ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл, Холбооны технологийн сургууль

БИЕ ДААЛТЫН АЖЛЫН ТАЙЛАН

Алгоритмын шинжилгээ ба зохиомж (F.CSM301)
2025-2026 оны хичээлийн жилийн намар

Лабораторийн цаг:	5 – 1
Хичээл заасан багш:	Д. Батмөнх
Бие даалтын ажил гүйцэтгэсэн:	Оюутан: Г. Баасандулам /B231960008/

2025 он

Агуулга

1. Хураангуй 3
2. Онол 3
1.1 Граф гэж юу вэ? 3
1.2 BFS (Breadth-First Search) алгоритм 3
1.3 DFS (Depth- First Search) алгоритм 4
1.4 Dijkstra алгоритм 6
3. Python хэрэгжилт 7
3.1 Хавтасны бүтэц 7
3.2 Системийн архитектур 7
3.3 Өгөгдлийн боловсруулалт 7
3.4 Графын бүтэц 8
3.5 Алгоритмуудын хэрэгжүүлэлт 8
3.6 Rest API 8

1. Хураангуй

Энэхүү бие даалтаар графын хайлтын алгоритмуудын (BFS, DFS болон Dijkstra) үндсэн санааг ойлгож, Улаанбаатар хотын газрын зургийг ашиглан богино зам тооцох даалгаврыг хэрэгжүүлэх болно. Графын хайлтын дараах гурван алгоритмыг ашиглан Улаанбаатар хот доторх дурын 2 цэгийн хооронд автомашинаар зорчиж болох боломжит замыг дараах байдлаар тооцно:

1. Хамгийн богино зам

2. Боломжит бүх зам

3. Хамгийн цөөн алхам

Улаанбаатар хотын газрын зургийн замын мэдээллийг OpenStreetMap (OSM) сангаас татаж авч ашиглана. Газрын зургийг боловсруулахад Python хэл дээр GeoPandas санг ашигласан.

2. Онол

Энэхүү бие даалтын хүрээнд граф хайлтын алгоритмуудыг ашиглан, Улаанбаатар хотын газрын зургийг ашиглан А цэгээс Б цэгийг хэрэглэгчээс аван тухайн цэг хооронд тооцоологдсон замыг олж дүрслэн харуулах зорилготой юм.

1.1 Граф гэж юу вэ?

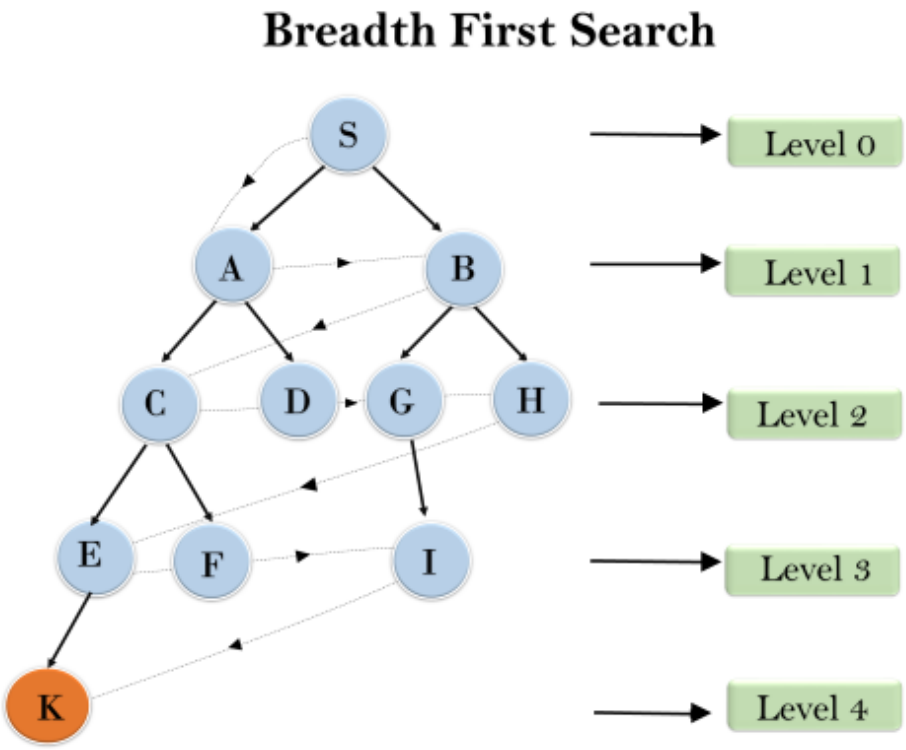
Компьютерын шинжлэх ухаан, математикийн олон салбарт граф (graph) нь объектуудын хоорондын харилцаа, холбоог дүрслэн үзүүлэх математик загвар юм. Товчхондоо хэлбэл, граф нь объектууд (vertex / node) болон тэдгээрийн холбоос (edge / link)-оос бүрддэг. Графын гол давуу тал нь объектуудын хоорондын харилцааг энгийн, ойлгомжтой байдлаар дүрслэхэд оршдог.

