[Project #1]

Traveling Salesman Problem

[Team Name (#팀 07)]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **이름** | **학번** | **학년** | **E-mail** |
| 팀장 | 박진혁 | 20180293 | 3 | [Jh040597@naver.com](mailto:Jh040597@naver.com) |
| 조원1 | 임성현 | 20180325 | 3 | jfknfj920@soongsil.ac.kr |
| 조원2 | 박병수 | 20170283 | 3 | byeongsu@soongsil.ac.kr |

***프로그램 사용법***

***all\_random***

* 완전 랜덤하게 도시 배열
* ./all\_random.exe
* exe파일과 같은 위치에 TSP.csv 필요함

***all\_GA***

* 유전 알고리즘만을 사용해서 도시 배열
* ./all\_GA.exe
* exe파일과 같은 위치에 TSP.csv 파일 필요함

***cluster\_random***

* 완전 랜덤하게 클러스터 나누기
* ./cluster\_random.exe
* exe파일과 같은 위치에 TSP.csv 파일 필요함

***cluster\_k\_mean***

* k-means 방법으로 클러스터 나누기
* ./cluster\_k\_mean.exe
* exe파일과 같은 위치에 TSP.csv 파일 필요함
* (중요)이 클러스터 파일로 최종 결과 파일을 만들수 없음!

***cluster\_mst***

* 클러스터별 최소 스패닝 트리 길이의 합이 가장 작도록 클러스터 나누기
* ./cluster\_mst.exe
* exe파일과 같은 위치에 TSP.csv 파일 필요함

***cluster\_hamilton***

* 클러스터별 최소 해밀턴 경로 길이 합이 가장 작도록 클러스터 나누기
* ./cluster\_hamilton.exe
* exe파일과 같은 위치에 TSP.csv 파일 필요함

***GA\_cluster***

* 클러스터들의 순서 배열
* 클러스터 파일 필요함
* ./GA\_cluster.exe cluster\_mst.tsv
* exe파일과 같은 위치에 TSP.csv 파일 필요함
* exe 파일과 같은 위치에 클러스터 파일 필요함(tsv), 명령줄 인수로 파일명 전달 필요

***소스코드 설명***

***all\_random***

* 완전 랜덤하게 경로를 생성함
* 결과 파일로 all\_random.csv 파일을 생성
* 이 파일은 city index의 sequence
* 실행시 exe파일과 같은 위치에 TSP.csv 파일 필요함

***all\_GA***

* 전체 city를 대상으로 유전 알고리즘을 사용해 경로를 찾음
* 결과 파일로 all\_GA.csv 파일 생성
* 이 파일은 city index의 sequence
* 실행시 exe 파일과 같은 위치에 TSP.csv 파일 필요함

***ai\_io.h – 파일 io관련 함수들***

* vector<pair<double, double>> read\_input\_file(void);
* TSP.csv 파일을 읽어서 좌표 벡터로 반환
* void save(const vector<int>& result);
* city index sequence를 받아서 파일로 저장

***ai\_utility.h – 랜덤함수, 거리 계산 함수 등..***

* int random(const int from, const int to);
* from에서 to – 1 사이의 랜덤한 정수 반환
* vector<int> create\_random\_indexes(const int sz);
* 0~sz - 1까지의 정수가 한번씩 들어간 랜덤한 벡터 반환
* double dist(const int, const int , const vector<pair<double, double>>&);
* 두 점 사이의 거리를 반환
* validate(const vector<int>&);
* city index sequence의 유효성 검사

***GA.h – 핵심이 되는 유전 알고리즘 부분***

* vector<int> GA::get\_indexes(void) const;
* 현재 city index sequence를 반환
* double GA::get\_distance\_sum(void) const;
* 현재 city index sequence를 따르는 경로의 길이 반환
* void next(const int ban\_count, const bool mutation, const int mutation\_count);
* 유전 알고리즘에서 다음 세대로 진행
* void ban(const int count);
* 양 옆 도시와의 거리가 가장 먼 순으로 count개의 도시를 ban pool로 보냄
* void random\_ban(const int count);
* 랜덤한 count개의 도시를 ban pool로 보냄
* void insert(void);
* Ban pool에 있는 도시들을 비어있는 sequence 중 양 옆 도시와의 거리가 가까운 곳에 삽입

ai\_io.h, ai\_utility.h는 공통이므로 아래에서는 설명하지 않음

***cluster\_random***

* 완전 랜덤하게 클러스터로 나눔
* 각 클러스터의 크기는 10
* 결과 파일로 cluster\_random.tsv 파일을 생성
* 이 파일은 클러스터 파일
* 한 행이 한 클러스터를 의미함(이하 클러스터 파일 모두 동일)
* 실행시 exe파일과 같은 위치에 TSP.csv 파일 필요함

