

ICE3020 알고리즘설계

설계프로젝트

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2021년 6월 20일

학부 정보통신공학과

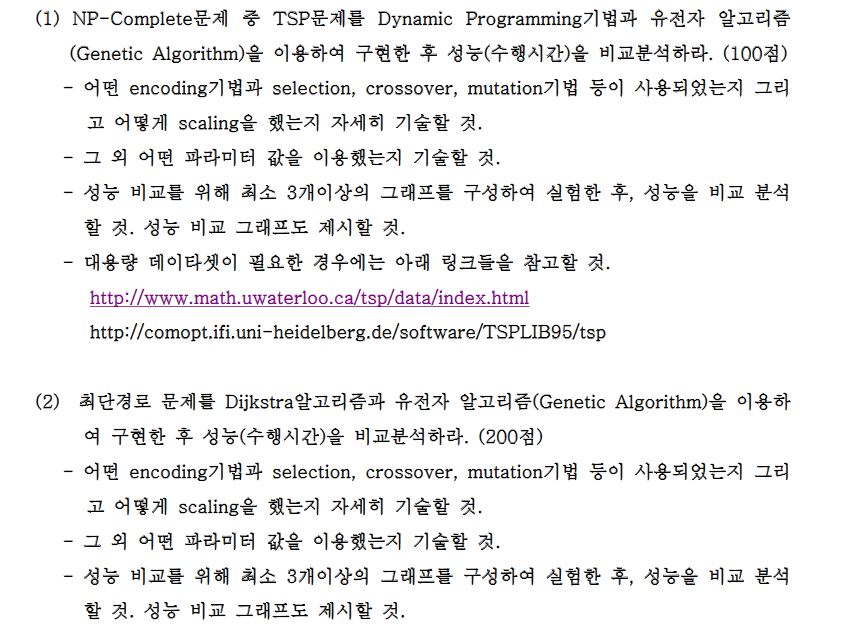
학년 4

성명 홍인호

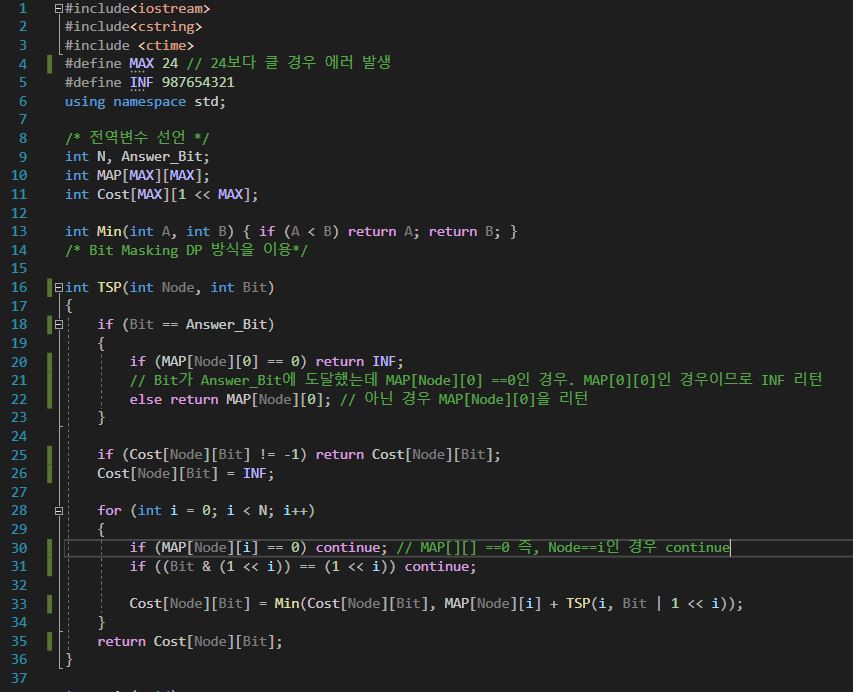
학번 12151641

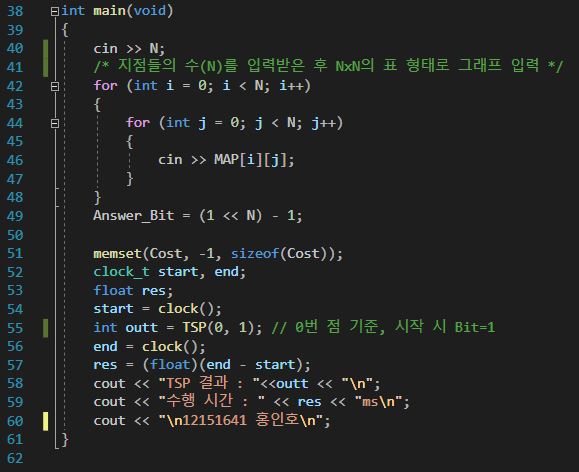


* **개요**



1. **TSP**
2. **Dynamic Programming 기법 소스코드 및 구현상 특징**





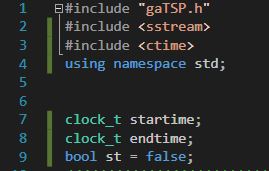
강의노트에 공개된 것과 비슷한 알고리즘을 Bit Masking을 이용하여 구현했다.

Bit Masking DP 방식이기 때문에 지점(city)들의 수는 24까지만 설정할 수 있었다.

1. **유전자 알고리즘 기법 구현상 특징**

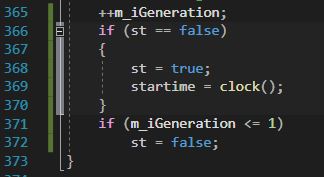
