Get on the Bus 2

New Mexico

Supercomputing Challenge

Final Report

April 3, 2010

Team 56

Homeschool

Team Member:

Isaac Koh

ifckoh@gmail.com

Teacher:

Aik-Siong Koh

askoh@askoh.com

Mentor:

Aik-Siong Koh

Summary:

This project was to create a web based version of my Atomic City Bus Scheduler and allow the user to type in his/her starting address and their destination's address and the program would tell them which bus stop is closest and how to take the bus from that stop. This also included a Google Map that would mark the start and end address and the two bus stops needed on the map so the user could see where they had to go.

Initially, the user had to select his/her starting bus stop, their destination and the time they want to arrive by. I created Routes which contained Trip Patterns and the program generated the Trips from the Patterns. The final product was successful and could correctly tell you how to take the bus. All of this was using Microsoft Visual C# 2008 Express Edition, which meant that this only ran on the hard drive and had to be downloaded. I now decided move the entire system into a version that could on a web browser.

Using Microsoft Visual Web Developer 2008 Express allowed me to move my C# code with only a slight change in code. This first version ran well, but I had hard coded all of the Routes and destinations into the main code body. This worked for the Atomic City Transit because it only had five routes to input, but to expand to Park and Ride, Santa Fe, Espanola, or any larger system would be a very tedious job for data input. This led me to read code in from a CSV file. I could now put the data in a spreadsheet and saved it as a .csv file. This allowed me to write 10 lines of code to read the file and get rid of the 700 lines that had been used to store the Routes. This also simplified data entry considerably (See example).

In order to read the spreadsheet consistently I created a Location class and a TransportSystem class. This will allow me to keep the Espanola and Santa Fe bus systems separate from each other and any other system. Each TransportSystem contains Routes with Trip Patterns and Trips, but instead of calling each stop a Destination we switched to Location. Location stores the latitude and longitude of each bus stop.