***cluster\_k\_mean***

* 클러스터를 k-평균법으로 나눔
* 각 클러스터의 크기는 일정하지 않음
* 결과 파일로 cluster\_kmeans.tsv 생성
* 이 파일은 클러스터 파일임
* 실행시 exe파일과 같은 위치에 TSP.csv 파일 필요함

***Kmeans.h – kmeans를 수행하는 클래스***

* vector<vector<int>> get\_clusters(void) const;
* 현재 클러스터들을 반환
* double get\_distance\_to\_centroids\_sum(void) const;
* 각 노드가 속한 클러스터의 중심점과의 거리 합을 반환
* int get\_min\_cluster\_size(void) const;
* 현재 가장 작은 클러스터 크기 반환
* int get\_max\_cluster\_size(void) const;
* 현재 가장 큰 클러스터 크기 반환
* void next(void);
* 다음 세대로 진행
* double distance(const pair<double, double>& centroid, const int node) const;
* 두 점 사이의 거리 구하기
* void update\_nodes(void);
* 현재 centroids를 기준으로 노드 재배치
* void update\_centroids(void);
* 현재 클러스터를 기준으로 centroids 구하기

***cluster\_mst***

* 클러스터를 최소 스패닝 트리의 길이 합이 최소가 되도록 나눔
* 클러스터의 크기는 모두 10
* 결과 파일로 cluster\_mst.tsv 생성
* 이 파일은 클러스터 파일임
* 실행시 exe파일과 같은 위치에 TSP.csv 파일 필요함

***Cluster.h – 노드 10개를 가지고 있는 클러스터 클래스***

* vector<int> get\_nodes(void) const;
* 클러스터를 구성하는 노드 반환
* double get\_min\_dist(const int idx) const;
* 입력으로 받은 좌표와 현재 클러스터 노드들 중 가장 가까운 거리 반환
* double get\_mst\_sum(void) const;
* 현재 노드들의 최소 스패닝 트리 경로 길이 합을 반환
* vector<int> ban(const int count);
* 옆 노드와의 거리 평균 길이가 큰 노드 count개를 현재 클러스터에서 제거 후 반환
* vector<int> ramdom\_pop(const int count);
* 랜덤하게 count개의 노드를 현재 클러스터에서 제거 후 반환
* void add\_nodes(const vector<int>& add\_nodes);
* 입력으로 받은 노드들을 현재 클러스터에 추가
* void update\_mst\_edges(void);
* 현재 노드들로 최소 스패닝 트리 계산

***Board.h – 클러스터 100개를 가지고 있는 현재 상태(status) 클래스***

* next(const int ban\_count, const bool mutation, const int mutation\_count);
* 다음 세대로 진행
* vector<vector<int>> get\_clusters(void) const;
* 현재 클러스터들 반환
* double get\_mst\_sum(void) const;
* 현재 클러스터들의 최소 스패닝 트리 경로 길이 합 반환

***cluster\_hamilton***

* 클러스터를 가장 길이가 짧은 해밀턴 순환 길이의 합이 최소가 되도록 나눔
* 클러스터의 크기는 모두 10
* 결과 파일로 cluster\_hamilton.tsv 생성
* 이 파일은 클러스터 파일임
* 실행시 exe파일과 같은 위치에 TSP.csv 파일 필요함

***Cluster.h – 노드 10개를 가지고 있는 클러스터 클래스***

* vector<int> get\_nodes(void) const;
* 현재 노드들 반환
* double get\_min\_dist(const int idx) const;
* 입력받은 좌표와 현재 클러스터를 구성하는 노드들 중 가장 짧은 거리 반환
* double get\_hamilton\_sum(void) const;
* 가장 짧은 해밀턴 순환 길이 합을 반환
* vector<int> ban(const int count);
* 양 옆 노드와의 거리가 먼 count개의 노드를 클러스터에서 제거 후 반환
* vector<int> random\_pop(const int count);
* 랜덤하게 count개의 노드를 클러스터에서 제거 후 반환
* void add\_nodes(const vector<int>& add\_nodes);
* 입력받은 노드들을 현재 클러스터에 추가
* void update\_hamilton\_edges(void);
* 현재 노드들로 해밀턴 순환 정보 계산

