ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ
Мэдээлэл, Холбооны Технологийн Сургууль

Бие Даалт 1

F.CSM301 Алгоритмын шинжилгээ ба зохиомж

Шалгасан багш: Д. Батмөнх
Гүйцэтгэсэн оюутан: B232270024 О. Дэлгэрмаа

Улаанбаатар хот

Агуулга

1 Танилцуулга.....3
2 Үндсэн нэр ,томьёо.....3
2.1 Хамаарах сангууд ба бүрэлдэхүүн хэсгүүд.....3
3 Алгоритмуудын тайлбар.....4
3.1 Breadth-First Search (BFS).....4
3.2 . Depth-First Search (DFS).....5
3.3 Dijkstra.....7
4 Дүгнэлт.....9

1 Танилцуулга

Энэ бие даалтаар Улаанбаатар хотын замын сүлжээ буюу газрын зургийг дүрсэлж, хэрэглэгчийн сонгосон эхлэл, төгсгөлийн цэгийн хоорондох явах боломжтой автомашины замыг хайж бөгөөд, үүнд гурван төрлийн алгоритм Breadth-First Search (BFS), Depth-First Search (DFS), болон Dijkstra ашиглан харьцуулалт хийж ажиллагааных нь талаарх ойлголт мэдлэг авна..

Энд GeoPandas санг ашиглан shapefile хэлбэрийн замын мэдээллийг уншиж, замуудыг LineString геометрээр дүрсэлж, эдгээрийг график болгон хувиргадаг. Хэрэглэгчээс эхлэл ба төгсгөлийн координат авсны дараа систем хамгийн ойр оройг тодорхойлж, сонгосон алгоритмаар замыг тооцоолно.

Программ нь Flask ашиглан веб сервер ажиллуулж, дараах REST API-уудыг гаргадаг:

- /api/bfs – BFS алгоритмаар зам тооцох
- /api/dfs – DFS алгоритмаар зам тооцох
- /api/dijkstra – Dijkstra алгоритмаар хамгийн богино зам тооцох

2 Үндсэн нэр ,томьёо

2.1 Хамаарах сангууд ба бүрэлдэхүүн хэсгүүд

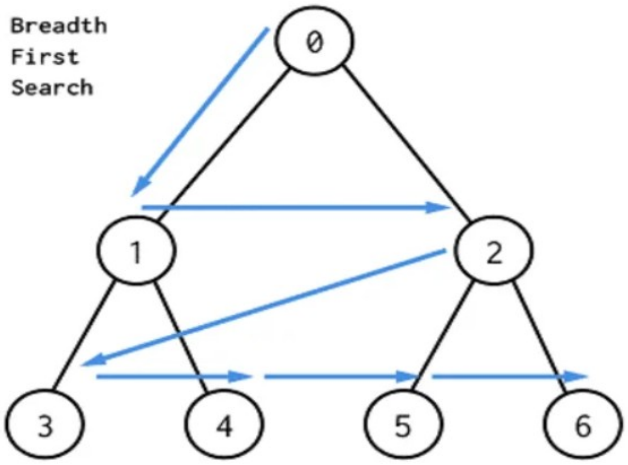
Код дотор ашиглагдсан гол сангууд:

- flask - HTTP сервер үүсгэж, хэрэглэгчийн хүсэлтийг хүлээн авч хариу илгээх
- geopandas (gpd) - Замын shapefile уншиж, геометр өгөгдөлтэй ажиллах
- math - Радиан хувиргалт болон геометрийн тооцоолол
- time - Алгоритмын гүйцэтгэлийн хугацааг хэмжих
- psutil - Санах ойн ашиглалтын хэмжээг тодорхойлох
- heapq - Priority Queue буюу Dijkstra-д ашиглагддаг өгөгдлийн бүтэц
- collections - deque (BFS-д зориулсан queue), defaultdict (графикийн ирмэг хадгалах)
- cKDTree - координатаар хамгийн ойрын зангилаа (nearest node) хурдан олох.

