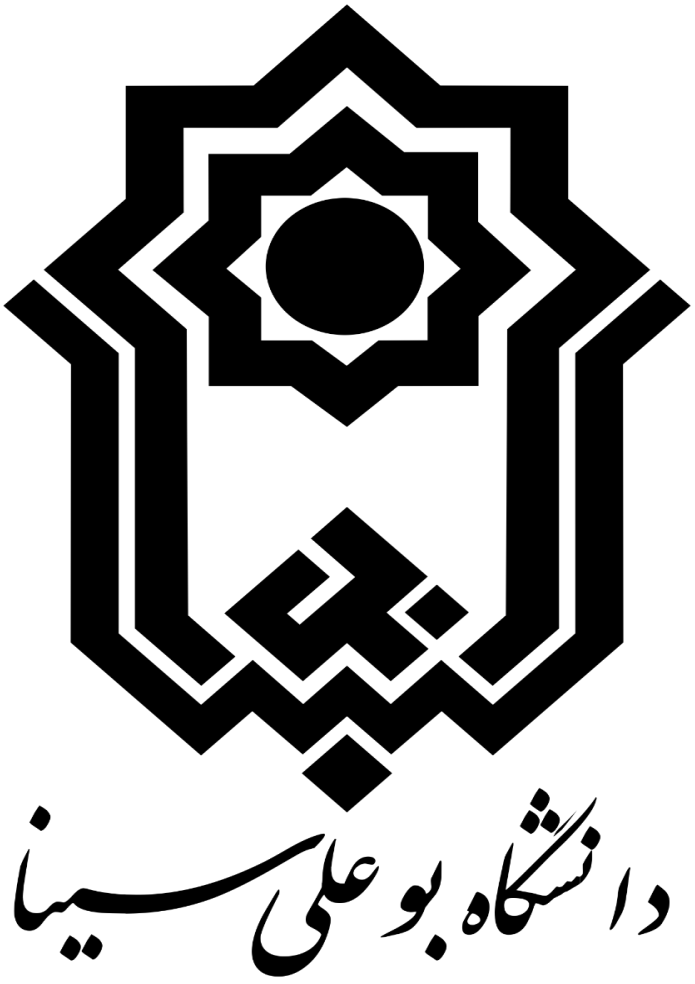
گزارش پروژه ساختمان های داده

برنامه مسیر یابی در شهر تهران



استاد درس: الهام افشار

اعضای گروه

دانیال کشاورز
سارا راشدی
فاطمه آقا حاصلی

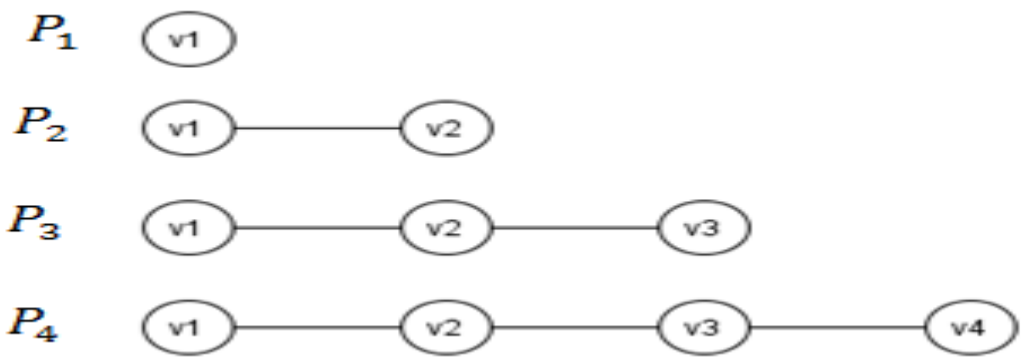
مقدمه

داستان samHan از پروژه ساختمان داده شروع شد این پروژه مسیریابی برای پیداکردن بهترین زمان، هزینه و مسافت بین تو دونقطه در تهران با وسایل نقلیه عمومی شروع شد در نهایت نه‌تنها توانست تمام رسالت‌های خودش را انجام بده؛ بلکه توانست یک برنامه باشد تا با مرتبه زمانی خوب O(n*k*lg k) جواب درست رو در هر نقشه‌ای بده برنامه‌ای که نه برای تهران بلکه برای نقشه هر شهر دیگر و نه فقط برای وسایل گفته شده بلکه قابل تکامل برای تمام وسایل نقلیه نوشته شد

