Шинжлэх Ухаан Технологийн Их Сургууль
Мэдээлэл, Холбооны Технологийн Сургууль

F.CSA12 Программ хангамжийн шаардлага ба шинжилгээ

Улаанбаатар хотод богино зам олох программ

Бие даалт №1

Шалгасан багш: Д.Батмөнх /F.CS21/

Гүйцэтгэсэн оюутан: Л.Ирмүүн /B232270021/

Улаанбаатар хот
2025 он

АГУУЛГА

1. БИЕ ДААЛТЫН ЗОРИЛГО

1.1 Программын зорилго

Энэхүү бие даалтын зорилго нь графын хайлтын алгоритмууд болох BFS, DFS болон Dijkstra алгоритмуудыг судалж, Улаанбаатар хотын газрын зураг дээр автомашин явах богино замыг тооцоолох систем хэрэгжүүлэхэд оршино. Эдгээр алгоритмууд нь графын өгөгдөлд дүн шинжилгээ хийх, замын холболтыг судлах, хамгийн оновчтой замыг олох зэрэг бодит хэрэглээнд өргөн хэрэглэгддэг.

2. АЛГОРИТМЫН ХЭРЭГЛЭЭ

2.1 BFS (Breadth-First Search) алгоритм

Үндсэн ажиллах зарчим:

BFS алгоритм нь графын жинг ашигладаггүй ба ажиллахдаа өгөгдсөн оройгоос эхлэн түүнтэй шууд холбогдсон бүх хөрш оройг 1-р түвшин гэж үзээд эхлээд шалгадаг, дараан дараагийн түвшний оройнууд болох 1-р түвшний хөрш оройнууд руу шилжээд шалгана. Энэ нь давхарга бүрийг эрэмбэлэн шалгадаг учраас хамгийн богино алхамтай замыг буцаадаг.

Ажиллах алхмууд:

1. Эхлэх оройг дараалалд (queue) оруулна.
2. Эхлэх оройн хөрш орой болох 1-р түвшний оройнуудыг дараалалд (queue) оруулна.
3. Дарааллаас эхэнд орж ирсэн оройг гаргаж, айлчлагдсан гэж тэмдэглэнэ.
4. Тэр оройтой холбогдсон бүх айлчлаагүй хөрш оройг дарааллын сүүлд нэмнэ.
5. Тухайн орой нь зорьж буй оройтой тэнцүү бол богино зам олдож давталт дуусна.
6. Үгүй бол дахин 2–5 алхмыг давтана, дараалал хоосорвол дуусна.

Давуу тал:

- Богино замыг баталгаатай олдог.
- Гүйцэтгэлийн хугацаа: O(|V| + |E|) — орой ба ирмэгийн тооноос хамаарна.

Сул тал:

Санах ой их ашигладаг, учир нь бүх түвшний оройг дарааллаар хадгалдаг.

2.2 Depth-First Search (DFS) алгоритм

Үндсэн ажиллах зарчим:

DFS буюу гүнээс эхлэх хайлт нь графын оройнуудыг аль болох гүн рүү дагаж судалдаг арга юм. Графын нэг оройгоос эхэлж, нэг замаар аль болох гүн рүү явна. Өмнөх BFS алгоритм нь эхлээд хэвтээ түвшний оройнуудыг шалгадаг бол DFS нь яг эсрэгээр нь эхлээд босоогоор түвшин бүрийг дамжиж гүн рүү шилжсээр цааш явах боломжгүй болсон үедээ буцаж бусад ирмэгийг шалгадаг

болохоор DFS-ийн эсрэг зарчмаар ажилладаг гэсэн үг. Энэ нь бүх замыг шалгах боломж олгодог боловч хамгийн богино замыг заавал олдоггүй.

Ажиллах алхмууд:

1. Эхлэх оройг stack ашиглан хадгалаад айлчилсан гэж тэмдэглэнэ.
2. Дараан stack-аас орой гаргаж (pop()), тухайн оройн хөрш оройнуудыг Stack руу нэмнэ.
3. Гарч ирсэн орой нь зорьж буй оройтой тэнцүү бол зам олдож давталт дуусна.
4. Дахин Stack-аас орой гаргаад (pop()) айлчилсан болгож тэмдэглээд, айлчлаагүй хөрш оройнуудыг stack руу оруулна.
5. Энэ үйлдэл бүх оройгоор дамжигдах хүртэл давтагдана.

Давуу тал:

- Графын бүх хэсгийг шалгахад тохиромжтой.
- Холбогдсон эсэх, цикл илрүүлэх, эрэмбэлэхэд ашигладаг.
- Гүйцэтгэлийн хугацаа: O(|V| + |E|) — BFS-тэй ижил, гэхдээ замын чиглэлээс хамаарна.

Сул тал:

- Богино замыг баталгаатай олдоггүй.
- Гүн рекурс үед стек дүүрэх эрсдэлтэй.

2.3 Dijkstra’s алгоритм

Үндсэн ажиллах зарчим:

Dijkstra нь зардалтай (жинтэй) графын нэг оройгоос бусад орой хүртэлх хамгийн бага зардалтай замыг олох алгоритм юм. Эхлэх оройгоос бусад бүх орой хүртэлх хамгийн бага зардлыг (жин) олж тогтоодог. Алгоритм нь хамгийн бага зардалтай оройг сонгон, түүгээр дамжих замыг үе шаттайгаар шинэчилдэг.