3 Алгоритмуудын тайлбар

3.1 Breadth-First Search (BFS)

BFS алгоритм нь эхлэлийг зааж өгсөн зангилаанаас эхлээд графикийг төвшин бүрээр нь ахиж явдаг алгоритм. Хамгийн түрүүнд эхлэлийн зангилаа дээр очоод хамгийн ойр орших буюу зэргэлдээ байгаа зангилаанууд уруу явж түүний дараа зэргэлдээ зангилааны дараагийн зэргэлдээ зангилаа уруу явж энэ үйл явцаар очиж болох буюу зааж өгсөн зангилаа хүртэл үргэлжлэн хүрнэ.



0-ээс эхлэн өөрт хамгийн ойр орших 1, 2 уруу зочлох хэрэгтэй болно. Гэхдээ аль нь уруу нь эхлээд зочлох 1, 2 хоёрын дарааллаар шийдэх бөгөөд B нь эхэнд байдаг учир эхлээд B уруу нь зочилно. Энэ мэтчилэн үргэлжлэн явсны эцэст **1 2 3 4 5 6** гэж зочилно.

График 1: BFS ажиллагааны график

Үүнийг доорх кодоор илэрхийлсэн болно.

Код 1: BFS алгоритмыг ашигласан код

```
from collections import deque

def bfs(graph, start, end):
    queue = deque([[start]])
    visited = set([start])

    while queue:
        path = queue.popleft()
        node = path[-1]
        if node == end:
            return path
        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(list(path) + [neighbor])
    return None
```

Queue-д зөвхөн оройг биш, харин тухайн оройд хүрэх замыг (List) хадгалж байгаа нь BFS-ээр дамжуулан замыг бүрэн эхээр нь хадгалах зорилготой. Энэ нь BFS-ийн гол зарчим болох хамгийн түрүүнд орсон цэгийг түрүүлж судлах (өөрөөр хэлбэл, ойр байгаа төвшнийг) үйлдлийг гүйцэтгэнэ.

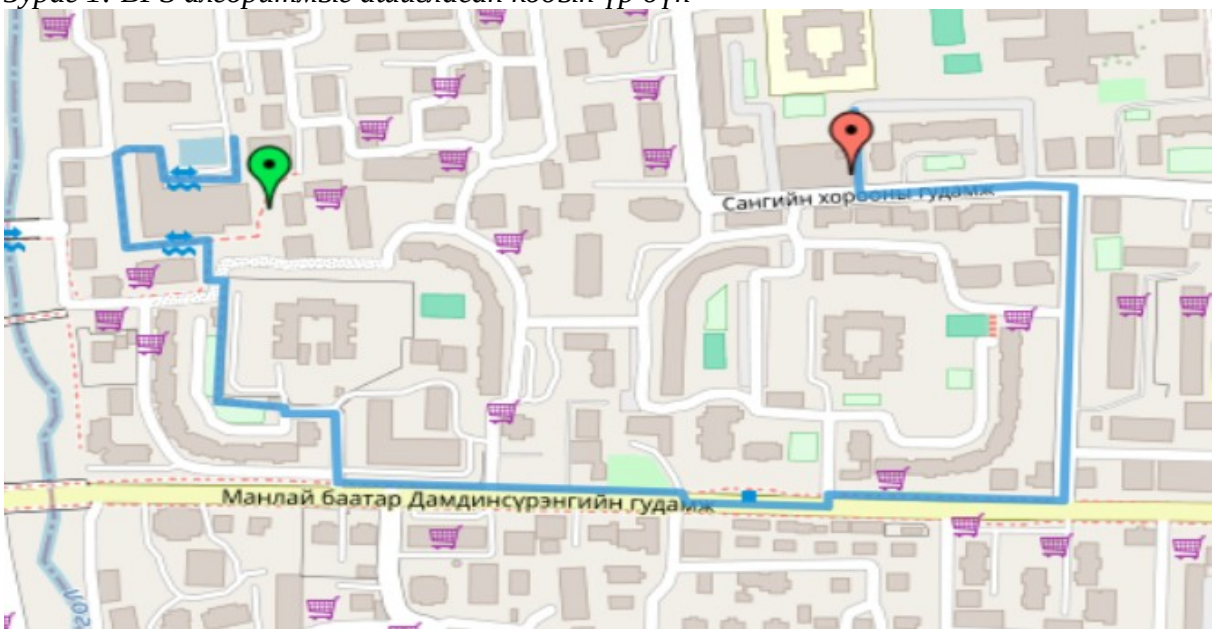
Ажлын зарчим: Эхнээс ойролцоо давталт (queue) ашиглан давталт хийнэ. visited-ээр дахин орохоос хамгаална.

