
SMNHSH

به نام خدا

نام درس: ساختمان داده

نام استاد: الهام افشار

تهیه شده توسط دانشجویان:

علی عزیزخانی ۴۰۱۱۲۳۵۸۰۲۷

یاسمن صفریان ۴۰۱۱۲۳۵۸۰۲۴

امیرمحمد ظفری۴۰۰۱۲۳۵۸۰۲۵

ترم پاییز ۱۴۰۲

مقدمه

در درس مذکور، استفاده از انواع ساختمان‌های داده را باتوجه‌به نوع مسئله یاد گرفتیم. شیوه‌های مختلفی برای حل یک مسئله می‌تواند وجود داشته باشد. در این مسئلهٔ به‌خصوص، نقشهٔ مسیر داده شده، برای ما نمونه‌ای از گراف به نظر می‌رسید. بدین ترتیب، الگوریتم دایجسترا برای یافتن کوتاه‌ترین مسیر و همچنین استفاده از پایهٔ ایدهٔ این الگوریتم برای یافتن "کمترین‌ها" به چشم می‌آمد. علاوه بر حل کلی مسئله، سعی داشتیم روندی برای حل پیش بگیریم تا توسعه‌پذیری برنامه بیشتر باشد. برای ذخیره‌سازی اطلاعات نیز شیوه‌های بسیار متنوعی وجود داشت که در ادامه دلیل شیوهٔ استفاده شده ذکر خواهد شد.

در این پروژه ما سعی داریم چندین ورودی با عناوین ساعت شروع سفر، مبدأ و مقصد نهایی را از کاربر دریافت کنیم و سپس کمترین مسافتی که کاربر می‌تواند پیمایش کند و زمان تقریبی رسیدن از این طریق، مسیری که کمترین هزینه را داشته باشد و در نهایت مسیری که پیمایش آن کمترین هزینهٔ زمانی را برای مخاطب دارد را به‌عنوان خروجی تولید کنیم.

چکیده

همان‌طور که گفته شد، قصد داریم با ورودی‌های داده شده توسط کاربر، کمترین هزینه، کمترین مسافت و کمترین زمان ممکن برای رسیدن، به‌علاوه زمان تقریبی برای کمترین مسافت را به دست آوریم.

برای نگهداری اطلاعات نقشه، از فایل‌های txt استفاده شده است که در مجموع ۵ فایل داریم که سه تای آنها حاوی ایستگاه‌ها و خط سه وسیلهٔ subway ، bus و taxi نگهداری می‌شود. سه وسیله در سه فایل جداگانه هستند.

برای نگهداری زمان موردنیاز برای جابه‌جایی و یا سوارشدن به وسایل و همچنین هزینهٔ لازم نیز یک فایل را اختصاص داده‌ایم، این کار موجب می‌شود تا در صورت تغییر در نرخ زمانی و مالی هر وسیله، فقط احتیاج به ویرایش فایل و تغییر ارقام داشته باشیم.

برای نگهداری ایستگاه‌ها از unordered_map استفاده می‌کنیم تا ماتریس مجاورت را اندیس‌گذاری کنیم.

برای این کار، سه کلاس جداگانه به نام‌های DataType، Operation و Dijkstra تعریف کرده‌ایم.

برای اجرای برنامه، از فایل SMNHSH استفاده می‌کنیم.

در این پروژه، ما برای هر یک از خواسته‌ها، از کلاس Dijkstra استفاده کرده‌ایم و الگوریتم مبنای هر چهار عملیات دایجسترا است؛ ولی در نوع جای‌گذاری‌ها تفاوت و شروط بسیار متفاوتی وجود دارد.

بخش گرافیکی پروژه، با استفاده از خط کدهای JavaScript و استایل Css نوشته شده است که در آن از فایل‌های Json هم استفاده شده است که تمام اینها با استفاده از انجین Construct3 به هم مربوط می‌شوند.