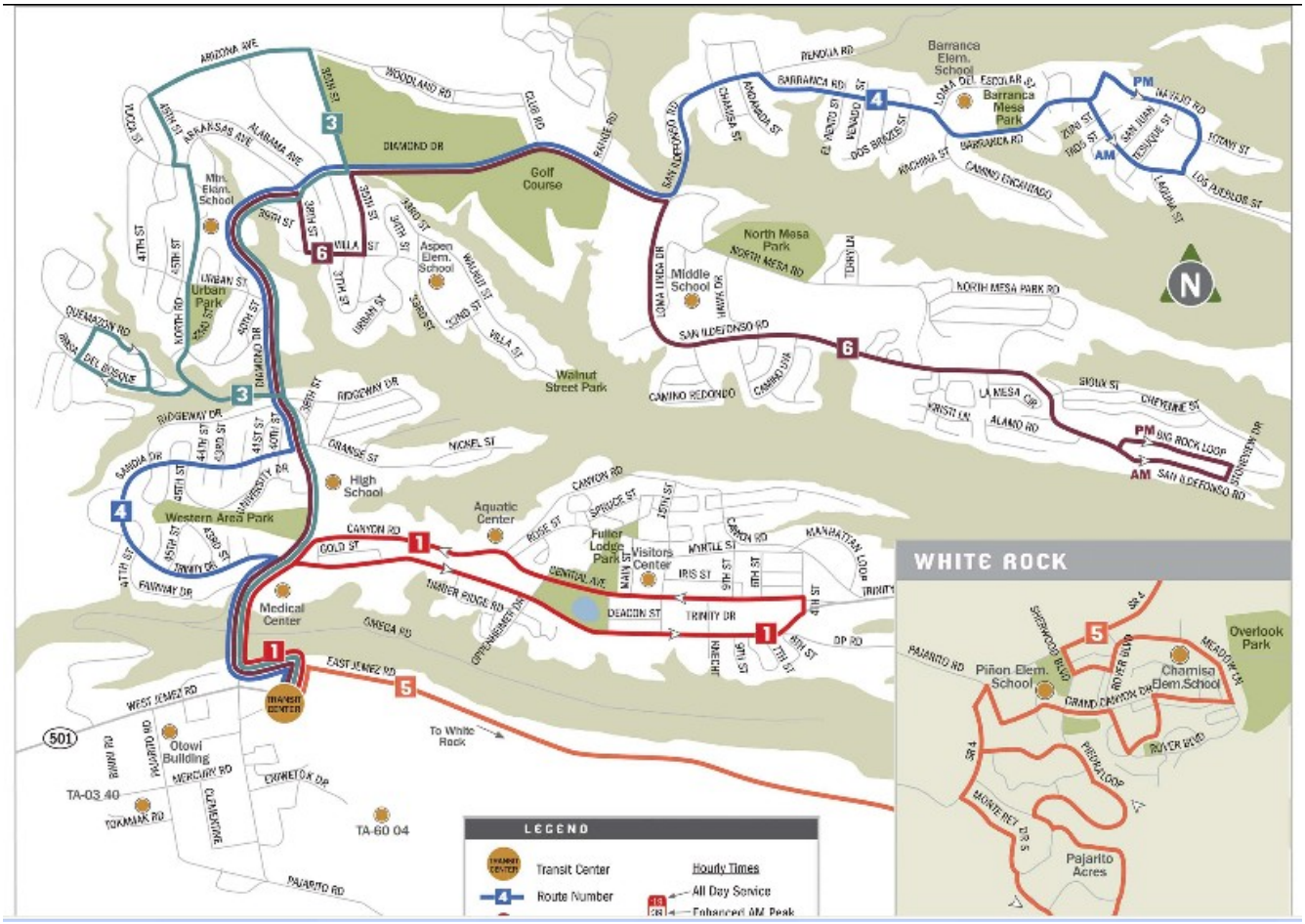
These modifications created a functional and rather clean web friendly version, but any user needed to know the bus stop nearest to them and the bus stop nearest their destination in order to use the system. To solve this I changed the program to accept typed in addresses and use the latitude and longitude to find the nearest bus stop. The program then tells you how to take the bus from that bus stop.

Problem Investigated:

I wanted to make a program for the Los Alamos Atomic City Transit bus system that lets the user input his/her starting place, destination, and the time they want to arrive. Then the program will return when and where to catch the bus, when and where to switch if necessary and when the user will arrive. Upon completing the program the next step was to put it on the Internet and increase usability by allowing users to type in their starting address and the address of their destination instead of having to know which bus stop was nearest to them and which stop was nearest their destination.

Classes (See Appendix A for code):

