컴퓨터 네트워크 과제 3 2021039057 이규하

Task 1: 첨부 파일을 참고하여 Dijkstra 알고리즘을 완성하시오. 첨부 파일에는 Dijkstra 알고리즘의 실행할 수 있는 GUI 프로그램이 포함되어 있습니다. Dijkstra.doc 파일의 지시에 따라서 controllers.py의 빈 영역을 구현하면 됩니다. 본 과제는 특별히 코드 copy에 대하여 면밀하게 봅니다. 베끼는 경우 이유를 불문하고 누가 누구 것을 베낀 것인지에 무관하게 0점입니다.
보고서: 2 페이지 정도의 Dijkstra를 어떻게 구현했는지 구현 후 실행 화면은 어떻게 되는지 보고서에 작성하기 바랍니다.

Task 1에서는 과제 첨부파일 중 Dijkstra_doc.docx 파일의 지시에 따라 controllers.py의 다음 세 함수의 빈 부분을 채우면 됩니다. 이를 구현할 환경 및 계획은 다음과 같습니다.

- def search_min(graph: Graph, queue: *list*) -> Node:
- def update_distances(graph: Graph, node: Node):
- def dijkstra(graph: Graph, start_node: Node):

1. **Python 환경 설정 및 라이브러리 조사:** 다익스트라 알고리즘을 구현할 GUI를 구동하기 위해 pip으로부터 'pygame' 모듈을 설치하여 사용

2. **개발 환경**

- OS 환경: Ubuntu 24.04 on Windows 11 WSL2
- Python 버전: 3.9.20 with conda env

```
gyuha_lee@KONGSUNILAPTOP:~/dijkstra$ conda activate comne3
(comne3) gyuha_lee@KONGSUNILAPTOP:~/dijkstra$ python
Python 3.9.20 (main, Oct  3 2024, 07:27:41)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
(comne3) gyuha_lee@KONGSUNILAPTOP:~/dijkstra$ cat  /etc/*release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=24.04
DISTRIB_CODENAME=noble
DISTRIB_DESCRIPTION="Ubuntu 24.04 LTS"
PRETTY_NAME="Ubuntu 24.04 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04 LTS (Noble Numbat)"
```

3. **controllers.py 원본 코드 분석:** 다익스트라 알고리즘을 구현하기 위한 함수 구조와 변수 선언만 포함하고 있었으며, 핵심적인 로직은 TODO 주석으로 표시되어 있었습니다. 파일에서 정의된 주요 함수와 흐름은 다음과 같습니다.

- init 함수: 다익스트라 알고리즘의 초기 상태를 설정하는 데 사용. 그래프의 모든 노드에 대해 초기 거리를 무한대(math.inf)로 설정하고, 시작 노드의 거리를 0으로 초기화 함.
- search_min 함수: 현재 남아 있는 노드(큐) 중에서 최단 거리 값을 가진 노드를 선택하는 역할
- update_distances 함수: 선택된 노드의 인접 노드에 대해 새로운 경로를 통한 거리를 계산하고, 더 짧다면 해당 거리와 선행 노드를 갱신
- dijkstra 함수: 알고리즘의 전체 실행 흐름을 담당하며 최소 거리 노드 선택, 거리 갱신, 큐 업데이트를 반복적으로 수행
- find_path 함수: 다익스트라 알고리즘 실행 후, 출발 노드에서 목표 노드까지의 최단 경로를 추적하여 반환

➔ **이 중에서 search_min, update_distances, dijkstra 함수 코드 중 로딕이 공란으로 비어져있어 다음 차례에 제시된 것과 같이 구현하였습니다.**

4. controllers.py 코드 수정

- search_min 함수: 최소 거리 노드 선택 구현

  노드들의 거리 값(graph.distances)을 순회하며 가장 작은 값을 가진 노드를 선택, 선택된 노드는 이후 거리 갱신 작업에서 사용될 예정

  1. 큐에 남아 있는 모든 노드를 순회
  2. 각 노드의 거리 값을 비교하여, 가장 작은 값을 가진 노드를 추적
  3. 탐색이 끝나면, 최소 거리를 가진 노드를 반환