기본적으로 강의노트에서 제공된 GA소스 중 “TSP” 폴더에 있는 소스코드를 사용하였다.

* 해당 코드에서 변경된 부분
* gaTSP.cpp

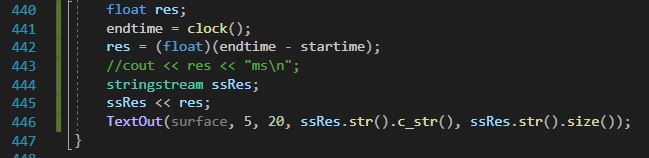


gaTSP.cpp의 코드 중 수행시간 출력을 위해 위와 같은 부분을 추가했다.





Epoch()함수에 대해 위와 같이 설정하였다. Generation이 재시작한 후 바로 다음 Epoch()가 시작 시간이 된다.



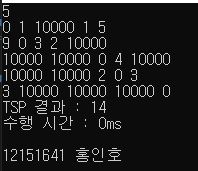
마지막 Render()함수 부분에 endtime을 추가하여 실행 시간을 출력한다.

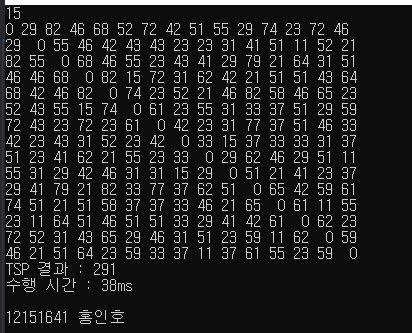
최적 답을 못 찾은 경우 Generation과 같이 계속해서 시간이 늘어나고, 찾은 경우 멈추게 된다.

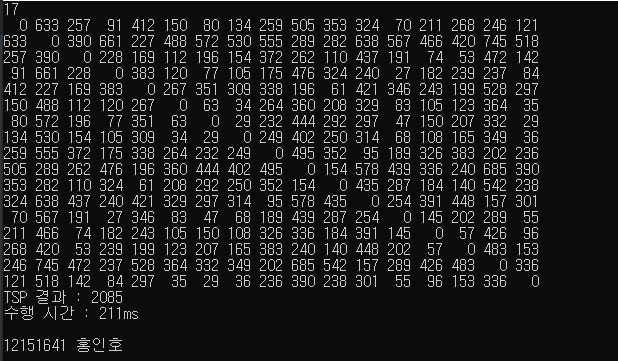
1. **실행 결과**

* Dynamic Programming의 경우

City의 숫자가 5, 15, 17인 세 가지 경우에 대해 실행해보았다.