کلاس‌های برنامه

DataType

```
class DataType
{

private:

    int distance=0;
    int cost=0;
    string path{};
    string line{};
    vector <int> times;
    vector <string> time_type;
    vector <string> time_line;

public:

    void set_dis (int);
    void set_cost(int);
    void set_time(int);

    void set_pathh    (string);
    void set_line     (string);
    void set_timeLine(string);
    void set_timeType(string);
    void free_time();


    int get_cost();
    int get_dis ();

    vector<int> get_time();

    vector<string> get_timeLine();
    vector<string> get_timeType();

    string get_path();
    string get_line();

};
```

ماتریس مجاورت از این نوع کلاس است. چرا که ما نیاز داریم بین هر دو ایستگاه، مسافت و وسیلهٔ حمل‌ونقل مربوط به مسافت به‌علاوه Line موجود را نگهداری کنیم. این اطلاعات برای به‌دست‌آوردن هزینه به کمک ما خواهند آمد. ازآنجایی‌که بین هر دو ایستگاه ممکن است بیش از یک وسیله برای حرکت موجود باشد، احتیاج به خانه‌هایی از حافظه داریم که برای جلوگیری از اتلاف حافظه از vector استفاده کرده‌ایم؛ چون اعضای کلاس private هستند، توابع مربوطه برای مقداردهی و یا گرفتن این اعضا نوشته شده‌اند.

چون برای مسافت ما فقط نیاز به نگهداری کمترین مسافت بین دو ایستگاه را نیاز داریم، احتیاج به استفاده از vector نیست و از یک متغیر استفاده می‌کنیم.

SaveDirect

```
using namespace std;
class saveDirect
    {
    public:
    int distance {__INT_MAX__};
    vector<string> direct;
    vector<string> type;
    vector<string> line;
    vector<int> arr_time;
    };
```

این کلاس شامل vectorهایی است که مسئول نگهداری نوع وسیله، خط موردنظر، زمان رسیدن و مسافت آن است و متغیر distance که در کلاس Dijkstra کاربرد دارد.

SaveType

```
class Save_data
{
    private:
    string station_name;
    vector<string> vehicle;
    vector<string> vehicle_line;
    public:
    void set_type(string vehicle);
    void set_line(string vehicle_line);
    void set_name(string neme);
    string get_name();
    vector<string> get_type();
    vector<string> get_line();

};
```

این کلاس برای یک node در نقشه، به ما خواهد گفت که از این ایستگاه چند وسیله با چه خط هایی عبور می‌کند.

برای به دست آوردن کمترین زمان از این کلاس برای نگه داری اطلاعات استفاده می‌کنیم.

Operation

در این کلاس عملیات هایی همچون خواندن از هر کدام از فایل‌ها و مقداردهی به ماتریس مجاورت وجود دارند.

چون پر کردن ماتریس برای به‌دست‌آوردن کمترین مسافت، هزینه و زمان با هم متفاوت است، توابع مقداردهی به ماتریس جداگانه نوشته شده‌اند.

پیش از به‌دست‌آوردن نتیجهٔ مدنظر، باید set Item مربوط به آن عملیات فراخوانی شود تا مقادیر درست در ماتریس قرار بگیرند. در این تابع هر سه فایل مربوط به وسیله‌ها از فایل خوانده و اطلاعات مربوط به آن‌ها توسط تابع read عملیات مذکور، خوانده و جاگذاری می‌شود؛ بنابراین به تعداد فایل وسایل موجود که در این مسئله سه نوع bus, subway , taxi هستند، عملیات خواندن از فایل صورت می‌گیرد.

SMNHSH

در واقع این فایل، main برنامه است که برای ارتباط با قسمت گرافیکی و فراخوانی توابع مربوط سر و کار دارد.

Dijkstra

این کلاس، دارای توابعی است که در توابع دایجسترا از آن‌ها استفاده می‌شود، تابع search و minDistance.

تابع search با جستجو در inputMap، ایستگاه مربوط به آن اندیس را بر می‌گرداند.

تابع minDistance باتوجه‌به گره‌های دیده نشده، گره‌ای که با کمترین هزینه (منظور از هزینه، هزینه زمانی و مسافتی نیز خواهد بود) به آن رسیده‌ایم را برمی‌گزیند تا ادامهٔ مسیر را از آن گره پیش برود.

Dijkstra

برای به‌دست‌آوردن کمترین مسافت از الگوریتم دایجسترا به شیوه و شروط معمول آن استفاده شده است. در این تابع هزینه ی یک ایستگاه تا تمامی خطوطی که به آن ها متصل است را پیمایش و بررسی می‌کند و سپس الگوریتم دایجسترا اعمال می‌شود.

