



SAMHANSHAT

ساحنشت



پروژه پایانی ساختمان های داده

نام استاد : الهام افشار

اسامی اعضای گروه :

امیرحسین براتی 40112358006

حسن سلیمانی 40112358019

ترم 4021

فهرست

3	فهرست
4	مقدمه
5	HeaderFiles
6	شروع برنامه (main.cpp)
7	Besttime.hpp
8	LowestCost.hpp
9	City.hpp
10	Cost.hpp , Path.hpp , TimedPath.hpp
11	Request.hpp
12	Station.hpp
13	Vehicle.hpp
14	Time.hpp
15	Hash.hpp
16	Routing.hpp
17	منابع

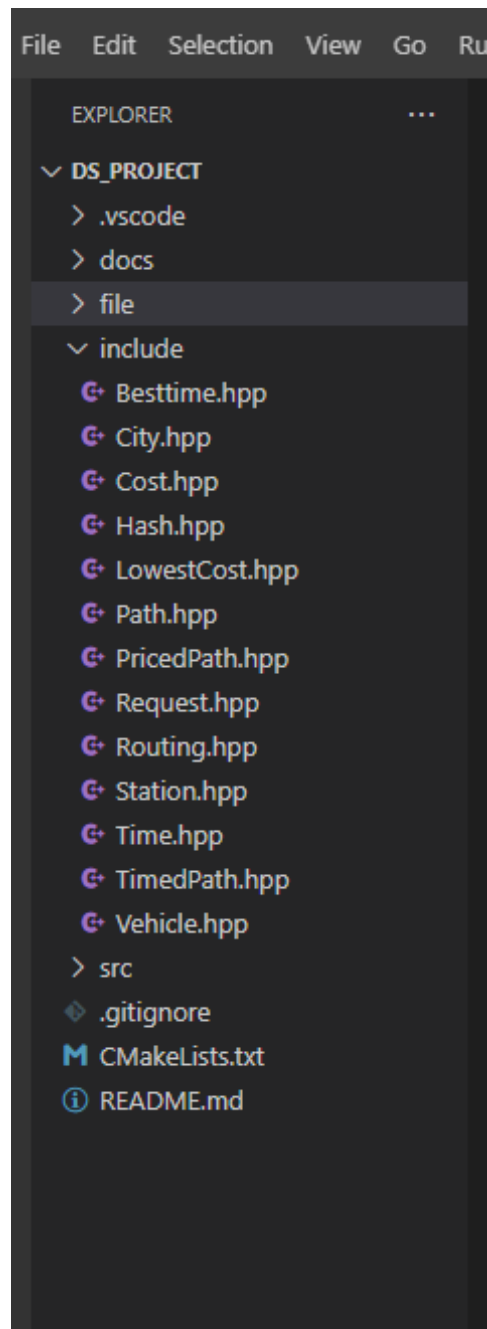
مقدمه

هدف از طراحی این پروژه مدیریت سامانه حمل و نقل شهر تهران هست که نقشه ای از خطوط اتوبوس ، مترو و تاکسی و فاصله بین ایستگاه ها در اختیار ما گذاشته شد.

بر اساس متن پروژه کاربر تعداد درخواست ها ، ساعت حرکت ، مبدأ و مقصد را وارد میکند و باید با استفاده از الگوریتم های کوتاه ترین مسیر و اعمال تغییراتی در آن ها سه نوع خروجی (کوتاه ترین مسافت ، کمترین هزینه و بهترین زمان) را با ذکر ایستگاه های گذرانده شده و نوع وسیله نقلیه استفاده شده نمایش داده شود .

