэн. Гол үүргүүд нь:

- `add_node(lat, lon)` - цэг нэмэх
- `add_edge(n1, n2, weight)` - хоёр цэгийг холбох
- `haversine(lat1, lon1, lat2, lon2)` - хоёр цэгийн хоорондын зайд тооцоолох
- `load_osm_data(path)` - OSM Shapefile уншиж граф үүсгэх

5 REST API Дизайн

5.1 API Endpoint-үүд

Endpoint	Method	Тайлбар
/api/path	POST	Зам олох
/api/graph_info	GET	Графын мэдээлэл
/api/debug_connection	POST	Холболт шалгах

Хүснэгт 1: REST API endpoint-үүд

API нь эхлэл ба төгсгөлийн цэгийг хүлээн авч, сонгосон алгоритмын дагуу замыг тооцож, JSON хэлбэрээр үр дүнг буцаадаг.

6 Туршилт ба Үр Дүн

6.1 Туршилтын орчин

- Графын хэмжээ: 1,109,131 цэг, 2,290,758 холбоос
- Замын тоо: 73,639
- Туршилтын цэгүүд: Улаанбаатар хотын гол байршлууд

6.2 Алгоритмуудын харьцуулалт

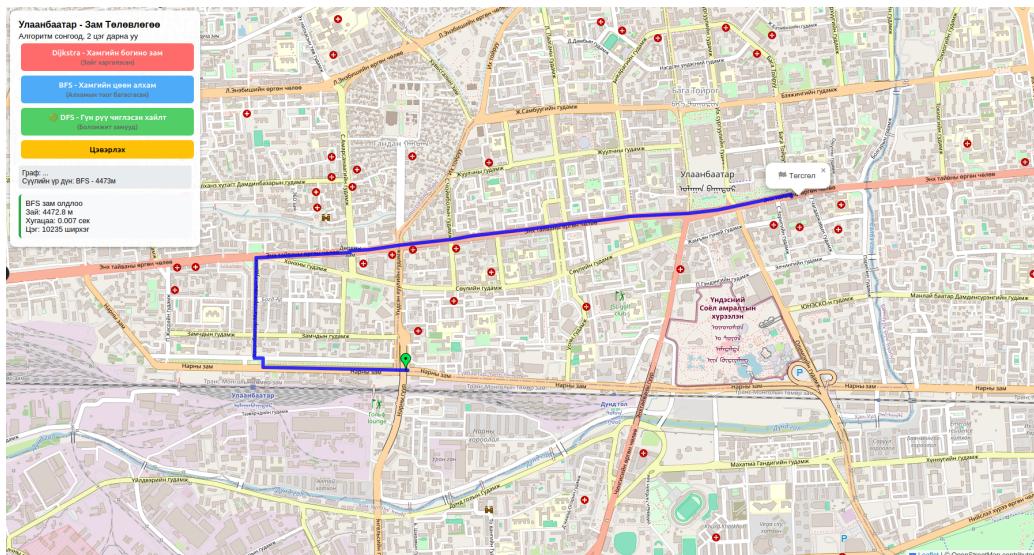
Алгоритм	Дундаж хугацаа	Цэгийн тоо	Замын урт
Dijkstra	0.17с	640	683.3м
BFS	0.0с	391	683.3м
DFS	0.001с	608	989.9м