Ажиллах алхмууд:

1. Эхлэх оройн зардлыг 0, бусдыг ∞ гэж тохируулна.
2. Одоогийн хамгийн бага зардалтай оройг сонгоно.
3. Түүний хөрш оройнуудын зардлыг шинэчилнэ (newDist = currentDist + edgeWeight).
4. Тухайн хөрш орой дээр очих бага зардалтай оройг хүснэгтэд хадгална
5. Оройг “айлчилсан” гэж тэмдэглээд дараагийн хамгийн бага зардалтай орой руу шилжинэ.
6. Энэ үйлдэл бүх оройгоор дамжигдах хүртэл үргэлжилнэ.
7. Богино замаа олохын тулд эсрэг чиглэлээр явах ба хүснэгтээ ашиглан хүрэх ёстой оройг бага зардалтай өмнөх оройтой холбоно.
8. Мөн дахиад өмнөх багатай зардалтай оройг холбосоор байгаад үүсэх зам нь хамгийн бага зардалтай байна.

Давуу тал

- Жинтэй граф дээр богино замыг баталгаатай олно.
- Гүйцэтгэлийн хугацаа: O((|E| + |V|) log |V|)

Сул тал

- Сөрөг жинтэй ирмэг байвал буруу үр дүн өгдөг.
- Priority Queue ашиглах шаардлагатай.

2.4 Зам тооцоолох тохиромжтой алгоритм

Улаанбаатар хот доторх дурын 2 цэгийн хооронд автомашинаар зорчиж болох боломжит замыг дараах байдлаар тооцъё:

1. хамгийн богино зам
2. боломжит бүх зам
3. хамгийн цөөн алхам

Дээрх гурван шийдлийг олоход аль алгоритм нь тохиромжтой байгааг доорх хүснэгтэд харуулав.

Тооцоолох зам	Алгоритм	Шалтгаан
1. Хамгийн богино зам	Dijkstra алгоритм	Зам бүрийн зардал (урт эсвэл хугацаа) тооцож, хамгийн бага нийт зардалтай замыг гаргадаг. Жинтэй граф дээр богино зам олоход хамгийн оновчтой.
2. Боломжит бүх зам	DFS алгоритм	Гүнээс хайх зарчмаар эхлэлээс төгсгөл хүртэлх бүх боломжит маршрутуудыг олж чаддаг. Аль ч замаар бүрэн явж, бүх хувилбарыг шалгахад тохиромжтой.
3. Хамгийн цөөн алхамтай зам	BFS алгоритм	Түвшин дараалан хайдаг тул хамгийн бага ирмэг (алхам) бүхий замыг түрүүлж олно. Алхмын тоо хамгийн бага байх нөхцөлд BFS нь хамгийн үр дүнтэй.

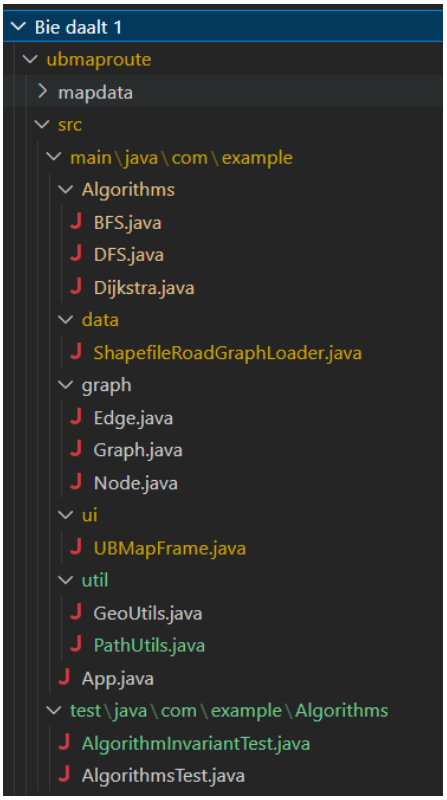
Хүснэгт 1. Зам тооцоолоход тохирох алгоритмууд

3. ПРОГРАММЫН АЖИЛЛАГАА БА ХЭРЭГЖҮҮЛЭЛТ

3.1 Төслийн ерөнхий бүтэц

Энэхүү бие даалтын зорилго нь Улаанбаатар хотын OSM shapefile замын өгөгдлөөс граф үүсгээд UI дээр эхлэл/төгсгөлийн цэг сонгон BFS, DFS, Dijkstra алгоритмаар замыг тооцох систем бүтээхэд оршино.

Төслийг Java хэл дээр хэрэгжүүлсэн бөгөөд дараах үндсэн бүтэцтэй.