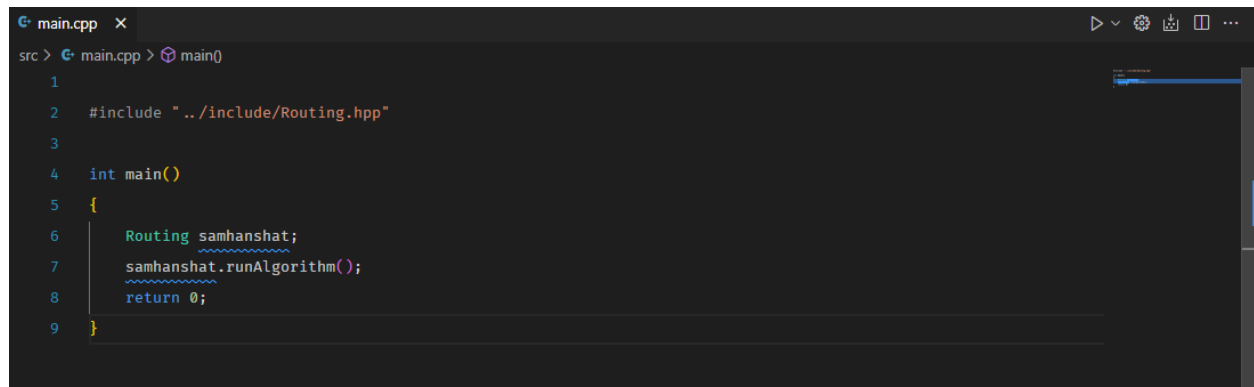
در توسعه این برنامه از محیط توسعه یکپارچه VSCode ، زبان برنامه نویسی ++C ، کامپایلر MinGw , ++g استفاده شده است. که در ادامه با بخش های مختلف آن آشنا میشویم .

HeaderFiles



شروع برنامه (main.cpp)

برنامه با ساختن یک شی از کلاس Routing و فراخوانی تابع runAlgorithm آن شروع میشود.



```
main.cpp x
src > main.cpp > main()
1
2 #include "../include/Routing.hpp"
3
4 int main()
5 {
6     Routing samhanshat;
7     samhanshat.runAlgorithm();
8     return 0;
9 }
```

Besttime.hpp

در این کلاس تمامی اعمال مربوط به بهترین زمان (Best-Time) انجام میشود که اصلی ترین بخش آن تابع Dijkstra است.

```
class Besttime
{
public:
    Besttime();
    ~Besttime();
    void Dijkstra(int source);
    void FillAdjMatrix(std::vector<Station>*, std::vector<Path>*);
    void PrintAdjMatrix(std::vector<Station>*);
    void Dijkstra(int, string);
    void printDijkstra(std::vector<Station>*);
    void Print(std::vector<Station>* , int, pair<pair<TimedPath, int>, pair<bool, int>>*>);
    void PrintPath(std::vector<Station> *, int, int);
    int* getDijkstraList ();
    void setArrivingTime(Time);
    Time getArrivingTime();
    pair<pair<TimedPath, int>, pair<bool, int>> * getParents();
    int getIndexFromParents(int, int);
private:
    std::vector<std::vector<TimedPath>> adjMatrix;
    int dijkstraList[N];
    pair<pair<TimedPath, int>, pair<bool, int>>* parents;
    Time arriving_time;
};

#endif
```

LowestCost.hpp

```
class LowestCost
{
public:
    LowestCost();
    void fillCostMatrix(std::vector<Station>*, std::vector<Path>*); /* filling adjacency matrix for calculating the best cost path */
    void printCostMatrix(std::vector<Station>*);
    void dijkstraOnCost(int, std::string); /* dijkstra implementation for the best cost path and invokes the print function */
    void printDijkstraOnCost(std::vector<Station>*);
    void dijkstra(int source);
    pair<pair<Cost, int>, pair<int, int>>* getParents();
    int * getCostDijkstraList();
    void setArrivingTime(Time);
    Time getArrivingTime();
    int getIndexFromParents(int, int);
    void print_cost(int, vector<Station>*, pair<pair<Cost, int>, pair<int, int>>*); /* prints some basics and printPath and calculateTime functions
    void print_path_cost(std::vector<Station> *, int, int);

private:
    Time arriving_time;
    pair<pair<Cost, int>, pair<int, int>>* parents;
    /*parents.first.second is time
    parents.second.first is traffic status
    parents.second.second is parent index
    */
    vector<pair<Cost, pair<int, int>>> path;
    std::vector<std::vector<Cost>> costMatrix; /* adjacency matrix for cost */
    int costDijkstraList[N];
```

در این کلاس متد های لازم برای یافتن کمترین هزینه (Lowest-Cost) موجود میباشد که پس از پر کردن ماتریس مجاورت تابع DijkstraOnCost برای این منظور طراحی شده است.