الگوریتم پروژه

گراف مسیر

گراف خطی همان‌طور که از اسمش واضح است یک خط شامل که رشته‌ای از رئوس روی آن است؛ یعنی درجه هر رأس یک یا دو است این گراف، گراف مسیر هم گفته می‌شود وقتی یک وسیله نقلیه حرکت می‌کند قطعاً یک گراف مسیر از خودش به جا می‌گذارد؛ زیرا یک وسیله نمی‌تواند همزمان به چپ و راست برود یکی از نکات جالب در گراف مسیر در پیداکردن کوتاه‌ترین فاصله پیدا می‌شود برای پیداکردن فاصله فقط کافی است از روی نودی که هستیم پیمایش کنیم به چپ و راست تا به انتهای لاین برسیم



گراف شهری

گراف شهری مملو از وسیله‌های نقلیه و همان‌طور که بحث شد دریافتیم که هر وسیله نقلیه لزوماً یک گراف خطی است پس ما به گراف شهر به چشم یک گراف نگاه نمی‌کنیم؛ بلکه مجموعه‌ای از گراف‌های خطی هستند که می‌دانیم تک‌به‌تک فقط با O(n) کمترین مسافت رو توی آنها حساب کنیم

در این الگوریتم برای ما نقاطی مهم هستند به اسم نقاط تقاطع در این نقاط حداقل دو وسیله نقلیه با هم تداخل دارند این نقطه یک‌طور نقطه ورود ما به گراف خطی است و کافی است از همین نقاط ورود گراف خطی رو پیمایش کنیم تا به بهینه‌ترین جواب برسیم

برای اینکه انتخاب کنیم کدام نقطه تقاطع رو انتخاب کنیم باید از کوچک‌ترین هزینه شروع کنیم و برای این کار بهترین ساختمان داده صف اولویت‌دار است که در نهایت هزینه ورود و استخراج k نقطه تقاطع در نقشه klgk هستش و با ترکیب این دو عمل بالا با زمان O(n*k*lg k) می‌توانیم ادعا کنیم که به بهینه‌ترین جواب رسیدیم

پیاده سازی

وظایف و روابط بین کلاس های برنامه

در دیاگرام به وضوح روابط بین کلاس ها به تصویر کشیده شده اما برای دید بهتر به وظایف آنها نیز نگاه اجمالی خواهیم داشت

کلاس samhan : این کلاس کلاس controller در برنامه هست و توانایی نگهداری انواع شهر ها را دارد و به کاربر اجازه می دهد حتی در چند شهر مختلف مسیریابی همزمان داشته باشد

کلاس city: این کلاس نگهدارنده اصلی گراف شهر است وظیفه مدیریت وسایل نقلیه داخل شهر بر عهده همین کلاس است در محور اصلی برای بدست آوردن کمترین را بازی میکند

کلاس tehran:کلاسی ارث برده از city که در آینده امکان مدیریت کردن تفاوت در شهر ها را دارا باشد

خانواده کلاسهای vehicle: در خانواده وسیله نقلیه گراف خطی که وسیله در آن تردد می کند ذخیره شده است این کلاس توانایی محاسبه کمترین زمان ، هزینه و مسافت را در خود با توجه توانایی هایش داراست