TransportSystem: Stores name, list of Routes, and list of Locations

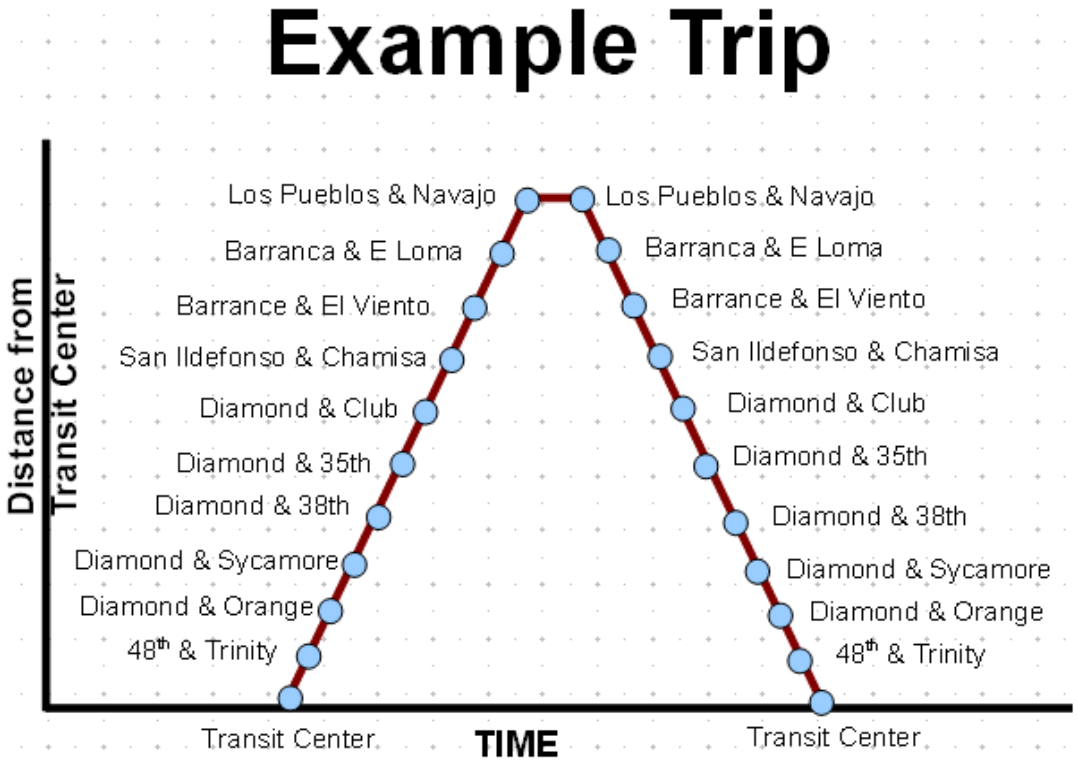


Data Example:

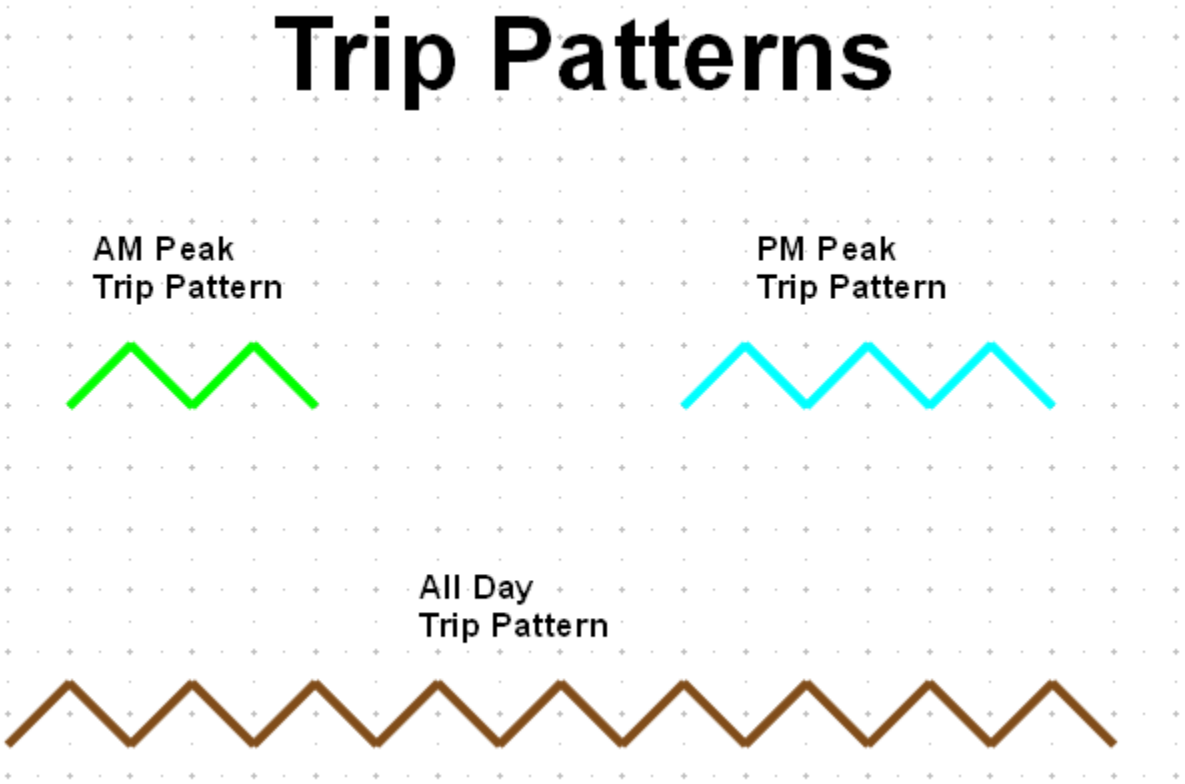
Route 1										
Location	Arrival Time	Wait Time	Repeat Time	First Repeat	Last Repeat					
Transit Center	3	5	20	340	1140					
Diamond & Trinity	10	0								
Trinity & Oppenheimer	11	0								
Trinity & 15th	12	0								
4th & Central	14	0								
Central & 15th	16	0								
Central & Canyon	18	0								
Diamond & Canyon	21	0								
Transit Center	23	5								
Route 3a										
Location	AM Regular Time	Wait Time	Repeat Time	First Repeat	Last Repeat	AM Peak	Wait Time	Repeat Time	First Repeat	Last Repeat
Transit Center	4	42	60	300	660	24	42	60	300	420
Diamond & Canyon	48	0				68	0			
Diamond & Orange/Sandia	49	0				69	0			
Diamond & Sycamore	50	0				70	0			
Diamond & 38th/Arkansas	52	0				72	0			
Diamond & 35th	52	0				72	0			
35th & Arizona	53	0				73	0			
North Rd & Mountain School	56	0				76	0			
North Rd & Urban	57	0				77	0			
Tranquillo & Quemazon	59	0				79	0			
Diamond & Orange/Sandia	61	0				81	0			
Diamond & Canyon	62	0				82	0			
Transit Center	64	42				84	42			