Хүснэгт 2: Алгоритмуудын гүйцэтгэлийн харьцуулалт

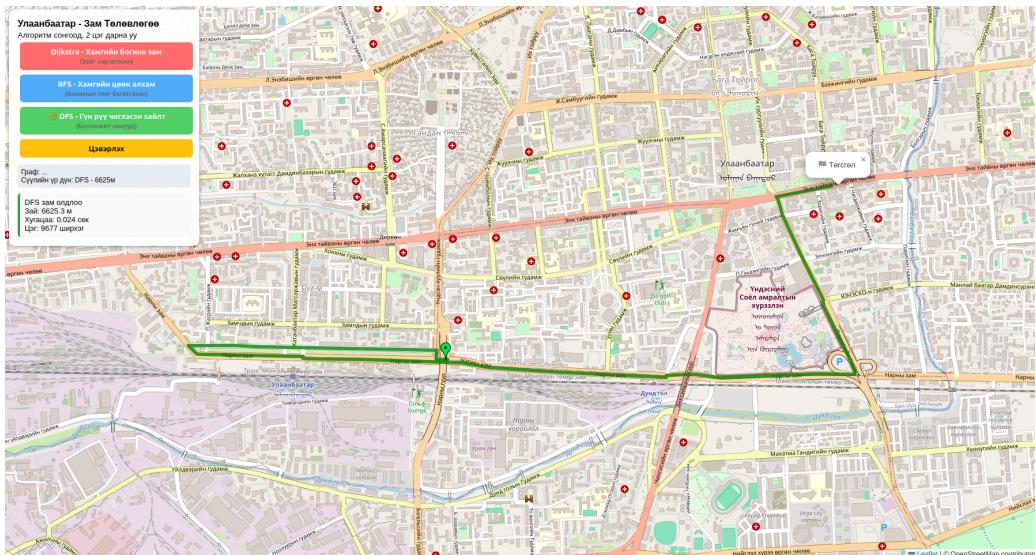
6.3 Frontend (Leaflet.js) ашиглалт

HTML + JavaScript ашиглан газрын зураг дээр интерактив байдлаар:

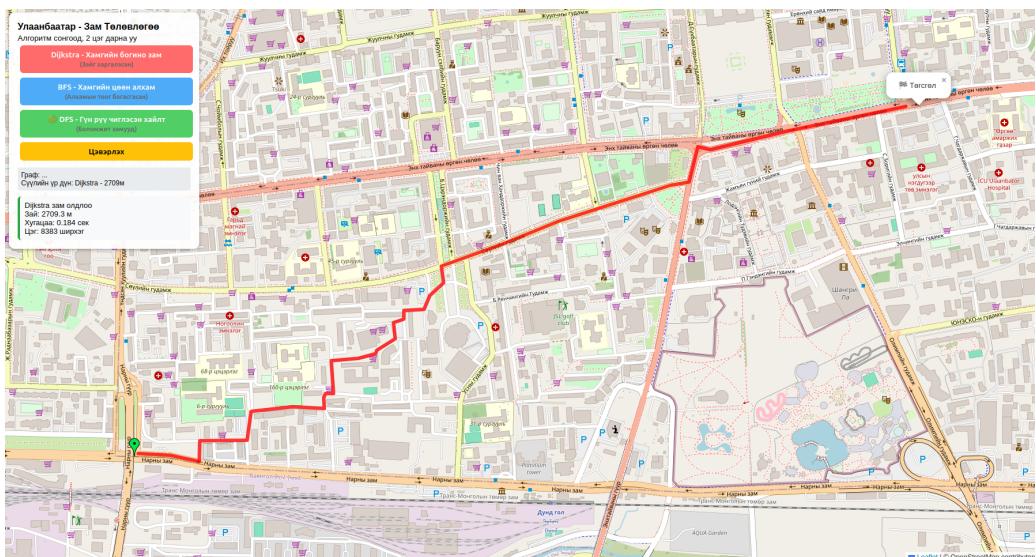
- Эхлэл, төгсгөлийн цэгийг сонгох
- Алгоритмыг сонгож зам тооцох
- Замыг газрын зураг дээр харуулах
- Статистик мэдээлэл (зай, хугацаа, цэгийн тоо) харуулах



Зураг 6: BFS алгоритмаар олсон зам



Зураг 7: DFS алгоритмаар олсон зам



Зураг 8: Dijkstra алгоритмаар олсон зам

7 Дүгнэлт

Энэхүү бие даалтын ажлын хүрээнд Улаанбаатар хотын газрын зургийг ашиглан графын хайлтын алгоритмуудыг хэрэгжүүлж, богино зам тооцох системийг амжилттай бүтээлээ. BFS, DFS, Dijkstra алгоритмуудын үндсэн онол, цаг хугацааны нарийн төвөгтэй байдал, санах ойн хэрэглээ, мөн давуу болон сул талуудыг судлав.

Туршилтын үр дүнгээс харахад:

- **Dijkstra** алгоритм нь зай тооцоололтой граф дээр хамгийн зөв богино замыг олдог бөгөөд гүйцэтгэл сайтай, үр дүнгийн нарийвчлал өндөр байна. Дундаж хугацаа бага, олсон замын урт нь бодит замын уртад хамгийн ойр байна.
- **BFS** алгоритм нь зай тооцоололгүй граф дээр богино замыг хурдан олох боломжтой боловч замын урт жингээр тооцогдох үед Dijkstra-тай харьцуулахад нарийвчлал бага байв.
- **DFS** алгоритм нь бүх боломжит замыг эрж хайж чаддаг ч гүнзгийрүүлсэн хайлт хийх үед цаг хугацаа, санах ойн хэрэглээ их, олсон зам нь заавал богино биш байсан.

Хавсралт

- OpenStreetMap
- BFS vs DFS vs Dijkstra - Baeldung
- Leaflet.js
- BFS and Dijkstra Shortest Path - GeeksforGeeks