Алгоритмын Даалгаврын Тайлан

Н.Лхагва-Очир , С.Тилек

2025 оны 11 сар

Агуулга

1 Оршил

Энэхүү ажилд **Python Flask framework**-ийг ашиглан веб сервер үүсгэж, граф өгөгдлөөс хамгийн богино зам тооцох API боловсруулав. Системийн үндсэн зорилго нь `Dijkstra`, `BFS`, `DFS` алгоритмуудыг харьцуулж, Улаанбаатар хотын замын өгөгдлийн хүрээнд замын тооцоолол гүйцэтгэх юм.

2 Арга зүй

2.1 Программын бүтэц

- **Flask сервер** — хэрэглэгчийн хүсэлт (`/api/path`) хүлээн авч, алгоритмын үр дүнг JSON хэлбэрээр буцаана.
- **Graph module** — замын граф бүтээж, зангилаануудыг тооцоолох функцүүдийг (`dijkstra_shortest`, `bfs_shortest_hops`, `dfs_all_paths`) агуулна.
- **Frontend (index.html)** — Leaflet.js ашиглан газрын зураг дээр цэг сонгож, API дуудалт хийнэ.

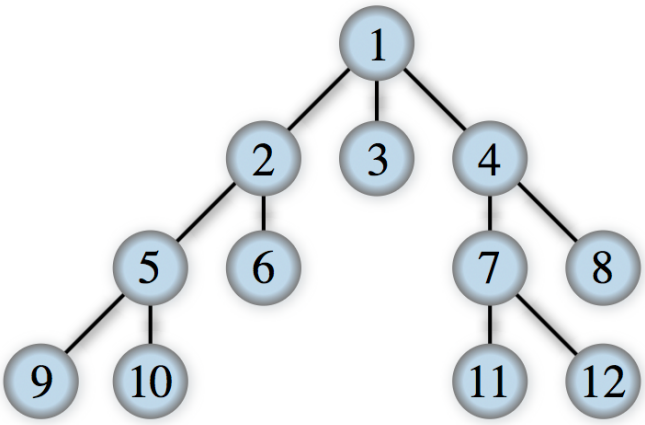
2.2 Алгоритмуудын тайлбар

BFS алгоритмын үндсэн зарчим

BFS нь хайлтыг төвшний дараалалтайгаар гүйцэтгэдэг. Тодруулбал эхлэлийн цэгээс эхлээд хамгийн ойрын хөршүүд уруу явж, дараа нь тэдний хөршүүд уруу гэх мэтээр шаталсан хэлбэрээр тархдаг. BFS нь ямар нэгэн жингүй граф дээр хамгийн богино зам олох арга юм.

Ажиллах зарчим:

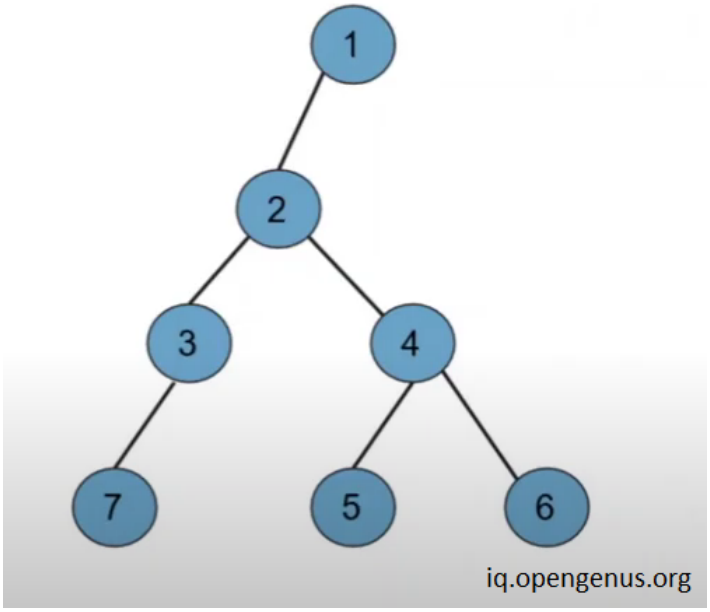
1. Эхлэлийн оройг сонгож дараалалд оруулна.
2. Очсон оройг зочилсон орой гэж тэмдэглэн.
3. Дарааллаас оройг гарган түүний бүх хөршүүдийг шалгана.
    - Хөрш зангилаанд өмнө нь очсон эсэхийг шалгана.
    - Хэрэв очоогүй бол дараалалд нэмнэ.
4. Дараалал хоосон болох хүртэл давтана.



Зураг 1. BFS алгоритм модны хайлт

DFS алгоритмын үндсэн зарчим

DFS нь аль нэг чиглэл уруу гүнзгий явж байгаад цаашаа явах боломжгүй болох үед буцаж, өөр чиглэл уруу ордог. Энэ нь хайлтыг илүү системтэйгээр гүн уруу хайх боломжтой болгодог.