Location name	Arrival times of first trip pattern (in minutes after the hour)	(Minutes after Mid-night)	Second pattern (if necessary)

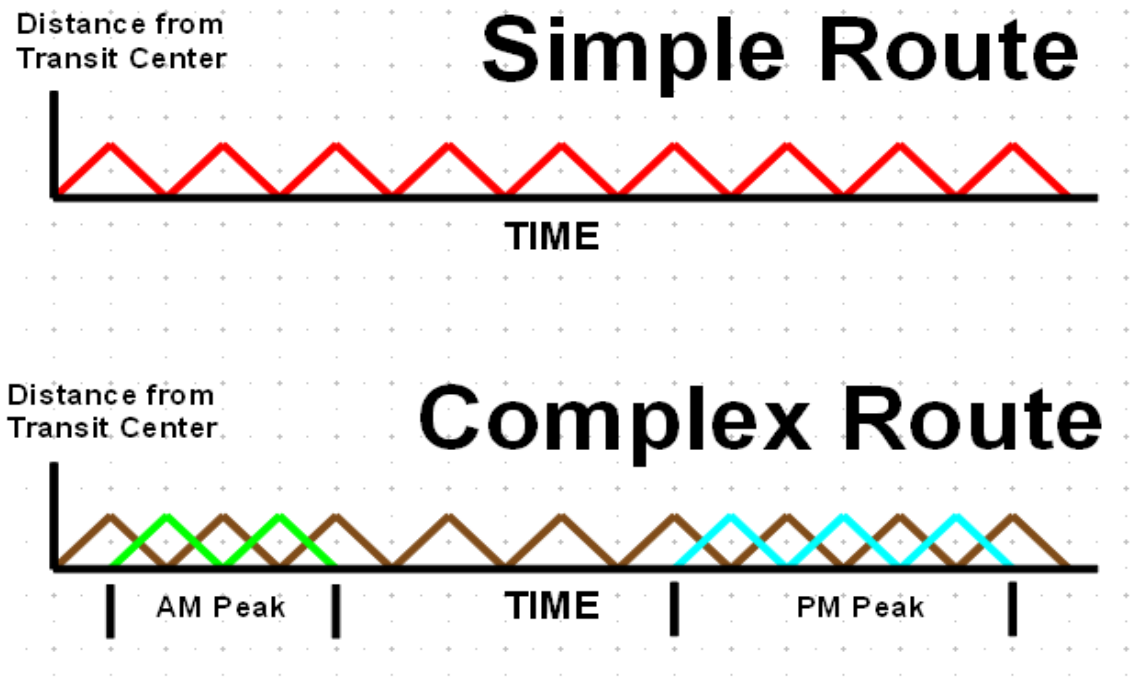
Trip: Name, Route it comes from, list of arrival times, and list of waiting times.