Код дээрх онцлогууд:

- find_nearest_node-оор эх болон төгсгөлийг олно.
- deque -ыг queue болгон ашиглана.
- Хэрвээ зам олбол path-аас оройн координатуудыг сэргээж буцаана.

Үр дүнд нь:

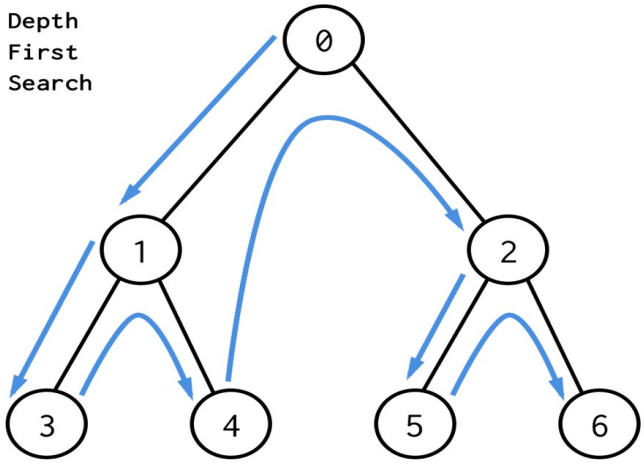
Зураг 1: BFS алгоритмыг ашигласан кодын үр дүн



3.2. Depth-First Search (DFS)

DFS алгоритм нь BFS-ийн нэгэн адил график дахь цэгүүдийг судлахад ашиглагддаг боловч, судалгааны чиглэл нь эсрэг төрөл нь юм. DFS нь эхлэлийн цэгээс нэг чиглэлийг сонгож аваад, тэр чиглэлдээ хамгийн гүн буюу хамгийн боломжтой эцэс хүртэл үргэлжлэн явдаг. Зөвхөн мухардалд (цааш явах боломжгүй цэгт) хүрсний дараа буцаж (Backtracking), өөр чиглэлээр хайлтыг үргэлжлүүлдэг.

Хэрэгжүүлэх арга нь stack -ийг ашигладаг. Ихэнхдээ рекурсив функцээр хэрэгжүүлдэг бөгөөд функцийн дуудлага нь өөрөө stack үүргээ гүйцэтгэдэг.



Эхлэлийн цэгээс эхлэн очиж болох хамгийн эцсийн цэг хүртэл явна. Үүний дараагаар буцах буюу backtrack хийж салаалсан хэсэг буюу өөр боломжтой зам уруу шилжиж түүнийхээ эцсийн цэг хүртэл явна.

Үүний эцсийн үр дүнд нь: **0 1 3 4 2 5 6**

График 2: DFS ажиллагааны график

Үүнийг доорх кодоор илэрхийлсэн болно.

Код 2: DFS алгоритмыг ашигласан код

```
def dfs(graph, start, end, path=None, visited=None):
    if visited is None:
        visited = set()
    if path is None:
        path = [start]
    visited.add(start)

    if start == end:
        return path

    for neighbor in graph[start]:
        if neighbor not in visited:
            result = dfs(graph, neighbor, end, path + [neighbor], visited)
            if result:
                return result
    return None
```

Энэ нь DFS-ийн гол зарчим болох хамгийн сүүлд орсон цэгийг түрүүлж судлах (өөрөөр хэлбэл, одоогийн цэгээс хамгийн гүн уруу үргэлжлүүлэх) үйлдлийг гүйцэтгэнэ. Сүүлчийн шалгалт нь аль хэдийн үзсэн цэг уруу буцаж орохгүй байх нөхцөлийг хангадаг.