- update_distances 함수: 인접 노드 거리 갱신 구현

  선택된 노드와 연결된 간선의 가중치를 기준으로, 인접 노드로 이동하는 새 거리를 계산, 새 거리가 기존 거리보다 짧다면, 해당 노드의 거리(graph.distances)와 선행 노드 정보(graph.preds)를 갱신

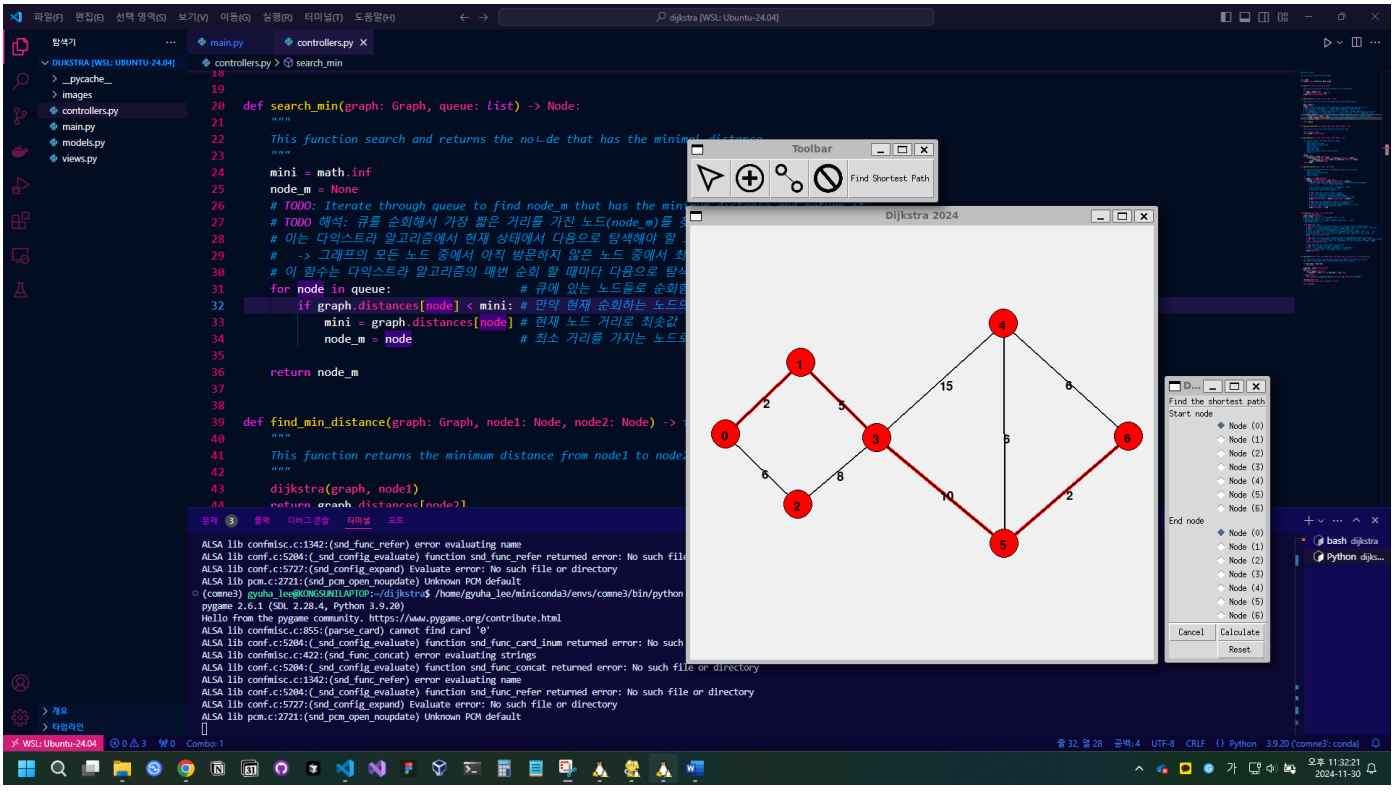
  1. 선택된 노드의 모든 인접 노드를 순회
  2. 경로의 가중치를 고려하여 새로운 거리를 계산
  3. 새로운 거리가 기존 거리보다 짧으면, 최단 거리와 선행 노드 정보를 갱신