TripPattern (inherits Trip): Stores the number of minutes between each Trip, the time Trips start, the time they end, and a collection of the Trips of that day for that pattern.



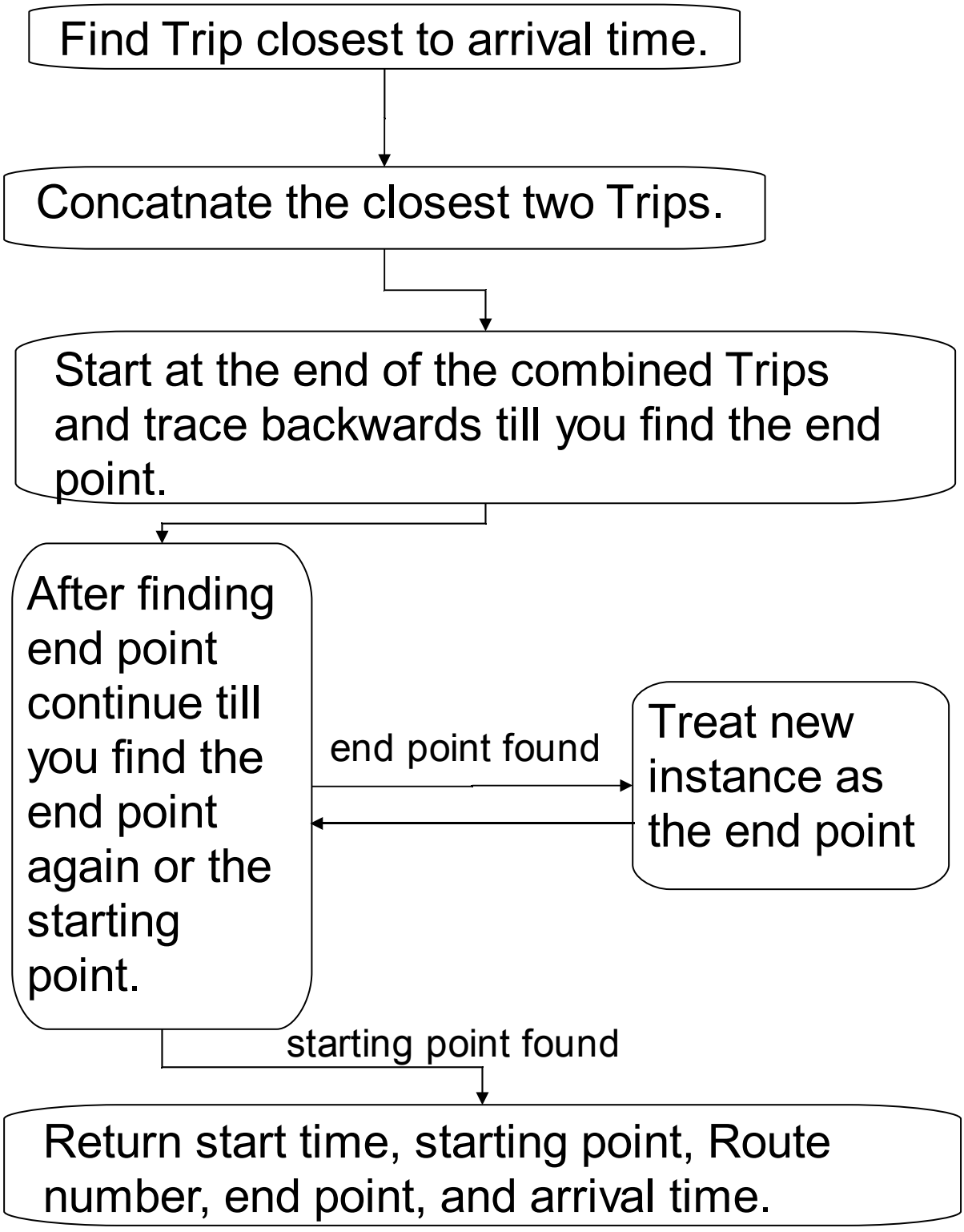
Route: Store name, list of Locations, list of TripPatterns, and list of Trips