کلاس Time: وظیفه نگهداری زمان راه دارد

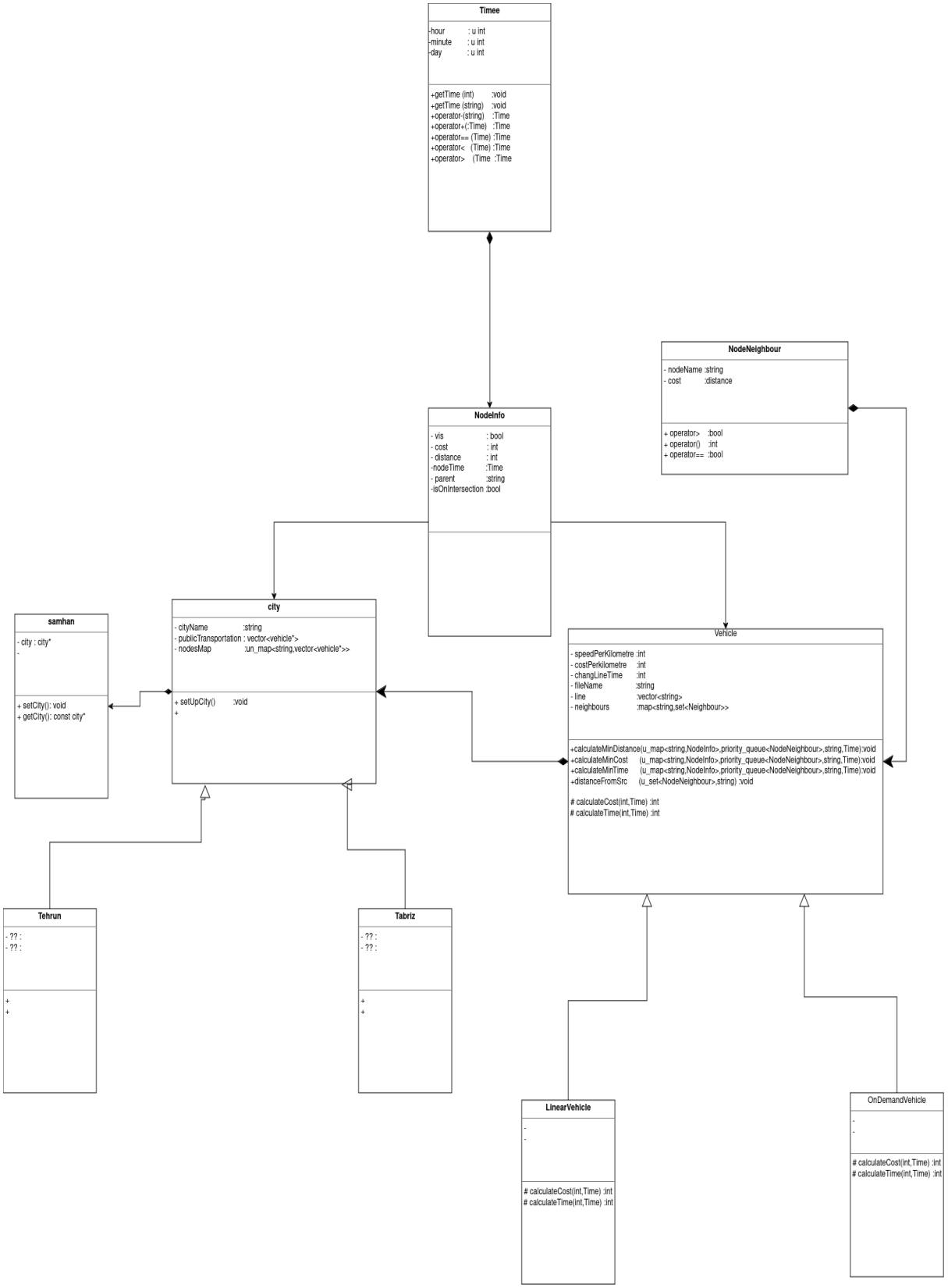
کلاس NodeInfo:در گراف در زمان مسیریابی از این کلاس به طور موقت ساخته میشود این کلاس به ما کمک کنیم بدانیم کمترین هزینه رسیدن به راس چقدر است با چه وسیله ای و از کجاست

کلاس nodeNeighber:وظیفه این کلاس یادآوری این است که راس ما به کدام رئوس راه دارد و با چه هزینه ای به آن میرسد

دیاگرام در فایل سورس پروژه نیز موجود است

.