- dijkstra 함수: 알고리즘의 흐름의 중추

  초기화를 통해 시작 노드의 거리를 0으로 설정하고, 다른 모든 노드의 거리는 무한대로 초기화, 큐에서 최소 거리 노드를 선택(search_min), 인접 노드의 거리를 갱신(update_distances)한 후, 선택된 노드를 큐에서 제거하는 과정을 반복, 큐가 비워질 때까지 이 과정을 반복하며, 최단 경로를 계산합니다.

  1. 초기화를 통해 시작 노드와 기타 노드의 거리를 설정
  2. search_min을 통해 최소 거리 노드를 선택
  3. 선택된 노드로부터 인접 노드의 거리를 update_distances로 갱신
  4. 선택된 노드를 큐에서 제거하여 반복 과정을 효율적으로 진행

5. 코드 전체 실행 화면: 예시와 동일한 결과를 얻음



Task 2: Distance vector 알고리즘에서 count-to-infinity 문제를 해결하기 위한 poisoned reverse는 어떤 방법인가?

Distance Vector 알고리즘은 라우터들이 서로 경로 정보를 주고받으며 최적의 경로를 계산하는 분산형 라우팅 방식입니다. 하지만 이 과정에서 네트워크의 특정 링크가 끊어지면, 잘못된 경로 비용이 순환하며 점차 증가해 무한대로 향하는 Count-to-Infinity 문제가 발생할 수 있습니다. 이런 상황은 네트워크 성능을 심각하게 저하시키며, 불필요한 계산과 데이터 전송이 반복되게 만듭니다. 예를 들어, 라우터 A가 B를 통해 목적지 D로 가는 경로를 사용하고 있었는데, D로의 직접적인 연결이 끊어진다면, A와 B는 서로 잘못된 경로 정보를 교환하며 점차적으로 비용을 증가시키게 됩니다.

이 문제를 해결하기 위한 방법 중 하나가 Poisoned Reverse입니다. 이 기법은 라우터가 특정 경로에 대해 이웃에게 광고할 때, 해당 경로 비용을 의도적으로 무한대(∞)로 설정하는 방식입니다. 예를 들어, 라우터 A가 B를 통해 C로 가는 경로를 가지고 있다면, A는 B에게 광고할 때 C로 가는 경로의 비용을 무한대로 설정합니다. 이렇게 하면 B는 A를 통해 C로 가는 경로를 선택하지 않게 되고, 잘못된 경로 정보가 되돌아오는 것을 방지할 수 있습니다.

Poisoned Reverse의 작동 과정은 다음과 같습니다. 평소에는 라우터 A가 B를 통해 C로 가는 경로의 비용을 정상적으로 광고합니다. 그런데 만약 A와 C 간의 경로가 끊어지면, A는 B에게 C로 가는 경로 비용을 무한대 값으로 설정해 광고합니다. 이렇게 하면 B는 C로의 경로를 다시 업데이트하지 않고, 잘못된 정보가 네트워크에서 순환하는 것을 막을 수 있습니다. 결과적으로 Count-to-Infinity 문제를 효과적으로 완화할 수 있습니다.

Task 2 참고문서

1. Code-Lab1. Count-to-Infinity 문제와 Poisoned Reverse 기법. Code-Lab1 기술 블로그, https://code-lab1.tistory.com/35.
2. GeeksforGeeks. Route Poisoning and Count to infinity problem in Routing. GeeksforGeeks, https://www.geeksforgeeks.org/route-poisoning-and-count-to-infinity-problem-in-routing/
3. Hedrick, C. Routing Information Protocol (RIP). RFC 1058, June 1988. https://datatracker.ietf.org/doc/html/rfc1058.