اندیس‌های src و dest به ترتیب اندیس‌های مبدا و مقصد هستند که از main به تابع فرستاده می‌شوند.

inputMap، همان unorderd_map برنامه است که از طریق hash_function اندیس‌ها را از ۰ تا ۵۸ به نام ایستگاه‌ها اختصاص داده است.

پیش از فراخوانی این تابع، مقادیر مربوطه توسط کلاس Operation در ماتریس station جای‌گذاری شده‌اند.

Dijkstra_cost

برای به دست آوردن کمترین هزینه ی بین مبدا و مقصد از این تابع استفاده می‌شود.

چون ممکن است بین دو ایستگاه بیش از دو قیمت وجود داشته باشد و انتخاب یکی از آن ها به مسیر گذرانده شده مربوط است، تمامی هزینه هایی که بین دو ایستگاه هست را بررسی می‌کنیم.

برای هزینه، ابتدا باید بررسی کنیم تا اندیس j را چه طور پیمایش کرده ایم. آیا مسیری که پیش روی ماست با همان وسیله ی قبلی و همان خط قبلی تطابق دارد؟ پاسخ به این سوالات را در حلقه های شرط درون حلقه ی پیمایش روی vector هزینه ی دو ایستگاه قرار داده ایم.

Dijkstra_Artime

این تابع زمان تقریبی رسیدن را اگر که از مسیر کمترین مسافت برویم، به ما خواهد داد. مانند روند کمترین مسافت، الگوریتم را اجرا می‌کنیم ولی در هر بخش با توجه به مسیری که انتخاب کرده ایم، تغییر خط و یا وسیله ی حمل و نقل را تشخیص می‌دهیم و در انتها تمام بازه های زمانی طول کشیده را با هم جمع می‌کنیم که حاصل در متغیر tut_time خواهد بود .

Dijkstra_time

محسابه ی زمان، پیچیدگی های بیشتری نسبت به دو تابع قبل برای ما دارد، لذا احتیاج به بررسی موارد بیشتری خواهد بود. چون در این تابع هم نیاز به بررسی تمامی هزینه های زمانی بین دو ایستگاه است، از vector هایی که تمامی ایستگاه های خطوط را شامل می‌شوند، استفاده می‌کنیم. در انتخاب مسیر در این تابع، باید بررسی کنیم که تا اندیس j با چه وسیله ای و در کدام خط مسیر را پیموده ایم، به علاوه اگر حین مسیر زمان جابجایی بین وسایل داشته باشیم به صرفه تر خواهد بود یا اگر از ابتدا با وسیله ی دیگری پیمایش می‌کردیم؟ تمامی این شروط در تابع بررسی شده اند. به علاوه اینجا به جز مسیری که پیمایش می‌کنیم و اطلاعات آن، که در تمامی توابع قبلی نیز در آرایه ی dir نگهداری می‌شدند، احتیاج داریم برای وسایل حمل و نقلی که داریم جداگانه مسیر را نگهداری کنیم تا اگر حین پیمایش متوجه شدیم مسیر دیگری به صرفه تر بود، مسیر را تغییر دهیم.