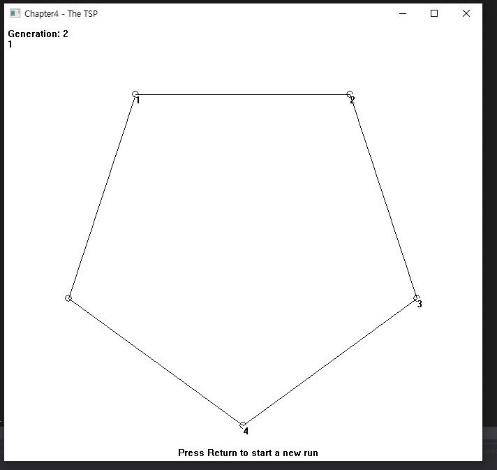
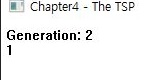




* 유전자 알고리즘의 경우

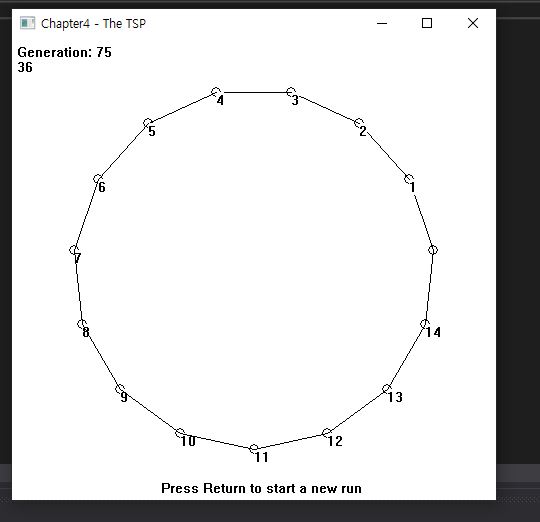
마찬가지로 5, 15, 17에 대해서 실행해봤고, 그보다 더 많은 수에 대해서도 진행해보았다.

최적 답을 찾은 경우에 대한 실행시간이 출력되도록 하였다.

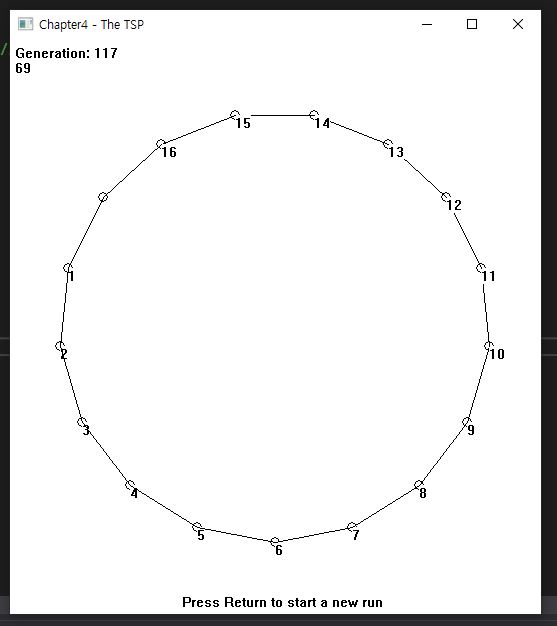
5의 경우 : 1ms (이하)

최적 답을 바로 찾지 못하는 경우가 많이 발생한다. 너무 적은 City 숫자에 대해서는 오히려 좋지 않은 것으로 보인다.

15의 경우 : 약 20 ~ 40 ms (Generation에 따라 변동)