Зураг 1. Төслийн бүтэц

Хавтас ба класс	**Үүрэг, зориулалт**
UI (User interface) - UBMapFrame	Газрын зургийг харуулах (OSM tiles), офлайн overlay (замын сүлжээ), route polyline зурах. Start/End цэг сонгох, алгоритм сонгох, “Route/ Clear” товчид хариу үйлдэл үзүүлэх.
Data - ShapefileRoadGraphLoader	GeoTools ашиглан UB bbox дох shapefile-ийн шугам геометруудыг уншиж граф үүсгэнэ. Атрибутууд: oneway, maxspeed, access, fclass зэргийг уншиж чиглэл, хурд, шүүлтүүр шийддэг.
Graph - Node, Edge, Graph	Граф үүсгэх суурь бүтцийн классууд.
Algorithms - BFS, DFS, Dijkstra	Үндсэн 3 алгоритмын классууд.
Util – GeoUtils, PathUtils	Газар зүйн тооцоолол (Haversine), bbox шалгалт, замын урт нийлбэр, замыг буцаан сэргээх.
Тест – AlgorithmInvariantTest, AlgorithmsTest	Алгоритмуудын ажиллагааг жижиг тестийн граф дээр ажиллуулж, нэг чигийн замыг зөв таньж ажиллаж буйг туршсан. Алгоритмуудын үнэн зөв ажиллагааг батлах induction ашиглан баталсан тестүүд.

3.2 Ашигласан технологи ба сангууд

Технологи ба сангууд	Үүрэг
Java	Гол программчлалын хэл
GeoTools	.shp, .dbf, .shx файлуудаас OSM замын өгөгдөл унших
JXMapViewer2	OSM дээр замын сүлжээ, маршрут, тэмдэглэгээ зурах Swing UI.
Spring Boot REST API	Алгоритмуудыг сервер талд ажиллуулж, frontend-д өгөгдөл дамжуулах

Хүснэгт 3. Технологи ба сангууд

OpenStreetMap (OSM)-оос татсан **shapefile** файлыг задлан уншиж, дараах мэдээллийг гаргаж авсан:

- **.shp** – замын геометрийн өгөгдөл:
  - Node (уулзвар)
  - Edge (замын ирмэг)
  - Weight (замын урт)
- **.dbf** – замын атрибутууд:
  - One-way (Нэг чигийн зам)
  - maxspeed (Хурдны хязгаар)
  - access (Автомашин нэвтрэх боломжтой эсэх)
  - fclass (Замын төрөл)
  - Bridge, tunnel (Гүүр, нүхэн гарч)
- **.prj** — координатын лавлах систем (CRS, ихэвчлэн WGS84) тодорхойлно.
- **.shx** — геометрийн индекс (уншилтын хурд).
- **.qix** — орон зайн индекс.
- **.cpg** — .dbf-ийн кодчиллын мэдээлэл

3.3 Зургийн файлын боловсруулалт

Энд Geofabrik-ийн shapefile-ээс өгөгдлийг уншиж, замын граф үүсгээд, UI дээр маршрут тооцон дүрсэлдэг. Доор газрын зургийн файлаас эхлээд хэрэглэгч 2 цэг оруулж үр дүнг буцаах хүртэлх алхмуудыг харуулав.

3.3.1 Өгөгдөл Уншилт

1. mapdata хавтсан доторх OSM-ийн замын shapefile багц (.shp/.dbf/.shx/.prj/.qix)-ийг ашиглана.
2. Гео сан: GeoTools ашиглан shapefile-ийн төрлийг нээж, замын геометр (LineString/MultiLineString) ба атрибутуудыг уншина.
3. Координат ба муж: Улаанбаатарын bbox (уртраг/өргөрөг)-оор шүүж, зөвхөн Улаанбаатар хотыг хязгаарласан талбайн шугамуудыг авна.

4. Атрибут ашиглалт:
   - **oneway**: нэг чигийн эсэх (yes/1/true - урсгалын дагуу, ‘-1’ - эсрэг чиглэл, бусад → хоёр чиглэл).
   - **maxspeed**: Дээд хурд ба байхгүй бол fclass-ын ангиллаас анхдагч хурд онооно (motorway, trunk, primary, …).
   - **access + fclass**: зөвхөн автомашинд тохирох шугамуудыг үлдээж, явган/дугуйн/хувийн нэвтрэх хязгаарлалтыг шүүнэ.

3.3.2 Өгөгдөл Урьдчилсан Боловсруулалт

5. Геометр задлалт: MultiLineString бүрийг LineString хэсгүүдэд салгаж, оройн (vertex) дарааллаар боловсруулна.
6. Давхардал бууруулах: Ойролцоо координатуудыг 1e−5 нарийвчлалд дугуйруулж нийлүүлснээр нэг уулзвар/зангилааг олон удаа үүсгэхээс сэргийлнэ.
7. Координатын тайлбар: WGS84 (уртраг=x, өргөрөг=y) гэж үзэж, метрийн тооцоололд Haversine хэрэглэнэ.

3.3.3 Граф үүсгэлт

8. Геометрийн цэг бүрээс оройг (node) бий болгоно (id, lat, lon).
9. Дараалсан зангилааны хооронд ирмэг (edge) тавина.
   - Oneway утгаас хамаарч нэг/хоёр чиглэлтэй ирмэгүүд бий болно.
   - Ирмэгийн жин (weight): = сегментийн урт / хурд (секунд).
     - Haversine томьёогоор метрээр уртыг тооцно.
     - Хурдыг maxspeed (км/ц) → м/с руу хөрвүүлнэ, байхгүй бол fclass-ын анхдагч.
10. Зангилаа бүрийн adjacency жагсаалт үүсгэж хөрш оройн бүтцийг хадгална.