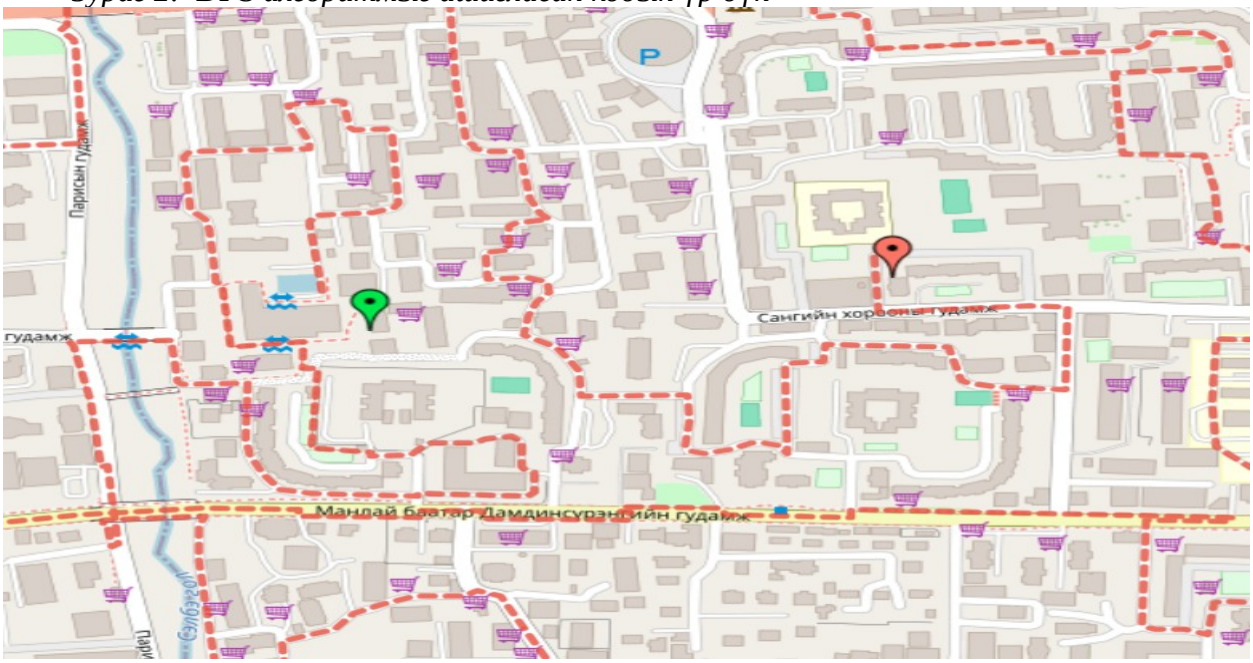
Ажлын зарчим: Рекурсив DFS (эсвэл стек) ашиглан эхнээс гүн уруу орох.

Код дээрх онцлогууд:

- visited нь бүх оройн дээр boolean.
- Рекурсив dfs_visit(node) функц замыг барьж, олбол буцааж өгнө.