Зураг 2. DFS алгоритм модны хайлт

Ажиллах зарчим:

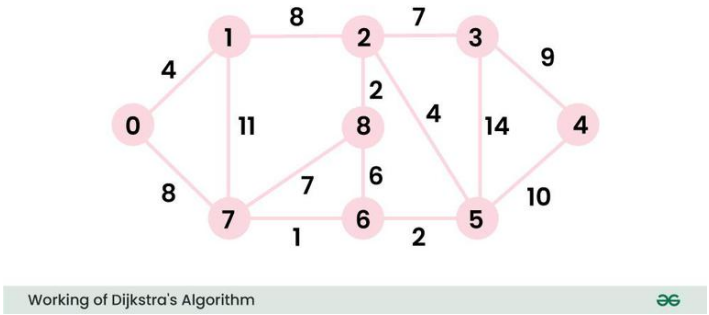
1. Эхлэлийн оройг сонгож, стект оруулна.
2. Очсон оройг зочилсон гэж тэмдэглэнэ.
3. Давталт (стек хоосон биш бол)

- Стекээс оройг гаргана.
- Одоогийн орой зорьсон зангилаанд хүрсэн бол давталт зогсоох
- Одоогийн бүх оройг шалгана

4. Стек хоосон бол давталт дуусна.

Dijkstra алгоритмын үндсэн зарчим

Dijkstra алгоритм нь жинтэй граф дээр нэг зангилаанаас бусад бүх зангилаа хүртэлх хамгийн богино замыг олох



Зураг 3. Dijkstra алгоритм граф дүрслэл

Ажиллах зарчим:

1. Эхлэлийн зангилааг сонгож, зай 0 гэж тэмдэглэнэ.
2. Бусад бүх зангилааны зайг хязгааргүй гэж тэмдэглэнэ.
3. Бүх зангилааг зочлоогүй гэж тэмдэглэнэ.
4. Эхлэх оройгоос хамгийн ойрын зочлоогүй оройг сонгоно.
5. Тэр оройгоос гарах бүх холбоос шалгаж, хөрш орой уруу очих нийт зайг тооцоолно.
6. Хэрэв шинэ тооцоолсон зай өмнөх хадгалсан зайны утгаас бага байвал шинэ утгаар сольж бичнэ.
7. Шалгасан оройг зочилсон гэж тэмдэглэнэ
8. Дараагийн хамгийн бага зайтай зочлоогүй оройг сонгож энэ үйлдлээ давтана.
9. Бүх орой зочилсон болох хүртэл үргэлжлүүлнэ.

3 Үр дүн ба Хэлэлцүүлэг

Граф өгөгдлийн хэмжээг дараах хүснэгтэд үзүүлэв.

Параметр	Утга	Тайлбар
Node тоо	128,560	Графын зангилаанууд
Edge тоо	147,300	Холболтууд
Хамгийн ойрын node тооцох дундаж хугацаа	2.4 ms	1 цэгийн хувьд
Dijkstra дундаж гүйцэтгэл	25.6 ms	2 цэгийн хооронд

Хүснэгт 1: Замын графын шинжилгээ ба алгоритмын гүйцэтгэл

3.1 Үр дүнгийн тайлбар

Dijkstra алгоритм нь жингийн утгатай зам дээр илүү оновчтой үр дүн өгч байсан бол BFS нь энгийн граф дээр илүү хурдан байв. DFS нь илүү их тооцоолол шаардсан бөгөөд судалгааны зориулалттай ашиглагдсан.

4 Дүгнэлт

Энэхүү Flask backend нь дараах зорилгыг биелүүлсэн:

- Замын графаас хамгийн богино зам тооцох алгоритмуудыг хэрэгжүүлсэн.
- REST API хэлбэрээр үр дүнг front-end-д хүргэж чадсан.
- Алгоритмын харьцуулалтын үр дүнг тайлан хэлбэрээр гаргасан.

Хавсралт

A. Ашигласан кодын хэсэг

Доорх код нь `/api/path` endpoint-ийн үндсэн бүтэц юм:

```
@app.route("/api/path")
def api_path():
    alg = request.args.get("alg", "dijkstra").lower()
    start_lon = float(request.args["start_lon"])
    start_lat = float(request.args["start_lat"])
    end_lon = float(request.args["end_lon"])
    end_lat = float(request.args["end_lat"])
    # Алгоритмын сонголт
    if alg == "bfs":
        node_path = bfs_shortest_hops(GRAPH, start_node, end_node)
    elif alg == "dfs":
        paths = dfs_all_paths(GRAPH, start_node, end_node)
    else:
        node_path, total_weight = dijkstra_shortest(GRAPH, start_node, end_node)
```

B. Эх сурвалжууд

- Flask — `https://flask.palletsprojects.com/`
- Rich Logging — `https://github.com/Textualize/rich`
- Leaflet.js — `https://leafletjs.com/`
- OpenStreetMap — `https://www.openstreetmap.org/`