3.3.4 Маршрут тооцоолох

11. BFS: Жин тооцдоггүй; давхарга дараалан хайж, хамгийн цөөн ирмэгтэй замыг олно. Зорилтод хүрмэгц эрт зогсоно.
12. DFS: Нэг салбарыг даган гүн рүү явж олдсон аль нэг замыг буцаана; богино зам баталгаатай биш. Бүх боломжит замыг жагсаах бол backtracking шаардлагатай.
13. Dijkstra: Ирмэгийн жин (сек) дээр тулгуурлаж хамгийн бага хугацаатай замыг олно. Сөрөг жингүй нөхцөлд баталгаатай.
14. Зам сэргээх: reconstruct функц ашиглаж бүх алгоритм prev хүснэгтээр зорилтоос эхлэл рүү ухарч, дарааллаар эргүүлэн маршрут гаргаж авна.

3.3.5 Маршрут дүрслэх

15. Олдсон зангилааны дарааллыг polyline болгож зураг дээр цэнхэр шугамаар зурна.
16. Эхлэх цэг (ногоон), хүрэх цэг (улаан) тэмдэглэнэ.
17. Хэрэглэгч эхлэх болон хүрэх цэгийг дахин сонгож, өөр алгоритм туршиж болно.

3.3.6 Үр дүн тооцоолох

18. Зай (км): Маршрут дагуух зангилаа хоорондын Haversine зайн нийлбэр.

19. Цаг (мин): Ирмэгийн жингийн нийлбэр (сек)-ийг минут руу хөрвүүлнэ (Dijkstra-д утга давхацна).
20. Гүйцэтгэлийн хугацаа (ms): System.currentTimeMillis() функц ашиглаж тухайн алгоритмын эхэлсэн ба ажиллаж дууссан хугацааны зөрүүг авна.
21. Санах ойг (MB) дараах аргаар тооцож гаргана:
    - тооцооллын өмнөх санах ойн хэмжээ: usedBefore = Runtime.totalMemory − Runtime.freeMemory (байт)
    - Тооцооллын дараах санах ойн хэмжээ: usedAfter = Runtime.totalMemory − Runtime.freeMemory (байт)
    - Хэрэглэсэн санах ой: used = max(0, usedAfter − usedBefore)
22. Программын дэлгэцийн баруун талын хайрцагт “Алгоритм, Зангилаа, Замын урт, Хугацаа, Ажиллах хугацаа, Санах ойн ашиглалт” харуулна.

4. ХЭРЭГЛЭГЧИЙН ИНТЕРФЕЙС

Төслийн хэрэглэгчийн интерфейс нь Swing дээр суурилсан бөгөөд JXMapViewer2 ашиглан Улаанбаатар хотын газрын зургийг харуулдаг. Дотоод бүтэц нь гурван давхаргатай:

1. OpenStreetMap-ийн суурь плит;
2. Офлайн замын сүлжээний overlay (shape агуулагдсан шугамууд)
3. Тооцоологдсон маршрут болон Start/End тэмдэглэгээ.

Эдгээрийг jxmapviewer.painter.CompoundPainter-аар нэгтгэж үзүүлдэг тул замын шугам, тэмдэглэгээ, маршрут нь давхаргын дарааллаар зөв харуулна.

Газрын зурагтай харилцахад JXMapViewer2-ийн координатын хөрвүүлэлтийг ашиглаж дэлгэцийн цэгийг газарзүйн өргөрөг/уртрагаар илэрхийлнэ. Хэрэглэгч газрын зураг дээр цэг дарж эхлэх (Start) ба төгсгөх (End) байршлаа тодорхойлно. Программ тухайн цэгийн координатыг граф дахь хамгийн ойрын зангилаатай холбож, алгоритмаар маршрут тооцоод polyline хэлбэрээр зурдаг.

UI-ийн урсгал дараах байдлаар явагдана:

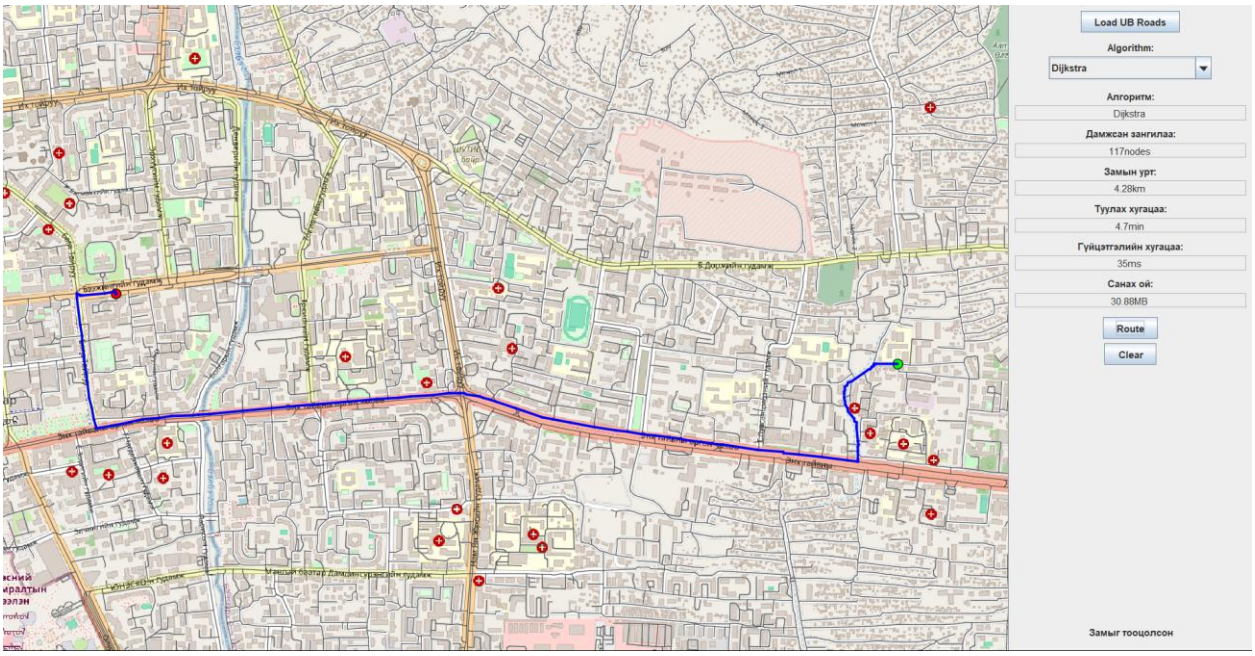
1. “**Load UB Roads**” дарснаар shapefile уншиж, UB хотын хязгаар дахь замуудыг саарал overlay болгон зурна.
2. “**Route**” дарснаар сонгосон алгоритм ажиллаж, үр дүнгийн polyline-г зураг дээр нэмнэ.
3. “**Clear**” нь өмнөх Start/End болон маршрутыг арилгана. Газрын зураг пан, zoom хийхэд UB хотын хилээс гадагш гарахаас сэргийлсэн хязгаарлалттай.

Алгоритмууд дараах зарчмаар ажиллана. **BFS** нь хамгийн цөөн ирмэгтэй замыг олдог бол **DFS** бүх боломжит маршрутыг хайдаг. **Dijkstra** зам бүрийн жинг урт, хурдны хязгаар зэргээр тооцож хамгийн бага хугацаатай замыг гаргадаг. Oneway, access, fclass зэрэг шинжийг харгалзсан тул бодит хөдөлгөөний дүрмийг хэсэгчлэн тусгасан.

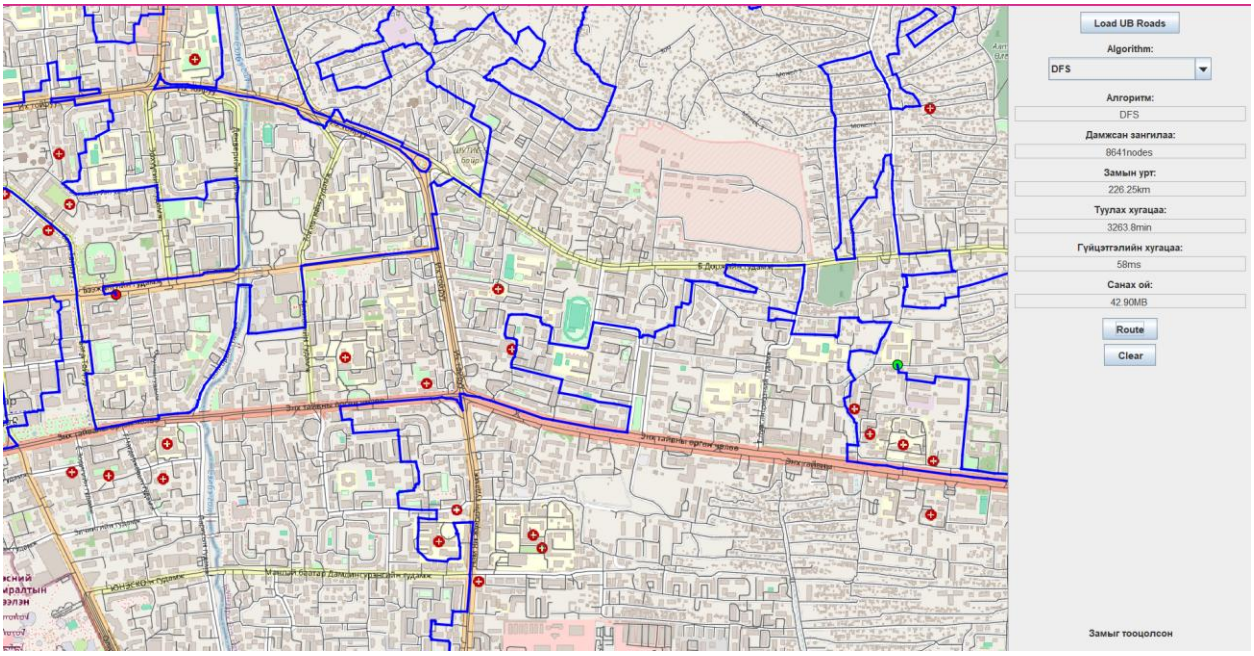
Маршрут тооцоолсны дараа баруун талын самбарт алгоритмын нэр, зангилааны тоо, замын урт, тооцоолсон хугацаа, гүйцэтгэлийн хугацаа, санах ойн хэрэглээ зэргийг харуулдаг.

Доор программын ажиллагааг үндсэн 3 алгоритм дээр харуулав.