$$G = (V, E)$$

V – vertex буюу графын цэгүүдийн олонлог
E – edge буюу графын ирмэгүүдийн олонлог
Графын төрлүүд :

- Чиглэлгүй граф (Undirected graph) - Edge нь хоёр vertex-г чиглэлгүй холбодог. Өөрөөр хэлбэл, A ↔ B гэсэн холболт нь аль аль талд үйлчилнэ.
- Чиглэлтэй граф (Directed graph) - Edge нь нэг чиглэлтэй, A → B гэсэн холболт нь зөвхөн A-аас B рүү ажиллана.
- Жинтэй граф (Weighted graph) - Edge-д жин оноогдсон. Энэ жин нь холбооны үнэ цэнэ, зай, цаг хугацаа гэх мэт байж болно.

1.2 BFS (Breadth-First Search) алгоритм



Зураг 1. BFS алгоритм дүрслэл

BFS буюу өргөнөөр хайх алгоритм нь графыг түвшин бүрээр нь (level by level) хайдаг.

Эхний vertex-аас эхлээд хамгийн ойрын хөршүүдийг бүхлээр нь шалгаж, дараа нь тэдгээрийн хөршүүд рүү шилжих замаар ажилладаг. Ихэвчлэн queue ашигладаг. (FIFO)
Алгоритмын үе шат:

1. Эх vertex-г queue-д хийж, очсон гэж тэмдэглэнэ.
2. Queue-ооc vertex гаргаж, түүний бүх хөршийг шалгана.
3. Хөрш нь очсон бол queue-д нэмнэ.
4. Queue хоосроогүй бол 2-р алхам руу шилжинэ.

Давуу тал:

- Богино замыг олоход тохиромжтой (unweighted graph-д хамгийн богино зам).
- Level-wise буюу шатлалын бүтэц гаргаж өгдөг.

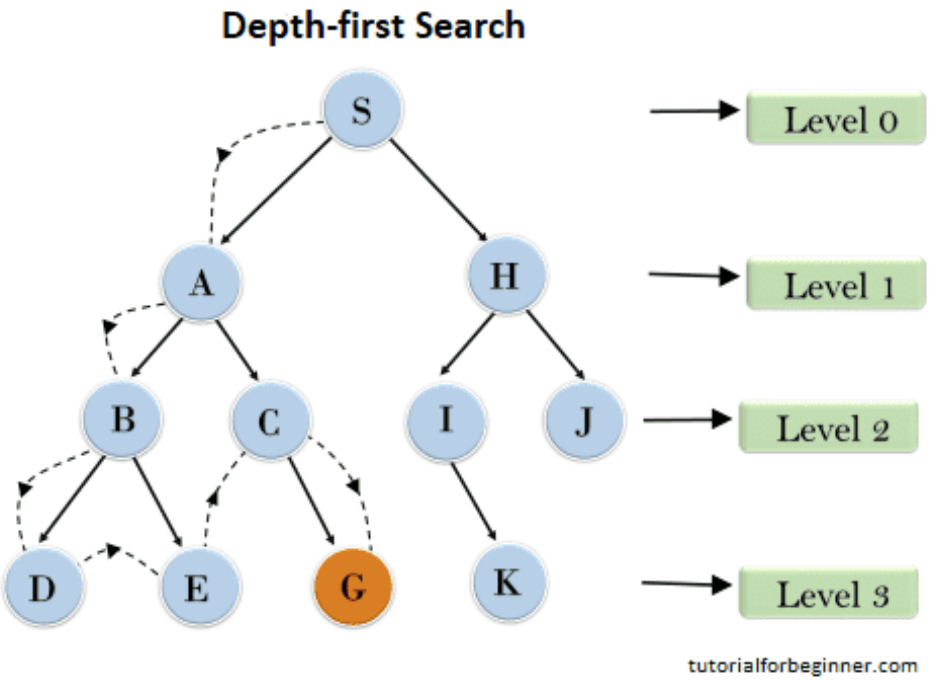
Сул тал:

- Том граф дээр санах ой их хэрэглэнэ.

Хугацаа: O(V + E)

Санах ой: O(V)

1.3 DFS (Depth- First Search) алгоритм



Зураг 2. DFS алгоритм дүрслэл

DFS буюу гүний хайх алгоритм нь графын нэг замаар аль болох гүн рүү нь орж, зам дуусах үед өөр зам руу ордог. Ихэвчлэн stack эсвэл recursive функц ашигладаг.

Алгоритмын үе шат:

1. Эх vertex-г очсон гэж тэмдэглэнэ.
2. Түүний хөршүүдийг нэг нэгээр нь шалгаж, очоогүй хөрш рүү орно.
3. Хөршүүд нь дууссан бол өмнөх vertex рүү буцаж, бусад хөршийг шалгана.
4. Бүх vertex-үүдийг судалтал үргэлжлүүлнэ.

Давуу тал:

- Санах ой бага хэрэглэдэг (stack ашигласнаар).
- Гүнзгий судалж болох тул замыг бүрэн судлахад тохиромжтой.

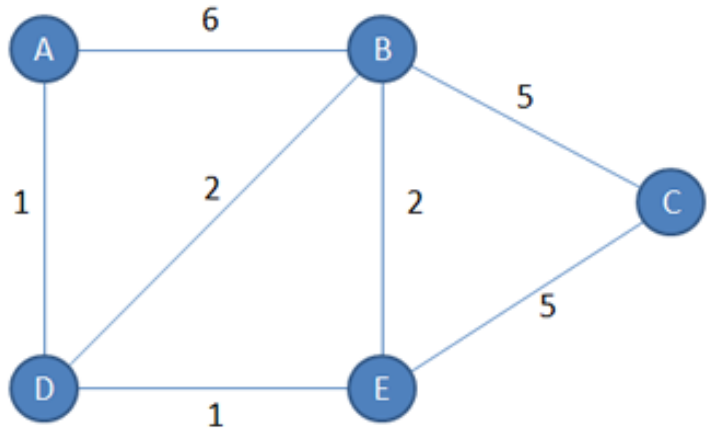
Сул тал:

- Богино зам олохгүй байж болно.

Хугацаа: O(V + E)

Санах ой: O(V)

1.4 Dijkstra алгоритм



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Зураг 3. Dijkstra алгоритм дүрслэл

Dijkstra алгоритм нь жинтэй графт эхлэх цэгээс бусад цэг рүү очих хамгийн богино замыг олдог. Алгоритм нь priority queue ашиглан хамгийн бага замтай vertex-г сонгон судална.
Алгоритмын үе шат:

1. Эх vertex-ийн замын жинг 0 болгож тэмдэглэнэ, бусдыг ∞ гэж тэмдэглэнэ.
2. Одоогийн vertex-ийн бүх хөршийг шалгаж, шинэ замын жинг шинэчилнэ.
3. Одоогийн vertex-г "очсон" гэж тэмдэглээд дараагийн хамгийн бага жинтэй vertex рүү шилжинэ.
4. Бүх vertex-үүдийн хамгийн бага замыг олох хүртлээ үргэлжлүүлнэ.

Давуу тал:

- Жинтэй граф дээр хамгийн богино замыг тодорхой гаргана.
- Энгийн граф болон замын системд хэрэглэж болно.

Сул тал:

- Negative weight ирмэгүүдтэй ажиллах боломжгүй (энэ тохиолдолд Bellman-Ford ашиглана).