طرح کلی برنامه



توابع مهم کلاس City

```
unordered_map <string , NodeInfo > City::calculateMin(const string& start,const string& end,MeasurementMetric metric,Time currentTime)
{
    unordered_map <string , NodeInfo >dijkstraTable ;
    priority_queue<NodeNeighbour,vector<NodeNeighbour>,greater<NodeNeighbour>> minHeap;

    for(const auto& i : this->nodesMap)
        dijkstraTable[i.first].setIntersection((i.second).size()>2);

    dijkstraTable[start].setCost(0);
    dijkstraTable[start].setDistance(0);
    dijkstraTable[start].setTime(currentTime);
    minHeap.push({start,0});

    while(!minHeap.empty())
    {
        string strMinNode = minHeap.top().nodeName;
        minHeap.pop();

        const auto& vihecleVec {nodesMap[strMinNode]};

        for(const auto& currentVihecle : vihecleVec )
        {
            switch(metric)
            {
                case DISTANCE:
                    currentVihecle->calculateMinDistance(dijkstraTable,minHeap,strMinNode,currentTime); break;
                case COST:
                    currentVihecle->calculateMinCost    (dijkstraTable,minHeap,strMinNode,currentTime); break;
                case TIME:
                    currentVihecle->calculateMinTime    (dijkstraTable,minHeap,strMinNode,currentTime); break;

                default :
                    cerr << "City::calculateMin erorr\n\n";
            }
        }

        dijkstraTable[strMinNode].setVis(true);
    }

    return dijkstraTable;
}
```

هدف اصلی این تابع پیدا کردن بهینه ترین مسیر در گراف شهری است در ابتدا هر راس به اطلاعات خودش مپ می شود و یک صف اولویت برای نگهداری نقاط تقاطع ایجاد میشود در for تمامی راس های گراف شهری در جدول ذخیر می شوند اگر راسی روی تقاطع باشد در همین جا علامت زده میشود و flag آن ست خواهد شد راس مبدا هزینه رسیدن آن معین می شود
تا زمانی که صف اولویت خالی نشده کم هزینه ترین راس را استخراج و به ازای تمام وسایل نقلیه ای که از آن ایستگاه به آن دسترسی داریم هر وسیله نقلیه گراف خطی خود را بروزرسانی میکند وظیفه بروزرسانی جدول کلی و صف اولویت نیز به وسایل سپرده میشود یعنی هر وسیله در زمان پیمایش خود نقاط تقاطع را در min heap می ریزد و table را آپدیت می کند در نهایت پرچم دیده شدن نود ست می شود تا از بازدید مجدد آن جلوگیری شود

توابع مهم کلاس Vehicle

```
void Vehicle::calculateMinTime(unordered_map<string,NodeInfo>& table,priority_queue<NodeNeighbour,vector<NodeNeighbour>
,greater<NodeNeighbour>>&minHeap, const std::string& srcNode ,Time currentTime)
{
    unordered_set<NodeNeighbour,NodeNeighbour::myHash> distanceSet;
    distanceFromSrc(distanceSet,srcNode);


    for(const auto &currentNode : distanceSet)
    {

        if(!table[currentNode.nodeName].getVis() &&
            table[currentNode.nodeName].getTimeInt() > table[srcNode].getTimeInt() + calculateTime(currentNode.distance,currentTime))
        {
            modifyDijkstraTable(table,srcNode,currentNode.nodeName,currentNode.distance,currentTime);
            if(table[currentNode.nodeName].getIntersection())
                minHeap.push({currentNode.nodeName,table[srcNode].getTimeInt() + calculateTime(currentNode.distance,currentTime)});
        }
    }
}
```

در هر وسیله برای پیدا کردن بهترین زمان از تابع زیر استفاد می شود در این تابع اول یک ست از نقاط ذخیره می کنیم در این ذخیره سازی اسم هر راس و فاصله آن از مبدا ذخیره میشود مثلا فرمت (Kashani ,9) یعنی مبدا از ایستگاه کاشانی ۹ کیلومتر فاصله دارد این مجموعه یک دور پیمایش میشود در شرط زمان رسیدن با وسیله نقلیه فعلی با زمان کنونی مقایسه می شود اگر زمان بهینه تری پیدا کرده باشیم در بدنه شرط اطلاعات وسیله جدید درون جدول جایگذاری میشود و اگر راس مورد نظر بر تقاطع باشد داخل صف اولویت ریخته میشود