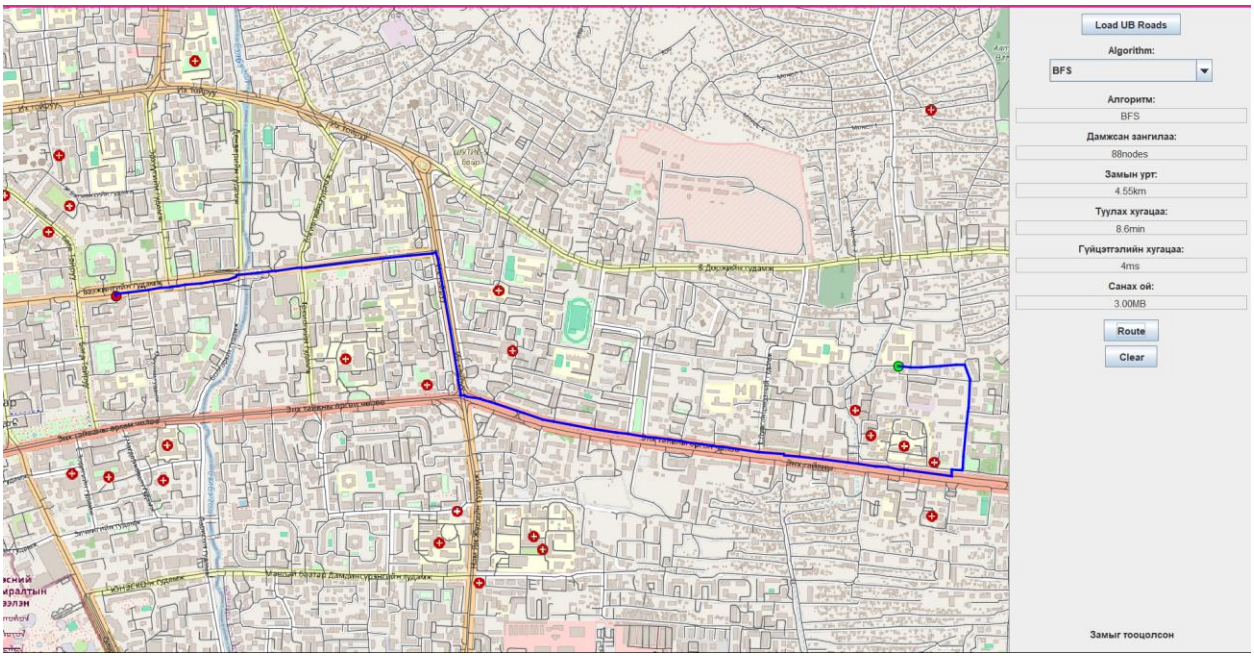
Эхлэх цэг (ногоон) – ШУТИС-ийн 6-р байр
Хүрэх цэг (улаан цэг) – ШУТИС-ийн 2-р байр



Зураг 2. Dijkstra алгоритмын ажиллагаа



Зураг 3. DFS алгоритмын ажиллагаа



Зураг 4. BFS алгоритмын ажиллагаа

5. АЛГОРИТМЫН ГҮЙЦЭТГЭЛИЙН ШИНЖИЛГЭЭ

Энэ хэсэгт гурван алгоритмыг (BFS, DFS, Dijkstra) ижил оролттой нөхцөлд тус бүрийг **5 удаа** ажиллуулж, гүйцэтгэлийн хугацаа ба санах ойн хэрэглээг хэмжсэн үр дүнг харьцуулан шинжилсэн болно.

Шинжилгээний зорилго нь алгоритм бүрийн хугацааны тогтвортой байдал, нөөцийн хэрэглээ, болон онолын төвөгшилтэй хэр зэрэг нийцэж байгааг тодорхойлох юм.

5.1 Туршилтын орчны мэдээлэл

Үзүүлэлт	Утга
Хөгжүүлэлтийн орчин	VSCode
OS	Windows 11 (64-bit)
RAM	16 GB
Процессор	Intel Core i5-13450HX (2.4 GHz)
Замын өгөгдөл	gis_osm_roads_free_1.shp (УБ хот)
Графын хэмжээ	254365 орой, 531662 ирмэг

Хүснэгт 4. Орчны үзүүлэлт

5.2 Туршилтын үр дүнгийн хүснэгт

Алгоритм	Туршилт №	Хугацаа (ms)	Санах ой (MB)	Дундаж хугацаа (ms)	Дундаж санах ой (MB)
BFS	1	4	4	3.4	3.998
	2	3	4		
	3	3	4		
	4	4	3.99		
	5	3	4		
DFS	1	53	47.71	43	46.26
	2	45	48.46		
	3	40	45.07		
	4	38	44.93		
	5	39	45.14		
Dijkstra	1	21	35.90	21.2	35.16
	2	22	35.04		
	3	21	35.12		
	4	22	35.23		
	5	20	34.55		

Хүснэгт 5. Үр дүн

5.3 Харьцуулсан шинжилгээ

BFS: Хамгийн бага хугацаа шаардаж, хурдан ажилласан. Энэ нь жигд жинтэй граф дээр богино замыг давхарга дараалан хайдагтай холбоотой.
Хурд хамгийн сайн, санах ой дунд зэргийн хэрэглээтэй.

