Шинжлэх Ухаан Технологийн Их Сургууль
Мэдээлэл, Холбооны Технологийн Сургууль

F.CSM301 Алгоритмын шинжилгээ ба зохиомж

БИЕ ДААЛТ 1

Шалгасан багш: Д. Батмөнх
Гүйцэтгэсэн оюутан: Н.Хангал /B222270032/

Улаанбаатар хот
2025 он

Агуулга

Зургийн жагсаалт

Хүснэгтийн жагсаалт

1 Оршил

Энэхүү бие даалтын зорилго нь графын хайлтын гурван алгоритм болох **BFS**, **DFS**, болон **Dijkstra** алгоритмуудын үндсэн зарчмыг ойлгож, Улаанбаатар хотын **OpenStreetMap** (OSM) газрын зураг дээр хоёр цэгийн хооронд хамгийн тооцох систем боловсруулах явдал юм. Төслийн үндсэн зорилт нь:

1. Жингүй граф дээр хамгийн цөөн алхамтай зам олох (BFS).
2. Том граф дээр бүх боломжит замыг судлах (DFS / Backtracking).
3. Жинтэй граф дээр хамгийн богино зам тодорхойлох (Dijkstra).

Энэхүү систем нь замын navigation, тээвэрлэлт болон машин хөдөлгөөний optimisation зэрэг бодит асуудлыг шийдэхэд ашиглагдана.

2 Алгоритмуудын онол

2.1 BFS (Breadth-First Search)

BFS нь **өргөнөөр хайх алгоритм** юм. Эх зангилаанаас эхлэн хөрш зангилаануудыг шалгаж, дараа тэдний хөршүүд рүү үргэлжлүүлэн хайна. Жингүй граф дээр хамгийн цөөн алхамтай замыг олдог. **Алгоритмын логик:**

1. Эх зангилааг queue-т хийж эхэлнэ.
2. Queue-ас зангилаа аван, бүх хөршүүдийг шалгана.
3. Шинээр орж ирсэн зангилааг queue-т нэмнэ.
4. Зорилттой зангилаа олтол давталт үргэлжилнэ.

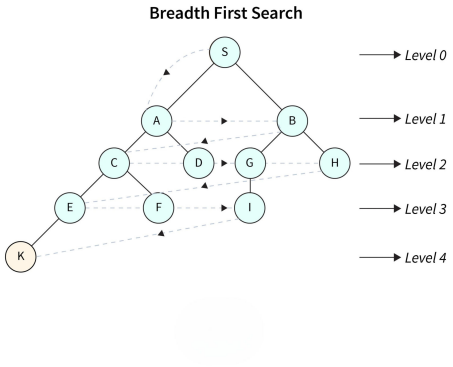
Давуу тал:

1. Хамгийн бага edge замыг олно.
2. Хялбар ойлгох, энгийн кодтой.

Сул тал:

1. Жингүй граф дээр л зөв ажиллана
2. Том граф дээр санах ой өндөр хэрэг болно

Жишээ: Эх зангилаа A, зорилт B бол BFS эхлээд A-аас 1 алхмын бүх хөршүүдийг шалгана. Хэрэв B олдохгүй бол 2 алхмын бүх хөршүүд рүү үргэлжлүүлэн явна. Ингэснээр хамгийн цөөн алхамтай зам олдоно.



Зураг 1: BFS алгоритмын жишээ

2.2 DFS (Depth-First Search)

DFS нь **гүнээр хайх алгоритм** юм. Эх зангилаанаас эхлэн нэг чиглэлд гүн уруу нэвтэрч, зам дуусахад эргэж буцаж, бусад замуудыг шалгана. Stack ашиглана. **Алгоритмын логик:**

1. Эх зангилааг stack-т нэмнэ.
2. Stack-ас зангилаа аван, нэгэн хөрш уруу гүн уруу явна.
3. Хэрэв зам дууссан бол буцаж, бусад боломжит замыг шалгана.
4. Зорилт олтол үргэлжилнэ.