Algorithm:

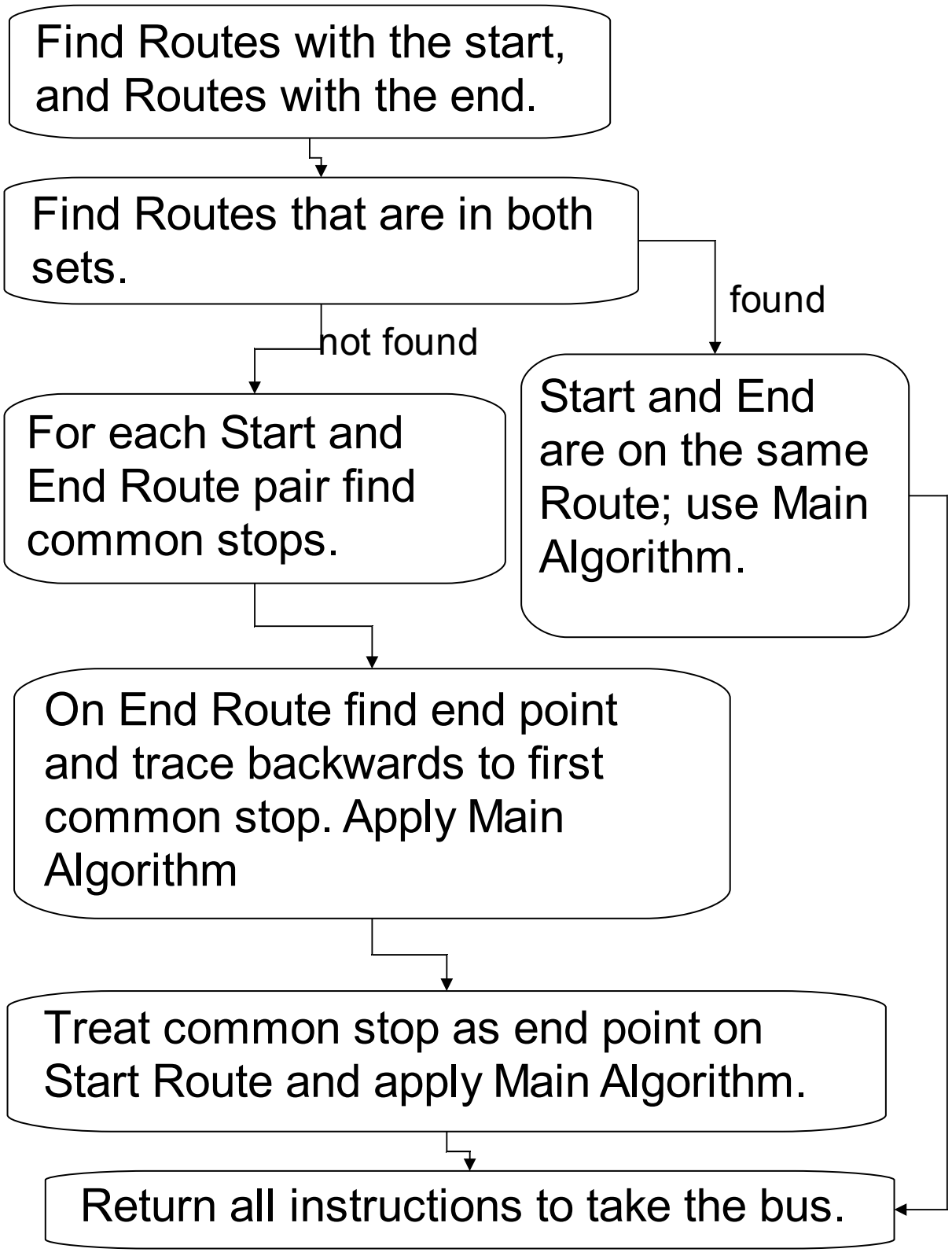
Main Algorithm

Condition: Start and End are on the same Route



Program:

Main Program



Final Result:

Los Alamos Atomic City Transit Scheduler

Start 3765 A Villa St Los Alam

End 1300 Diamond Dr Los A

Desired time of Arrival

Hour	Minutes
5	00
6	05
7	10
8	15
9	20
10	25
11	30
12	35
13	40
14	45
15	50
16	55
17	
18	
19	

OK Clear

Directions

Start Time is 16:58 "Diamond & 38th/Arkansas", "Route 3p"
End Time is 17:01 "Diamond & Orange/Sandia", "Route 3p"

Start Time is 17:35 "Diamond & 38th/Arkansas", "Route 4"
End Time is 17:37 "Diamond & Orange/Sandia", "Route 4"

Start Time is 17:37 "Diamond & 38th/Arkansas", "Route 6"
End Time is 17:39 "Diamond & Orange/Sandia", "Route 6"

Computational/Mathematical Model:

This program was created to solve a scheduling problem. The Routes are already given in the booklet and there is nothing more to add to that. There is no Traveling Sales Man complications since the program is for only two locations. There is no need to minimize distance for the bus since there is only one way to get there per Route. There is a maximum of three different Routes that a person can take in the Atomic City Transport to arrive at one destination. My program currently returns all three and lets the user pick which bus they want to take. There is no optimization between multiple Routes. The only criteria is arriving as close as possible but before the desired time, and this optimization occurs inside each individual Route.

Adding Google Map with a starting address not necessarily a bus stop made me create criteria for picking the best bus stop. Currently the program returns the closest stop. Closest is determined by the distance formula using the latitude and longitude of the starting address and all the stops. This means that the program is returning the closest bus stop as the crow flies. It does not take canyons or winding paths into account. Sometimes the bus stop it returns is across a large canyon. The next step will be to add in the new two hundred individual bus stops that have been added since the creation of this program. Once these new stops are added the probability that the user will be assigned a bus stop across a canyon is much lower, but also showing them a list of the closest three stops and letting them pick is another improvement.

Adding Google Maps:

The final change I decided to make this year was to remedy this. I removed the list of bus stops and put in two text boxes each paired with a Google Map. Now users can type in their starting address and the address of their destination instead of having to know the closest bus stop to their starting point and the closest stop to their destination. Now I used the latitude and longitude that is stored in each Location and compare it to the latitude and longitude of the input addresses. Now the program takes the bus stop (Location) nearest the starting address and the stop nearest the destination's address and run the the normal algorithm to tell you how to take the bus. So the final result has a pop up marker on the starting address and the nearest bus stop on the map paired with that text box. The second map has markers on the ending bus stop and final address. I was hoping to add the blue direction lines and get a path for people to follow, but that required Javascript and a lot of time that I didn't have.