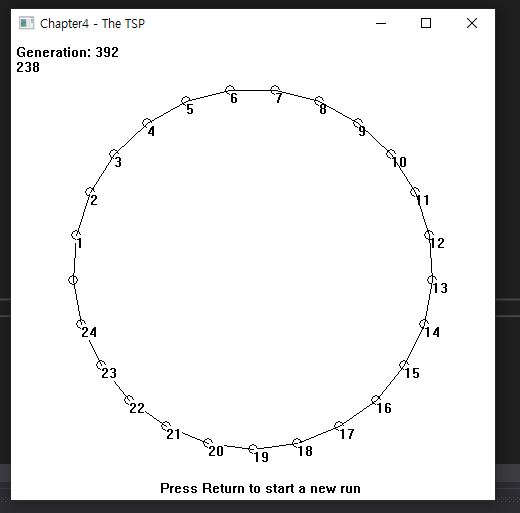
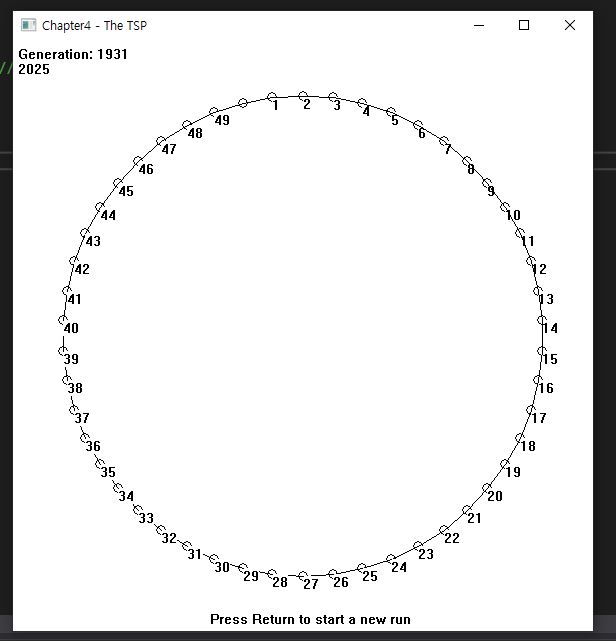
최적 답을 찾지 못한 경우의 빈도 수는 5때보다 훨씬 줄었다. Dynamic Programming과 비교했을 때 큰 차이를 보이지는 않는다. City 수가 적기 때문인 것으로 보인다.

17의 경우 : 약 20~100ms (40~80ms에 대부분 분포)

최적 답을 찾지 못한 경우의 빈도 수가 15보다 늘었다. City 수가 많아졌기 때문인 것으로 보인다.

최적 답을 찾지 못한 경우의 시간을 제외하면 Dynamic Programming 방법보다 훨씬 빠른 것을 확인할 수 있다.

25의 경우 : 약 200~300ms 50의 경우 : 약 2000~3000ms

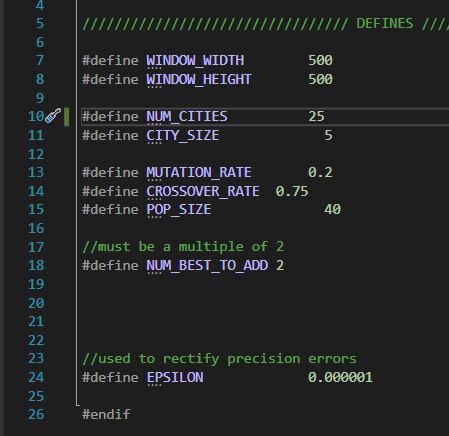
25부터는 Bit Masking 방식으로 구현한 Dynamic Programming 방법에서는 실행할 수 없었다.



Dynamic Programming 방법은 City수가 15에서 17로 늘어난 것 만으로 실행시간이 38ms에서 211ms로 매우 크게 늘었다. 그에 비해 Genetic Algorithm 방법의 경우 City수 17에서는 약 70ms로 DP보다는 훨씬 빨랐고, City수 25에서 238ms로, City수가 늘었는데도 DP 의 City수 17과 비슷한 실행시간인 것을 확인할 수 있다.