***Hamilton.h – 해밀턴 순환들을 구하기 위한 클래스***

* double get\_cycle\_length(void) const;
* 가장 짧은 해밀턴 순환 경로 길이 반환
* deque<int> get\_path(void) const;
* 가장 짧은 해밀턴 순환 경로 반환
* void calculate(const int current\_node, const double current\_length, deque<int>& path, vector<bool>& visit);
* 입력받은 노드들로 가장 짧은 해밀턴 순환 계산

***Board.h – 클러스터 100개를 가지고 있는 현재 상태 클래스***

* void next(const int ban\_count, const bool mutation, const int mutation\_count);
* 다음 세대로 진행
* vector<vector<int>> get\_clusters(void) const;
* 현재 클러스터들 반환
* double get\_hamilton\_sum(void) const;
* 현재 클러스터들의 최소 해밀턴 순환 길이 합 반환

***GA\_cluster***

* 클러스터들의 순서를 유전 알고리즘을 이용하여 정함
* 실행시 exe파일과 같은 위치에 TSP.csv파일, 클러스터 파일 필요함
* 사용법: this.exe 클러스터파일이름
* 결과파일로 ga\_cluster.csv 생성
* 이 파일은 city index의 sequence

***Cluster.h – 시작점/끝점 별 가장 빠른 해밀턴 경로를 가지고 있는 클러스터 클래스***

* double get\_path\_length(cons int begin, const int end) const;
* 시작점과 끝점을 받아서 그 점들을 시작/끝점으로 하는 가장 짧은 해밀턴 경로 길이 반환
* deque<int> get\_path(const int begin, const int end) const;
* 시작점과 끝점을 받아서 그 점들을 시작/끝점으로 하는 가장 짧은 해밀턴 경로 반환
* vector<int> get\_nodes(void) const;
* 현재 클러스터 노드들 반환
* void hamilton\_path\_calculate(const int current\_node, const int path\_sum, vector<bool>& visit, deque<int>& path);
* 시작점/끝점별 가장 짧은 해밀턴 경로 계산
* int pos\_to\_index(const int pos) const;
* 좌표 인덱스를 클러스터 노드 인덱스로 변환
* int index\_to\_pos(const int index) const;
* 클러스터 노드 인덱스를 좌표 인덱스로 변환

***Board.h – 클러스터 클래스를 100개 가지고 있는 상태 클래스***

* next(const int ban\_count, const bool mutation, const int mutation\_count);
* 다음 세대로 진행
* vector<vector<int>> get\_clusters(void) const;
* 현재 클러스터들 반환
* vector<int> get\_path(void) const;
* city index sequence 반환
* double get\_path\_length(void) const;
* city index sequence 길이 반환
* void update\_path(void);
* 현재 클러스터의 순서로 전체 해밀턴 순환 길이와 그 노드 순서 구하기
* double get\_cluster\_distance(const Cluster& c1, const Cluster& c2);
* 클러스터들 사이의 가장 가까운 두 점 거리 구하기
* void ban(const int count);
* 주변 클러스터들과의 거리가 먼 순으로 count개의 클러스터를 ban pool로 옮김
* void random\_ban(const int count);
* 랜덤하게 count개의 클러스터를 ban pool로 옮김
* void insert(void);
* ban pool에 있는 클러스터들을 옆 클러스터들과의 거리가 가까워질 수 있는 순서로 배치함
* double get\_cluster\_distances(const Cluster& c1, const Cluster& c2);
* 클러스터간의 거리 구하기
* Pair<int, int> get\_cluster\_gate(const Cluster& c1, const Cluster& c2);
* 클러스터와 다른 클러스터간의 가장 가까운 점 구하기
* int get\_second\_gate(const Cluster& me, const cluster& you, const int banned) const;
* 클러스터와 다른 클러스터간의 두번째로 가까운 점 구하기