بخش گرافیکی

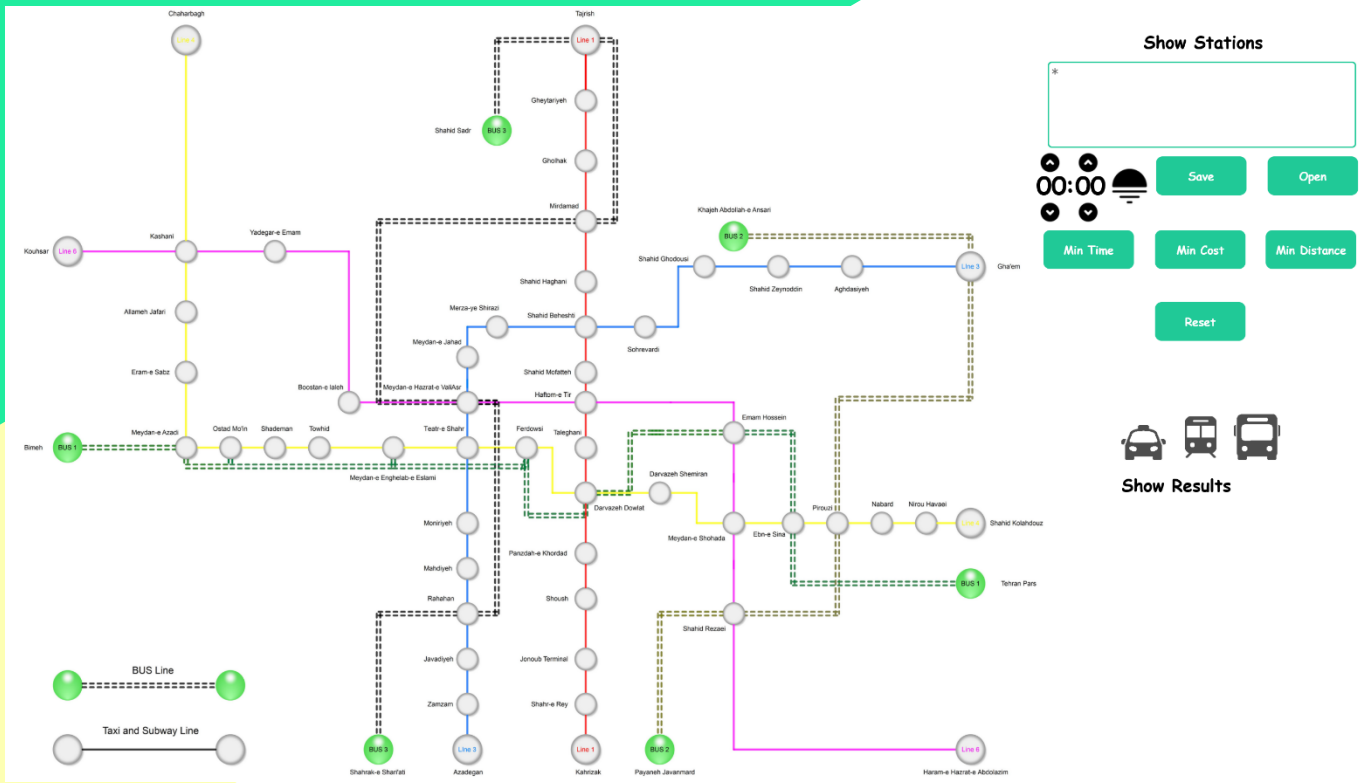
این بخش که پشت زمینه ی آن با JavaScript است و دکمه ها و textplain های آن با Html و Css استایل شده است.

کاربر نام دو ایستگاه را از روی نقشه انتخاب کرده و سپس زمان مورد نظرش را مشخص کرده و با زدن دکمه ی open فایل send_testCases.txt را انتخاب می کند و با زدن دکمه ی save این اطلاعات را در فایل send_testCases.txt ذخیره می کند ، تا هنگام کامپایل و اجرا کردن کد اصلی، این تست کیس ها از فایل send_testCases.txt خوانده شود و سپس برنامه ی اصلی جواب ها را در سه فایل

min_cost.txt , min_time.txt و min_dis.txt کـه بـه ترتیـب بـرای کمتـرین هزینـه ، کمتـرین زمـان و کمتـرین مسـافت هسـتند نوشـته تـا بـا اسـتفاده از دکمـه هـای مربـوط بـه نشـان دادن جـواب هـا ، فایـل مربـوط توسـط بخـش گرافیکـی برنامـه خوانـده شـود تـا جـواب هـا را بـر روی نقشـه و textplain مشـخص کنـد، سـپس بـرای انتخـاب دوبـاره ی تسـت کـیس دکمـه ی reset را زده و همان کار های قبلی را تکرار می کند.

(بـه علـت مجـاز نبـودن اسـتفاده از data base اینگونـه بـین بخـش گرافیکـی و فایـل اصـلی رابطه ایجاد شده است!)

بخـش گرافیکـی برنامـه بـه صـورت فایـل exe اکسـپورت گرفتـه شـده کـه بـر روی پلتفـورم هـای x86 , x64 و ARM64 قابل اجرا می باشد.
تمام این فایل های مربوطه نیز در پوشه ی SMNHSH_UI قرار دارد.



ارسال

https://github.com/AAzizkhani/SMNHSH