Үр дүнд нь:

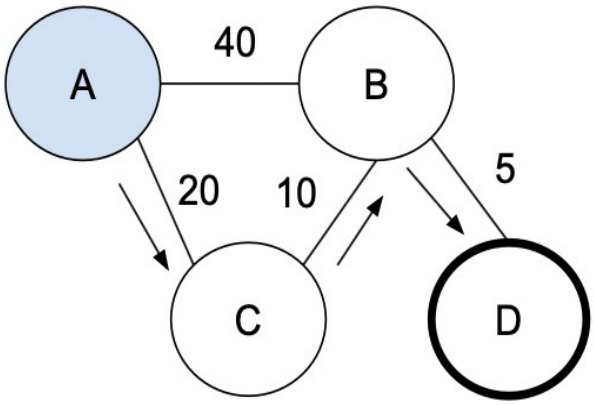
Зураг 2: DFS алгоритмыг ашигласан кодын үр дүн



3.3Dijkstra

Dijkstra-ийн алгоритм нь эхлэлийн цэгээс графикийн бусад бүх цэгүүд уруу хүрэх хамгийн бага нийт жинтэй (Weighted) замыг олох зорилготой. Энэ алгоритм нь BFS-ийн сайжруулсан хувилбар гэж ойлгогдож болох бөгөөд, Edge-ийн жин буюу өртгийг тооцоолон хайлт хийдгээрээ онцлог юм. Зөвхөн эерэг (Positive) жинтэй графикт ашиглагдана.

График 3: Dijkstra ажиллагааны график



Эхлэлийн цэг, зангилаанаас эхлээд хөрш цэгүүд уруу зочлон явах ба дараагийн цэгээ сонгохдоо хамгийн бага жинтэй буюу weight хамааран хамгийн бага жинтэйгээс эхлэн дараалж явдаг.

1. Эхлэлийн цэгийн зайг 0 гэж, бусдыг ∞ гэж авна.
2. Одоогийн хамгийн бага зайтай зангилааг сонгоно.
3. Түүний хөршүүдийн зайг шинэчилж, хамгийн бага утгыг хадгална.
4. Эцсийн цэг олдох хүртэл энэ үйл явцыг давтана.

Үүнийг доорх кодоор илэрхийлсэн болно.

Код 3: Dijkstra алгоритмыг ашигласан код

```
import heapq

def dijkstra(graph, start, end):
    queue = [(0, start)]
    dist = {start: 0}
    prev = {start: None}

    while queue:
        cost, node = heapq.heappop(queue)
        if node == end:
            break

        for neighbor, weight in graph[node]:
            new_cost = cost + weight
            if neighbor not in dist or new_cost < dist[neighbor]:
                dist[neighbor] = new_cost
                prev[neighbor] = node
                heapq.heappush(queue, (new_cost, neighbor))

    path = []
    while end:
        path.insert(0, end)
        end = prev.get(end)
    return path
```

Эхлэлийн цэгээс бусад бүх цэг нүүх богино зайн утгыг хадгалж авна. Энэ нь dijkstra-ийн гол зарчим болох хамгийн богино зайтай (хамгийн бага жинтэй) цэг уруу түрүүлж явах үйлдлийг хангаж байна. Энэ нь шинэ, илүү богино замыг олсон үед зайн утгыг амжилттай шинэчилж буйг илтгэнэ.