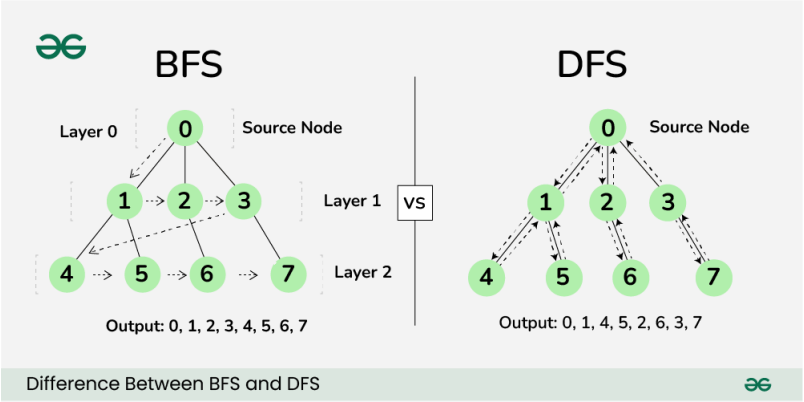
Давуу тал:

1. Хэрэгжүүлэхэд хялбар, хурдан хайлт хийнэ
2. Бүх замыг судлах боломжтой

Сул тал:

1. Замын урт, жинг үл харгалзана
2. Зарим тохиолдолд оновчгүй зам гарна

Жишээ: Эх зангилаа A-аас эхлэн хамгийн гүнрүү явах замыг судална. Хэрэв зам хаалттай бол буцаж, бусад салбарыг шалгана.



Зураг 2: BFS vs DFS алгоритмын харьцуулалт

2.3 Dijkstra алгоритм

Dijkstra нь **жинтэй граф дээр хамгийн бага зардалтай зам** олдог алгоритм юм. Priority Queue ашиглаж, хамгийн бага хуримтлагдсан жинтэй зангилааг үргэлж сонгон авдаг. **Алгоритмын логик:**

1. Эх зангилааг 0 жинтэй гэж тэмдэглэнэ, бусад зангилааг unlimited гэж тэмдэглэнэ.
2. Priority Queue-т эх зангилааг нэмнэ.
3. Queue-ас хамгийн бага жинтэй зангилааг аван, хөршүүдийн жинг шинэчилнэ.
4. Зорилт зангилаа олтол үргэлжилнэ.

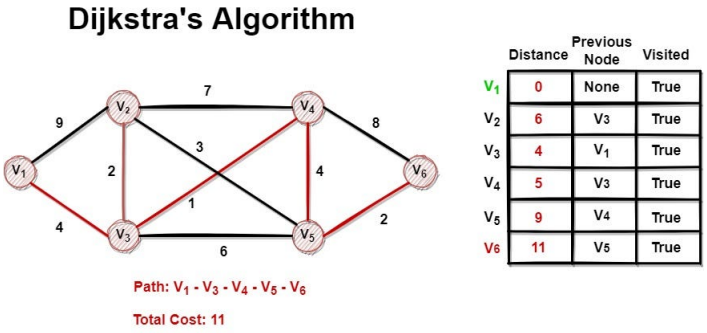
Давуу тал:

1. Жингийн хувьд хамгийн оновчтой үр дүн гаргана
2. Хамгийн богино замыг тодорхойлно

Сул тал:

1. BFS/DFS-ээс илүү тооцоолол шаардана

Жишээ: Эх зангилаа A, зорилт B бол Dijkstra нь бүх боломжит замын жинг харгалзан хамгийн бага нийт жинтэй замыг тооцно.



Зураг 3: Dijkstra

3 Замын өгөгдлийг боловсруулсан шат дамжлага

OpenStreetMap-аас татсан `gis_osm_roads_free_1.shp` файлыг **GeoPandas** сан ашиглан уншиж, дараах байдлаар граф үүсгэсэн: **Шат дамжлага:**

1. Shapefile-ийг уншиж замын геометр координатыг гарган авсан.
2. Polyline бүрийг ирмэг (edge) болгон задалж, зангилаа (node)-үүдийг тодорхойлсон.
3. Замын уртыг Haversine томьёогоор жин (weight) болгон тооцсон.
4. Графыг Python dictionary хэлбэрт хөрвүүлж `graph.pkl` болгон хадгалсан.