DFS:
Харьцангуй илүү хугацаа зарцуулсан бөгөөд бүх боломжит замыг шалгадаг учир CPU-ийн ачаалал нэмэгдсэн.
Гүйцэтгэл дундаж, санах ой бага хэрэглэдэг.

Dijkstra:
Хамгийн нарийн үр дүн өгсөн боловч хамгийн их хугацаа зарцуулсан. Priority Queue ба жинтэй ирмэгийн тооцооллоос шалтгаалан илүү нөөц хэрэглэдэг.
Оновчтой үр дүн, гэхдээ хамгийн их хугацаа, дунд зэрэг санах ой.

6. ТЕСТ БА БАТАЛГААЖУУЛАЛТ

6.1 Зорилго

Энэхүү тестийн зорилго нь маршрут тооцоолох алгоритмууд (BFS, DFS, Dijkstra)-ын үнэн зөв ажиллагааг баталгаажуулах юм.

Тестүүд дараах логик шинж чанаруудыг шалгана:

- BFS: ирмэгийн тоогоор хамгийн цөөн алхамтай (hop) замыг олох.
- DFS: хүрч болох зорилтот цэг рүү зам олж чадаж байгаа эсэх (reachability).
- Dijkstra: сөрөг жингүй нөхцөлд хамгийн бага нийт жинтэй (манай төсөлд хугацаа) замыг олох.

Нэгж тестүүд: Хиймэл жижиг графууд дээр алгоритм бүрийн шийдвэр зөв гарч буй эсэхийг шалгана. (жишээ нь hop-ын тоо, нийт жингийн нийлбэр).

Инвариант / Индукцийн тестүүд: Алгоритмын онолын зөв ажиллагааг (давталтын инвариант, математик индукц) шалгана.

6.2 AlgorithmsTest.java — Нэгж тестийн баталгаа

BFS — богино алхамтай зам:
A→B→C болон A→D→C гэсэн хоёр замтай граф үүсгээд BFS нь 3 зангилаатай A-B-C замыг буцаадаг болохыг баталгаажуулна.

Dijkstra — хамгийн бага хугацаа:
Ирмэгийн жинг секунд гэж үзвэл A→B→C (1+1) нь A→D→C (1+10)-оос бага тул Dijkstra зөв замыг сонгож байгаа эсэхийг шалгана.
Нийт жинг pathWeightSeconds() функцээр тооцож харьцуулна.

DFS — зам олдож байгаа эсэх:
Хүрч болох зорилтод DFS үр дүнтэй зам буцааж байгаа эсэхийг шалгана.

One-way — чиглэлийн хязгаар:
E→F чиглэл зөв ажиллаж, F→E буцааж явах боломжгүй (ирмэг байхгүй) гэдгийг BFS-ийн үр дүнгээр батална.

Эдгээр тестүүд нь графын чиглэл, жин, hop зэрэг гол шалгуур үзүүлэлтүүдийг баталгаажуулдаг.

6.3 AlgorithmInvariantTest.java — Онолын баталгааны тестүүд

6.3.1 BFS — bfsPathGraphInduction

Үндсэн санаа:
n зангилаатай шулуун граф (A—B—…—Z) дээр BFS алгоритмаар эхлэлээс төгсгөл хүртэл хайлт хийхэд hop-ын тоо үргэлж **n−1** байх ёстой.

Тохируулга:
n = 2–12 хүртэл зангилааг дараалан (1 ↔ 2 ↔ 3 ↔ … ↔ n) холбосон шугаман граф үүсгэнэ. Ирмэг бүрийн жин 1 боловч BFS жинг тооцохгүй, зөвхөн ирмэгийн тоогоор (алхам) зам хайна.

Суурь тохиолдол:
n = 2 үед ганц ирмэгтэй тул hop = 1 байх ёстой.

Индукцийн алхам:
n = k үед hop = k−1 байвал, шинэ зангилаа (k+1)-г төгсгөлд нэмж холбох үед цорын ганц шинэ зам нэмэгдэнэ. Энэ шинэ зам өмнөхөөсөө 1 ирмэгээр уртассан тул hop = (k+1)−1 болж индукцийн нөхцөл биелнэ.

Давталтын инвариант:
BFS мод дотор түвшин (level) нь замын дагуу алхам тутамд яг **1-ээр өсөх ёстой**. Үүнийг туслах levelsFrom() функц ашиглан баталгаажуулдаг — хөрш зангилаануудын түвшний ялгаа яг 1 байх ёстой.

Баталгаа:

- Буцаасан замын ирмэгийн тоо n−1 байна.
- Түвшний инвариант нь BFS алгоритм “давхарга дараалан” хайж байгааг нотолдог.

6.3.2 Dijkstra — dijkstraLineGraphInduction

Үндсэн санаа:
Нэгж жинтэй шулуун граф дээр Dijkstra алгоритм үргэлж хамгийн бага нийт жинтэй замыг олдог бөгөөд нийт жин нь **n−1** байх ёстой.

Тохируулга:
n = 2–20 хүртэл шулуун граф үүсгэж, ирмэг бүрийн жинг 1 гэж өгнө.

Суурь тохиолдол:
n = 2 үед замын нийт жин 1 байна.