1. **유전자 알고리즘 세부 사항**

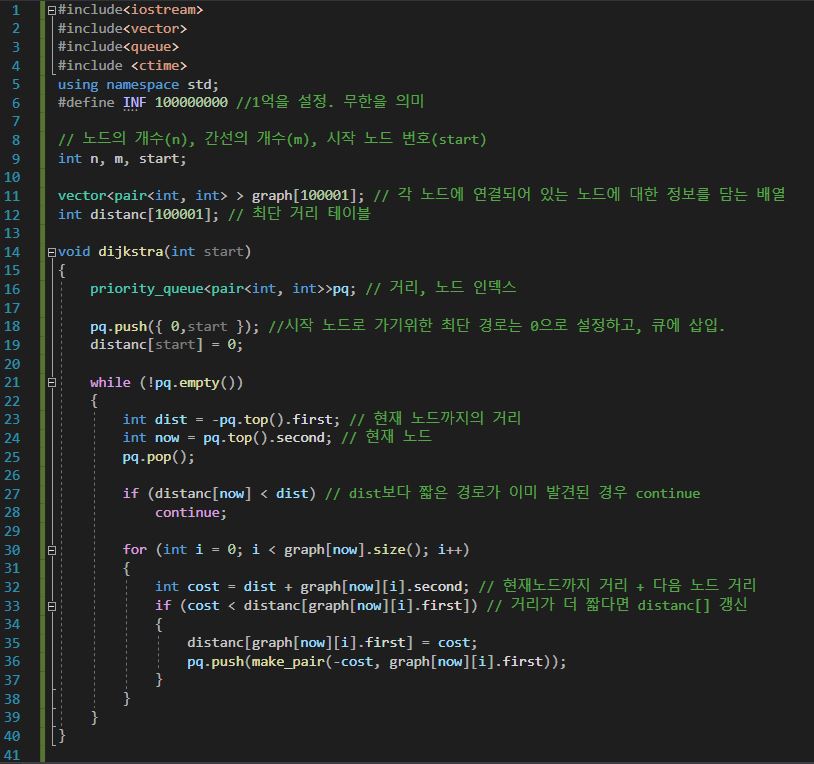
* encoding 기법 : defines.h에서 주어진 City들의 숫자와 size에 맞춰서 원 형태로 출력하도록 세팅
* selection : 룰렛 선택 사용, 엘리트주의 적용
* crossover : Permutation Crossover, PMX
* mutation : Exchange Mutation
* scaling : 주어진 소스코드 defines.h 부분에서 City의 숫자와 size를 설정



1. **성능 비교 그래프**

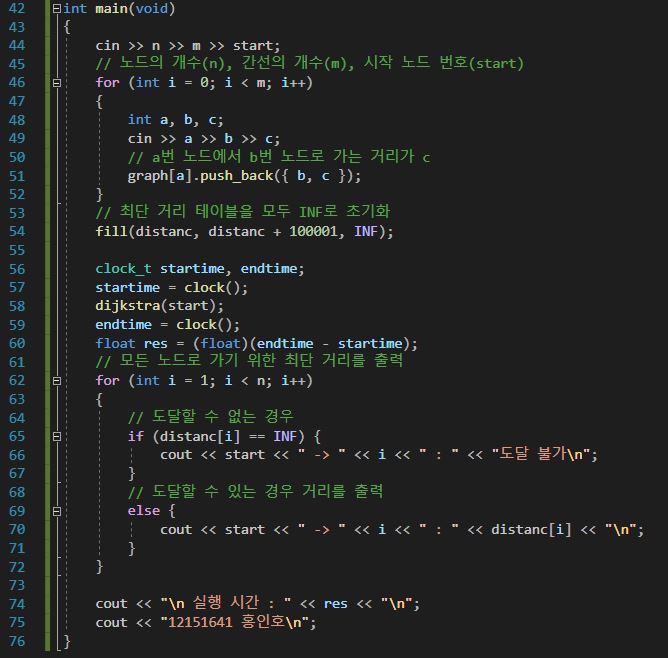
x축 : City의 수 , y축 : 실행 시간 (단위 : ms)

1. **최단 경로 문제**
2. **Dijkstra 알고리즘 소스코드 및 구현상 특징**



Priority Queue를 이용하여 Dijkstra 알고리즘을 조금 더 간단하게 구현하였다.

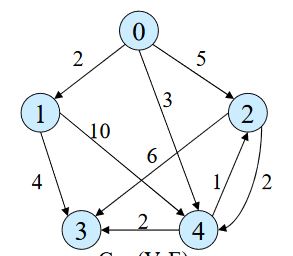
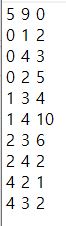
대부분의 설명은 코드 주석으로 표현하였다.