City.hpp

```
19 class City
20 {
21 public:
22     /*===== Distance =====*/
23
24     void FillAdjMatrix(std::vector<Station>*, std::vector<Path>*);           /* filling adjacency matrix for cal
25     std::vector<std::vector<Path>> GetAdjMatrix();                         /* returns the distance adjacency m
26     void PrintAdjMatrix(std::vector<Station>*);                           /* prints the distance adjacency ma
27
28     int MinDistance(int distance[], bool sptSet[]);
29     int GetShortestDistance(int);
30     void Dijkstra(int);
31     void Print(std::vector<Station>*, int, int);
32     void PrintAllPaths(std::vector<Station> *, int);
33     void PrintPath(std::vector<Station> *, int, int);
34     void setArrivingTime(Time);
35     Time getArrivingTime();
36     /*===== Cost =====*/
37     void Dijkstra_FloydWarshall(int, int, std::vector<Station>*, std::vector<Path>*, Time);
38     /*===== Time =====*/
39     void dijkstraOnTime(int, int, std::vector<Station> *, std::vector<Path>*, Time);
40 private:
41     std::vector<std::vector<Path>> adjMatrix; /* adjacency matrix for distance */
42     int dijkstraList[N]; /* dijkstraList[i] will hold the shortest distance/cost form source to i */
43     Time arriving_time; /* the time when user reach the destination */
44     pair<pair<Station, int>, pair<std::string, int>> includedStations[N];
45     /*first.first is station name first.second is the distance,
46     second.first is the name of line, second.second is the number of station's parent. */
```

متد های این کلاس همه ی اعمال برای نمایش کوتاه ترین فاصله (Shortest-Distance) ،
کمترین هزینه (Lowest-Cost) و بهترین زمان (Best-Time) را پیاده سازی میکند که در
کلاس Routing از این کلاس استفاده شده است تا با توجه به درخواست کاربر عملیات مربوطه
انجام شود.

Cost.hpp , Path.hpp , TimedPath.hpp

```
8  class Path
9  {
10 public:
11     void setFirstST(std::string firstst);
12     void setSecondST(std::string secondst);
13     std::string getFirstST();
14     std::string getSecondST();
15     void setSubway_Taxi_Line(std::string);
16     void setBus_Line(std::string);
17     std::string getSubway_Taxi_Line();
18     std::string getBus_Line();
19     void setTrainTaxiDistance(unsigned int);    /* sets the distance between two stations connected by train or taxi */
20     void setBusDistance(unsigned int);         /* sets the distance between two stations connected by bus */
21     unsigned int getTrainTaxiDistance() const; /* returns the distance between two stations connected by train or taxi */
22     unsigned int getBusDistance() const;       /* returns the distance between two stations connected by bus */
23     void setDijkstraList(std::string);         /* pushes a station into a list which is used for dijkstra algorithm */
24     void clearDijkstraList();                  /* clears the list if it updates */
25     void printDijkstraList();
26     void Split(std::string record, char delimiter , std::vector <Path> *P);
27     void readFromFile(std::vector <Path>*,std::string);
28 private:
29     std::string firstST="",secondST="";
30     std::string subway_taxi_line="";
31     std::string bus_line="";
32     unsigned int trainTaxiDistance=0;          /* distance between two stations connected by train */
33     unsigned int busDistance=0;               /* distance between two stations connected by bus */
34     std::vector<std::string> dijkstraList;     /* list for showing the stations after dijkstra algorithm */
35 }
```

ماتریس های مجاورت برای فراخوانی توابع Dijkstra از نوع این سه کلاس می باشند که شامل مواردی همچون نام ایستگاه ، فاصله ، هزینه ، اسامی لاین مربوطه و نوع وسیله ی نقلیه ای که از آن برای پیمون مسیر استفاده شده است می باشد.

Request.hpp

```
7  class Request
8  {
9  public:
10     void SetTime(std::string Time);
11     void SetOrigin(std::string Origin);
12     void SetDestination(std::string Destination);
13     Time GetTime();
14     std::string GetOrigin();
15     std::string GetDestination();
16     std::vector<std::string> Split(std::string Time, char delimiter);
17     // split function for separating am and pm
18     // delimiter is white space
19
20 private:
21     Time time;
22     std::string origin = "";
23     std::string destination = "";
24 };
25
26 #endif
```

درخواست های کاربر از نوع این کلاس هستند که شامل زمان ، مبدأ و مقصد می باشد .