Индукцийн алхам:
n = k үед хамгийн бага жин k−1 байвал, шинэ зангилаа (k+1)-г төгсгөлд нэмэхэд замд заавал 1 шинэ ирмэг орж ирнэ → нийт жин (k+1)−1 болно.

Давталтын инвариант:
Dijkstra алгоритмын Priority Queue (PQ)-оос сугалж буй **баталгаажсан** зангилааны зай (distance) тухайн мөчид үргэлж хамгийн бага байх ёстой. Шулуун граф дээр энэ инвариант нь замын дарааллыг яг баримталж, зангилаа бүрийн жин 1-ээр өсөж байгааг батална.

Туслах функц:
sumOfWeight() функц олдсон замын ирмэгүүдийн жинг нэмэж, нэгж жинтэй граф дээр нийт жин **n−1** байх ёстойг баталгаажуулдаг.

Баталгаа:

- Замын зангилааны тоо n, нийт жингийн нийлбэр n−1 байна.

- Инвариант нь Dijkstra алгоритм хамгийн бага жинтэй замыг баталгаатайгаар сонгож байгааг нотолдог.

6.3.3 DFS — dfsLineGraphInduction

Үндсэн санаа:
Шулуун граф дээр эхлэлээс төгсгөл хүртэл зөвхөн нэг зам байдаг тул DFS алгоритм үргэлж **n−1** ирмэгтэй замыг олно.

Тохируулга:
n = 2–12 хүртэл зангилааг дараалан холбосон шулуун граф үүсгэн, DFS алгоритмаар зам хайна.

Суурь тохиолдол:
n = 2 үед ганц ирмэгтэй тул замын урт 1 байна.

Индукцийн алхам:
n = k үед замын урт k−1 байвал, шинэ зангилаа (k+1)-г төгсгөлд нэмж холбох үед ганц шинэ ирмэг нэмэгдэж, замын урт (k+1)−1 болно.

Баталгаа:

- Олдсон замын ирмэгийн тоо үргэлж n−1 байна.
- Энэ нь DFS алгоритмын “зам олдог” буюу **reachability** шинжийг индукцийн аргаар баталж буй хэлбэр юм.
- Богино замын шаардлага энд тавигдахгүй, учир нь шулуун граф дээр богино зам = цорын ганц зам гэсэн үг юм.

7. ДҮГНЭЛТ

Энэхүү бие даалтын хүрээнд Улаанбаатар хотын замын өгөгдлийг ашиглан BFS, DFS, Dijkstra алгоритмуудыг хэрэгжүүлж, тэдгээрийн гүйцэтгэл болон зөв ажиллагааг туршиж баталгаажууллаа.

BFS нь хамгийн цөөн алхамтай замыг хурдан хугацаанд олох чадвартай, DFS нь бүх боломжит замыг бүрэн судалж чаддаг, Dijkstra нь жинтэй замын хувьд хамгийн оновчтой богино замыг олдог болох нь практик болон онолын туршилтаар батлагдсан.

Гүйцэтгэлийн хувьд BFS хамгийн хурдан, Dijkstra хамгийн нарийн үр дүнтэй боловч илүү хугацаа шаардсан. DFS нь гүн хайлтын онцлогоос шалтгаалан дундаж гүйцэтгэлтэй байсан.

Тест ба инвариантийн баталгаануудын үр дүнгээр алгоритмууд онолын дагуу зөв ажиллаж буй нь нотлогдсон бөгөөд систем нь Улаанбаатар хотын зам сүлжээний богино замыг үр дүнтэй тооцоолж, газрын зураг хэлбэрээр дүрслэх боломжийг бүрдүүлсэн.

8. ХАВСРАЛТ

[1] “Dijkstra's shortest path algorithm in Java using PriorityQueue,” 11 7 2025. [Холбогдсон]. Available: https://www.geeksforgeeks.org/java/dijkstras-shortest-path-algorithm-in-java-using-priorityqueue/.

[2] “Breadth First Search or BFS for a Graph,” [Холбогдсон]. Available: https://www.geeksforgeeks.org/dsa/breadth-first-search-or-bfs-for-a-graph/.

[3] “Depth First Search or DFS for a Graph,” 3 10 2025. [Холбогдсон]. Available: https://www.geeksforgeeks.org/dsa/depth-first-search-or-dfs-for-a-graph/.

[4] “https://download.geofabrik.de/asia/mongolia.html,” 18 10 2025. [Холбогдсон]. Available: https://download.geofabrik.de/asia/mongolia.html.

[5] Reducible, “Depth First Search (DFS) Explained: Algorithm, Examples, and Code,” 6 July 2020. [Холбогдсон]. Available: https://www.youtube.com/watch?v=PMMc4VsIacU.

[6] WilliamFiset, “Dijkstras Shortest Path Algorithm Explained | With Example | Graph Theory,” 16 September 2020. [Холбогдсон]. Available: https://www.youtube.com/watch?v=bZkzH5x0SKU.

[7] WilliamFiset, “Breadth First Search Algorithm | Shortest Path | Graph Theory,” 2 April 2018. [Холбогдсон]. Available: https://www.youtube.com/watch?v=oDqjPvD54Ss.

[8] “aymanElakwah/Open-Street-Map-Navigation,” 4 August 2021. [Холбогдсон]. Available: https://github.com/aymanElakwah/Open-Street-Map-Navigation.