Хугацаа: O(V^2)~O((V + E) logV)

Санах ой: O(V + E)

3. Python хэрэгжилт

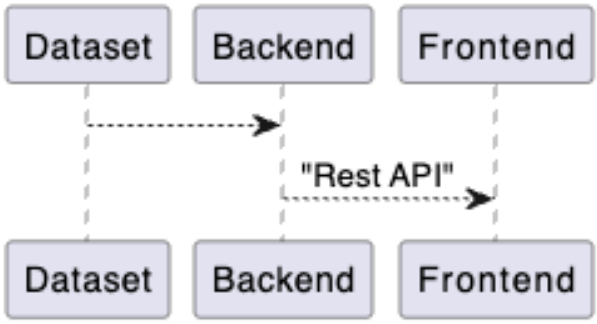
3.1 Хавтасны бүтэц

```
(base) macbookpro@Baaskas-MacBook-Pro biy daalt1 % tree
.
├── backend
│   ├── Read.md
│   ├── __pycache__
│   │   ├── app.cpython-310.pyc
│   │   ├── app.cpython-311.pyc
│   │   └── test.cpython-311.pyc
│   ├── app.py
│   ├── data
│   │   ├── gis_osm_roads_free_1.cpg
│   │   ├── gis_osm_roads_free_1.dbf
│   │   ├── gis_osm_roads_free_1.prj
│   │   ├── gis_osm_roads_free_1.shp
│   │   └── gis_osm_roads_free_1.shx
│   ├── requirements.txt
│   ├── settings.example.env
│   ├── test.py
│   └── uml.plantuml
└── frontend
    └── index.html

5 directories, 15 files
```

Зураг 4. Хавтасны бүтэц

3.2 Системийн архитектур



Зураг 5. Архитектур

3.3 Өгөгдлийн боловсруулалт

Энэ код нь .shp файл унших, CRS-д хөрвүүлэх , хоосон геометрийг арилгах , compound geometry-г задлах, индексийг цэвэрлэх гэсэн өгөгдөл боловсруулах алхмуудыг хийж байна.

```
def load_shapefile(self, shp_path: str):
    # .shp файл унших
    gdf = gpd.read_file(shp_path)
    # CRS-г EPSG:3857 стандарт руу хөрвүүлэх
```

```
if str(gdf.crs).lower() != "epsg:3857":
    gdf = gdf.to_crs(epsg=3857)
# Хоосон геометрийг арилгах ба multipolygon-г задлах
gdf = gdf[gdf.geometry.notnull()].explode(index_parts=False).reset_index(drop=True)
```

3.4 Графын бүтэц

Графын бүтэц нь node болон ирмэгүүдийг хадгалах
self.nodes = np.zeros((0,2), dtype=float) # node-ийн координат хадгалах массив
self.edges: Dict[int, List[Tuple[int,float,int]]] = {} # edge-үүд: (target_node, weight, edge_id)
self.geom_by_edge: Dict[int, LineString] = {} # ирмэгүүдийн геометр
def nearest_node(self, x:float, y:float): # Хамгийн ойрын node-г олох

3.5 Алгоритмуудын хэрэгжүүлэлт

1. Dijkstra (shortest path):
   def dijkstra(s:int, t:int): s -> t хамгийн богино замын жин
2. BFS (fewest hops):
   def bfs_fewest_hops(s:int, t:int, max_hops:int=10**9): хамгийн цөөхөн үсрэлттэй зам
3. DFS (all simple paths):
   def dfs_all_simple_paths(s:int, t:int, max_paths:int, max_hops:int, max_cost:float): бүх боломжит замуудыг хайж олно

3.6 Rest API

1. Dijkstra

   ```
   @app.get("/route/shortest")
   def route_shortest(start_lat: float, start_lng: float, end_lat: float, end_lng: float):
   start/end-г хамгийн ойрын node-д snap хийх
   Dijkstra алгоритм ажиллуулах
   GeoJSON-аар буцаах
   ```

2. BFS

   ```
   @app.get("/route/fewest_hops")
   def route_fewest_hops(start_lat: float, start_lng: float, end_lat: float, end_lng: float):
   BFS алгоритм ажиллуулна
   ```

3. DFS

   ```
   @app.post("/route/all")
   def route_all(req: AllPathsRequest):
   DFS алгоритм ашиглан бүх боломжит замуудыг хайна
   ```