Station.hpp

```
class Station
{
public:
    void SetName(std::string Name);
    void SetBusStatus(bool Status);
    void SetTaxi_SubwayStatus(bool Status);
    std::string GetName();
    bool GetBusStatus();
    bool GetTaxi_SubwayStatus();
private:
    std::string name = "";
    bool bus = false;
    bool taxi_subway = false;
};

#endif
```

پس از خواندن اطلاعات از فایل Stations.txt این اطلاعات در وکتوری از نوع این کلاس ذخیره می شوند که برای پر کردن وکتور path و ماتریس های مجاورت از آن استفاده می شود.

Vehicle.hpp

```
3
4  class Vehicle
5  {
6  public:
7      void setCost_for_each_line (int);
8      void setCost_for_each_line_Traffic (int);
9      void setMinute_per_km (int);
10     void setMinute_per_km_Traffic (int);
11     void setMinute_gettingOn_gettingOff (int);
12     void setMinute_gettingOn_gettingOff_Traffic (int);
13     int getCost_for_each_line ();
14     int getCost_for_each_line_Traffic ();
15     int getMinute_per_km ();
16     int getMinute_per_km_Traffic ();
17     int getMinute_gettingOn_gettingOff ();
18     int getMinute_gettingOn_gettingOff_Traffic ();
19 private:
20     int cost_for_each_line;
21     int minute_per_km;
22     int minute_gettingOn_gettingOff;
23     int cost_for_each_line_Traffic;
24     int minute_per_km_Traffic;
25     int minute_gettingOn_gettingOff_Traffic;
26 };
27 class Bus : public Vehicle
```

سه کلاس Taxi , Subway , Bus از این کلاس ارث بری می کنند که اطلاعات مربوط هر سه نوع وسیله نقلیه در این کلاس ها ثبت شده است .

Time.hpp

```
class Time
{
private:
    std::string Hour = "";
    std::string Minute = "";
    std::string am_pm = "";
public:
    Time(std::string,int,int);
    Time();
    friend Time operator+(Time const&,int);
    void operator=(Time const&);
    int setHour(int);
    int setMinute(int);
    void setAm_Pm(std::string);
    std::string getHour();
    std::string getMinute();
    std::string getAm_Pm();
    void printTime();
};
```

از این کلاس برای تبدیل زمان 12 ساعته به زمان 24 ساعته استفاده می شود همچنین با `overwrite` کردن علامت + میتوان هر مقدار دلخواهی بر حسب دقیقه را به زمان مورد نیاز اضافه کرد که از آن برای محاسبه `arriving time` استفاده می شود.

Hash.hpp

```
6  class Hash
7  {
8  private:
9      Station hashTable[N];
10     int maximum_i = 0;
11 public:
12     Hash();
13     int hashFunction(Station);
14     int returnKey(std::string);
15     int getMaximumI();
16 };
17
18
```

تابع hashFunction این کلاس با دریافت نام ایستگاه کلید متناظر با آن را تولید و در hashTable اطلاعات ایستگاه را ذخیره می کند. تابع returnKey هم با دریافت نام ایستگاه کلیدی که قبلا تولید شده و متناظر با ایستگاه هست را برمی گرداند و در صورت موجود نبودن ایستگاه عد -1 را بر می گرداند.

Routing.hpp

```
class Routing
{
private:
    Hash hash;
public:
    void Split(std::string record, char delimiter , std::vector <Station> *S);
    // split function record is one line from Stations file
    // delimiter is _
    // S is our station vector
    void readFromFile(std::vector <Station>*,std::string); // read from a file then split each line
    int findIndex(std::string,std::vector <Station>*); // finds the index of a specific station in the vector
    void runAlgorithm();
};
```

پر کردن وکتور های stations , path ، گرفتن درخواست ها از کاربر و ذخیره سازی آن ها در Queue و فراخوانی توابع Dijkstra در تابع runAlgoritm این کلاس انجام می شود.

منابع

از سایت هایی مانند <https://stackoverflow.com> ، www.geeksforgeeks.org ، www.freecodecamp.org در ساخت این پروژه استفاده شده است.

همچنین از

<https://www.dropbox.com/scl/fi/ow2txw0hvswchttazpux6/Git.paper?rlkey=583imvymr2lb4hrv7dbjnb6ke&dl=0>

برای آموزش دستورات گیت کمک گرفته شد.

تمامی فایل های برنامه در گیت هاب اعضای گروه بارگذاری شده است :

<https://github.com/Mister-Me/samhanshat>

https://github.com/amirhosseinbt/Routing_DataStructure_Project

با تشکر از زحمات گرانقدر استاد الهام افشار و دستیاران آموزشی ایشان