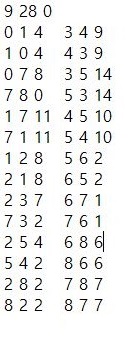
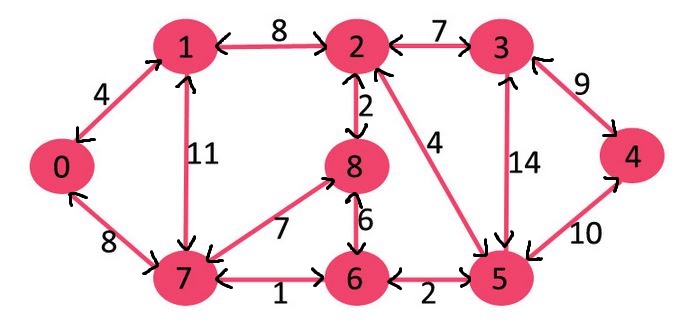


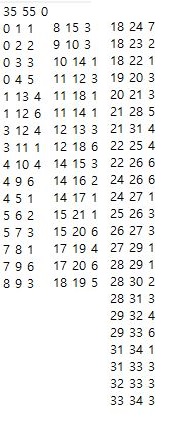
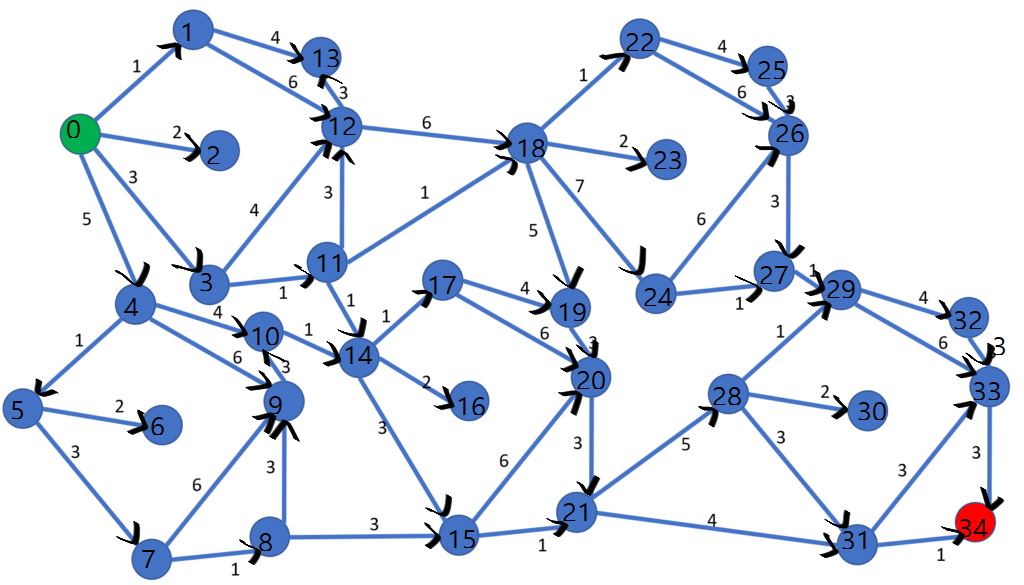
첫째 줄에 노드의 수, 간선의 수, 시작 노드 번호를 입력하고 시작한다.

1. **실행 결과**

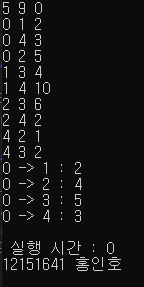
총 세 개의 그래프를 이용하였다. 그 그래프들은 다음과 같다.

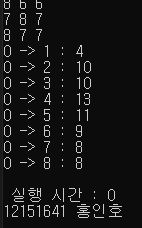




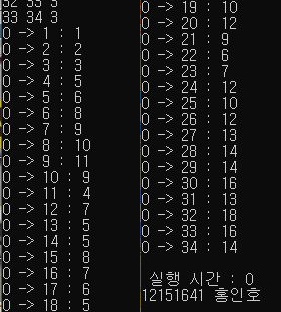
* 첫 번째 그래프



* 두 번째 그래프



* 세 번째 그래프



정상적으로 출력된 것을 확인할 수 있다.

시간 복잡도가 O(ElogV) 이기 때문에 세 그래프 모두 굉장히 빠르게 처리된 것을 알 수 있다.

모두 1ms 미만의 속도로 처리되었다.

1. **특별 사항**

* Genetic Algorithm을 이용한 최단 경로 문제는 구현하지 못했습니다.
* TSP 문제 중 Genetic Algorithm을 사용한 경우는 I-Class에 있는 GA소스 중 “TSP”폴더에 있는 코드를 이용하여 해당 보고서에 쓴 내용 부분만 수정했습니다.
* GA의 경우 보고서에 적힌 부분만 수정된 것이므로, 제출 파일에 포함하지 않았습니다.