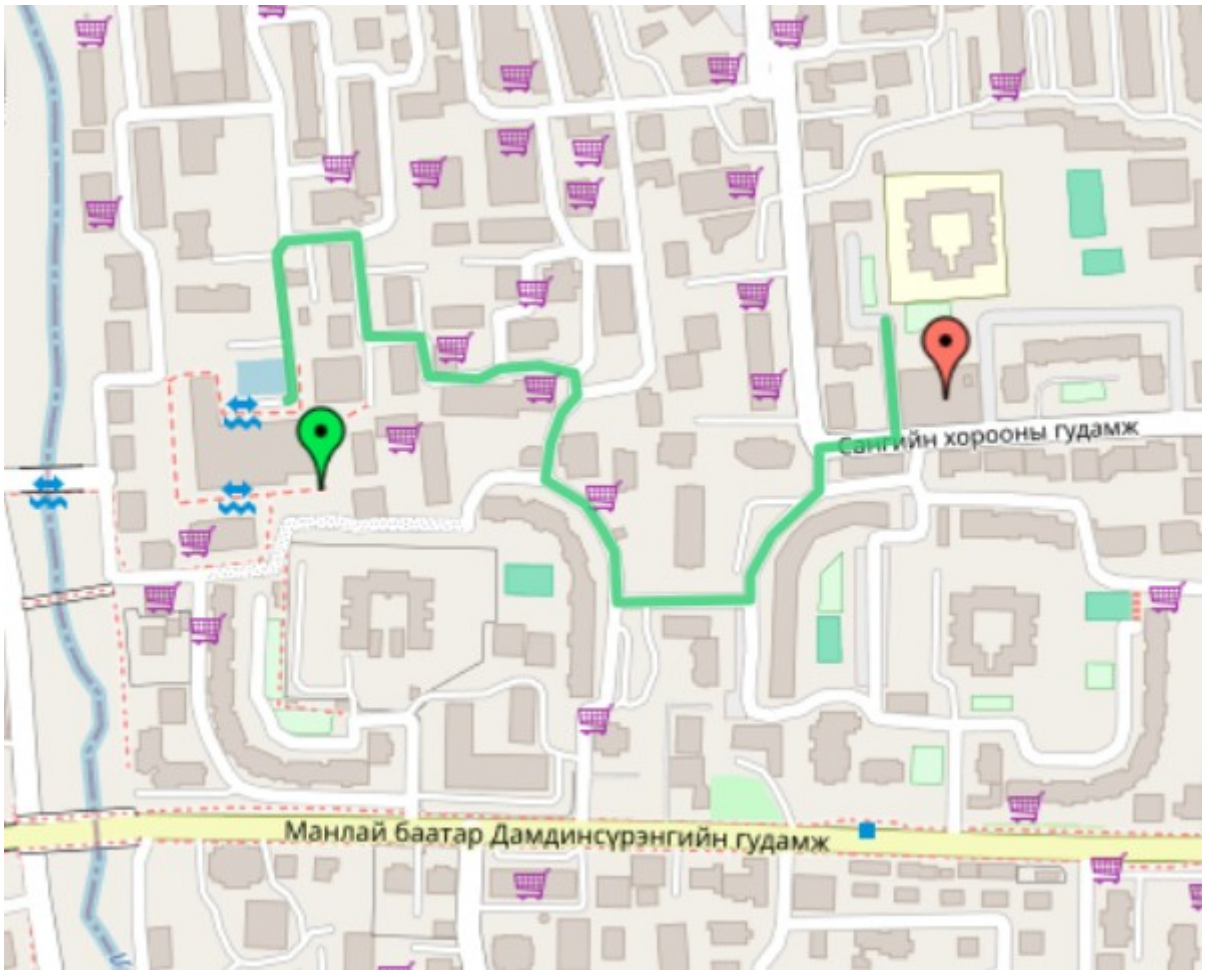
Ажлын зарчим: ашиглан эхнээс хамгийн ойрын (current shortest) оройг тэлнэ, дамжуулсан зайг шинэчилнэ.

Код дээрх онцлогууд:

- dist — бүх оройн хамгийн бага мэдэгдэх зайг хадгална.
- prev — урьдчилсан оройг хадгалах (замыг сэргээхэд ашиглана).
- heapq — priority queue.

Үр дүнд нь:

Зураг 3: Dijkstra алгоритмыг ашигласан кодын үр дүн



4 Дүгнэлт

Энэхүү систем нь Flask хүрээ ашиглан график онолын үндсэн алгоритмууд болох BFS (Breadth-First Search), DFS (Depth-First Search), болон Dijkstra алгоритмуудыг бодит газрын зургийн өгөгдөл (shapefile) дээр хэрэгжүүлсэн бүрэн ажиллагаатай веб API юм.

Энэ бие даалтаар BFS нь хамгийн хурдан боловч богино алхамын зам л гаргадаг, DFS нь бүх боломжит замыг судалж чадах ч урт зам гаргах хандлагатай, Dijkstra нь хамгийн оновчтой замыг гаргадаг бөгөөд хамгийн GPS системд тохиромжтой нь Dijkstra юм.