Схем:
Газрын зураг → Замын polyline → Зангилаа, ирмэг → Граф (dictionary)

4 Backend хэсэг — REST API

Flask сервер ашиглан BFS, DFS, Dijkstra алгоритмуудыг дараах байдлаар хэрэгжүүлсэн: **API логик:**

- Хэрэглэгч эх, зорилт координатыг өгнө
- Хүссэн алгоритмыг сонгоно (bfs, dfs, dijkstra)
- Server граф дээр замыг тооцож JSON хэлбэрээр буцаана

```
@app.route("/find_path", methods=["GET"])
def find_path():
    start = tuple(map(float, request.args.get("start").split(",")))
    goal = tuple(map(float, request.args.get("goal").split(",")))
    algo = request.args.get("algo", "dijkstra")

    start_node = nearest_node(graph, start)
    goal_node = nearest_node(graph, goal)

    if algo == "bfs":
        path = bfs(graph, start_node, goal_node)
    elif algo == "dfs":
        path = dfs(graph, start_node, goal_node)
    else:
        path, cost = dijkstra(graph, start_node, goal_node)

    return jsonify({"path": path})
```

5 Frontend хэсэг — Зам дүрслэл

Leaflet.js ашиглан газрын зураг дээр хоёр цэгийг сонгож, серверээс буцаасан замыг Polyline хэлбэрээр зурсан. API дуудлага нь дараах хэлбэртэй:

```
GET /find_path?start=47.918,106.917&goal=47.921,106.940&algo=dijkstra
```

Схем:

- Газрын зураг
- Эх, зорилт цэг
- Замын Polyline
- Зардал/урт (Dijkstra тооцоолсон бол)

6 Алгоритмуудын гүйцэтгэлийн харьцуулалт

Хүснэгт 1: BFS, DFS, Dijkstra алгоритмуудын харьцуулалт

Алгоритм	Цагийн төвөгшил	Санах ой	Замын чанар
BFS	$O(V+E)$	Өндөр	Цөөн алхамтай зам
DFS	$O(V+E)$	Бага	Тохиолдлын зам
Dijkstra	$O(E \log V)$	Дунд	Хамгийн богино жинтэй зам

Тайлбар:

- BFS: хурдан, гэхдээ жин харгалздаггүй
- DFS: бүх замыг шалгаж чадах боловч оновчгүй зам олж магадгүй
- Dijkstra: замын жинг харгалзан бодитой үр дүн гаргана

7 Үнэн зөв ажиллагааны баталгаа

Loop invariant: Dijkstra алгоритмд Priority Queue дотор байгаа бүх node нь үргэлж хамгийн бага хуримтлагдсан зардалтайгаар хадгалагддаг. **Induction:**

- Хэрэв k шатанд хамгийн бага зардалтай node тодорхойлогдсон бол
- $k+1$ шатанд мөн хамгийн бага зардалтай дараагийн node тодорхойлогдоно.

Ингэснээр алгоритм бүрэн гүйцэхэд хамгийн оновчтой зам олно.

8 Дүгнэлт

- **BFS**: хамгийн цөөн алхамтай замыг хурдан олдог.
- **DFS**: нэг чиглэлд гүн судалдаг тул оновчгүй зам гаргаж болдог.
- **Dijkstra** :замын жинг харгалзан хамгийн богино замыг бодитой гаргадаг.

Иймд бодит газрын зураг дээрх зам тооцоололд Dijkstra алгоритм хамгийн тохиромжтой гэж дүгнэв.

Ашигласан ном

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[2] "Openstreetmap data." `https://www.openstreetmap.org`, 2025.

[3] *GeoPandas Documentation*, 2025. `https://geopandas.org/`.