لازم به ذکر است صف اولویت و جدول دایجسترا هر دو به صورت رفرنس پاس داده شده اند تا تغییرات را بعدا در کلاس city بازتاب دهند

```
void Vehicle::distanceFromSrc(unordered_set<NodeNeighbour,NodeNeighbour::myHash>& distanceSet,const string& srcNode)
{
    unordered_set<string> visitedNodes;
    queue <NodeNeighbour> searchQueue ;
    searchQueue.push({srcNode,0});

    while(!searchQueue.empty())
    {
        NodeNeighbour currentNode = searchQueue.front();
        searchQueue.pop();

        for(const auto& item :  neighbours[currentNode.nodeName])
            if(!visitedNodes.count(item.nodeName))
                searchQueue.push({item.nodeName,currentNode.distance+item.distance});

        visitedNodes.insert(currentNode.nodeName);
        distanceSet.insert (currentNode);
    }
}
```

این تابع وظیفه پیدا کردن فاصله هر راس از مبدا ، تا بقیه ایستگاه های موجود در مسیر وسیله را دارد این تابع با استفاده از الگوریتم BFS پیاده سازی شده است یعنی ابتدا به نود مبدا میرویم و همسایه های آن را به صف اضافه میکنیم سپس آن ایستگاه رو در لیست visit شده ها قرار می دهیم سپس از صف عضوی را بیرون می کشیم و دوباره الگوریتم را تکرار میکنی تا تمام گراف را به صورت عرضی سرچ کنیم

توابع مهم فرزند کلاس Vehicle

```
//*****************protected
int OnDemandVehicle::calculateCost(int km,Time currentTime)
{
    float trafic = (isOnTrafic(currentTime)? 1.5 :  1);
    return km*getCostPerKilometre()*trafic;
}
int OnDemandVehicle::calculateTime(int distance,Time currentTime)
{
    float trafic = (isOnTrafic(currentTime)? 2 :  1);
    return Vehicle::calculateTime(distance,currentTime)*trafic;
}
bool OnDemandVehicle::isOnTrafic (Time current)
{
    return (Time("06:00 pm") < current && current < Time("08:00 pm"));
}
```

هر کدام از فرزندان این کلاس خود نماینده یک وسیله نقلیه هستند با توجه به انعطاف پذیری الگوریتم مسیریابی نیازی به override کردن توابع اصلی مسیریابی در کلاس های فرزند نیست اما هر فرزند می تواند مشخصات شخصی خود را داشته باشد به طور مثال توابع محاسبه هزینه ، زمان و تایم ترافیک در یکی از کلاس های فرزند بازنویسی شده
این دیزاین برنامه نویس را قادر می سازد با ارث بری به سادگی و سرعت دستگاه های نقلیه جدید را به برنامه اضافه کند

امتیازی های پروژه

پروژه در بخش امتیازی داشت که هر دو بخش آن انجام شده اولین مورد گرافیک پروژه بود که با استفاده از QtQuick و Qml پروژه زده شد در نسخه دسکتاپ برنامه کاربر با نقشه تهران مواجه می شود و می تواند ایستگاه های خود را انتخاب کند و خروجی با رنگی شدن یال های مربوطه با خروجی های زمان ، فاصله و هزینه به کاربر اعلام می شود

دومین بخش امتیازی مربوط به هش بود لازم به ذکر است در جای جای این پروژه از ساختمان داده هایی استفاده شده که از هش برای ذخیره سازی استفاده میکند و برنامه را قادر ساختن با O(1) توانایی پیدا کردن یک راس و همسایه های آن را داشته باشد اما پا را فراتر گذاشته و برای برخی کلاس های خود برنامه نیز توابع هش نوشته شده است تا درهم سازی به بهینگی برنامه بیافزاید

سخن آخر

از تمام دوستان و همکلاسی ها و اساتیدی که به ما در این پروژه یاری رساندن کمال تشکر را داریم استاد افشار تیم تی ای ایشان بچه های بی نظیر ورودی ۴۰۱ که از هیچ کمکی به تیم ما دریغ نکردن و تشکر ویژه تر از هم تیمی هایم خانم سارا راشدی و خانم آقا حاصلی که این پروژه بر نتیجه زحماتش بنا شد و گربه های سیاه، طوطی های سفید و وال های آبی مهربانی که مشوق ما در این پروژه بودن

لینک گیتهاب پروژه

github.com/DKeshavarz/DataStructureProject

زمستان ۱۴۰۲