Conclusion:

The program was successful and runs smoothly. I made a smooth transition from desktop application to a web based application. I learned how to add and program a Google Maps API in Visual Web Developer. I generalized data entry by allowing the program to read data from spreadsheets. I also made the program simpler by storing all of the Locations in one place instead of having the program find them again every time it ran. I added wait time to show when the bus is staying at a stop for any duration of time. The TransportSystem & Location classes are to make expansion easier and will allow me to keep Santa Fe's, Espanola's, and Park and Ride's collection of Routes separate from the Atomic City Transit(ACT).

Future Work:

I plan to input the two hundred new individual stops that have been put in place recently for the ACT. The county has also recently added a new Route that I have not included in the current version. Once these simple tasks are done I hope to get my map to show the best path to walk to the bus stop it returns. Once I get the path to show the plan is to give walking directions and a time estimate for the walk. Once I perfect the program for Los Alamos I plan to expand to Santa Fe, Espanola, and Park and Ride buses also.

Acknowledgments:

I would like to thank Aik-Siong Koh my teacher and mentor for helping and coaching me through this project.

Resources:

Microsoft Visual Web Developer 2008 Express Edition
Google Maps API
Subgurim.NET
Atomic City Transit Schedule
Compaq Presario V3000 Laptop

Works Cited

GoogleMaps.Subgurim.NET. http://en.googlemaps.subgurim.net/. Subgurim. 6 March 2010.

CTA Bus Tracker. http://www.ctabustracker.com/bustime/home.jsp. CTA. 10 March 2009.

Google Maps. http://maps.google.com/maps?hl=en&tab=wl. Google. 12 Nov. 2009.

Appendix A

TransportSystem:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AtomicCityBusScheduler
{
   public class TransportSystem
   {
      public string name;
      public List<AtomicCityBusScheduler.Location> cLocation;
      public List<Route> cRoute;
      public TransportSystem()
      {
         cLocation = new List<AtomicCityBusScheduler.Location>();
      }

      public AtomicCityBusScheduler.Location aLocationAt(string locationName)
      {
         return
         cLocation.Find(delegate(AtomicCityBusScheduler.Location obj)
            {
               return obj.name == locationName;
            });

      }

      public Location aLocationNearest(Subgurim.Controles.GLatLng latlon)
      {
         double minDistanceSQ = 10^100;
         Location bestLocation = null;
         foreach (Location l in cLocation)
         {
            double distanceSQ = Math.Pow((latlon.lat - l.latitude), 2) + Math.Pow((latlon.lng - l.longitude), 2);
            if (distanceSQ < minDistanceSQ)
            {
               minDistanceSQ = distanceSQ;
               bestLocation = l;
            }
         }
         return bestLocation;

      }
   }
}
```

Route:

```
using System;
using System.Collections.Generic;
//using System.Linq;
using System.Web;
using System.Text;

namespace AtomicCityBusScheduler
{

   public class Route
   {
      public List<AtomicCityBusScheduler.Location> cLocation;
      public string name;
      public List<TripPattern> cTripPattern;
      public List<Trip> cTrip;
      public Route()
      {
         cLocation = new List<AtomicCityBusScheduler.Location>();
         cTripPattern = new List<TripPattern>();
      }

      public List<Trip> setcTrip()
      {
         cTrip = new List<Trip>();
         foreach (TripPattern tp in cTripPattern)
         {
            cTrip.AddRange(tp.setcTrip());
         }

         // storing yesterday's trips.
         int nTrip = cTrip.Count;
         int day = 0;
         while (cTrip.Count < (nTrip + 4))
         {
            day = day + 1;
            for (int ii = 0; ii < nTrip; ii++)
            {
               Trip yTrip = cTrip[ii].Copy();
               for (int i = 0; i < yTrip.c1ArrivalTime.Count; i++)
               {
                  yTrip.c1ArrivalTime[i] = yTrip.c1ArrivalTime[i] - (day * 24 * 60);
               }
               cTrip.Add(yTrip);
            }
         }
         return cTrip;
      }


      public List<int> cLocationIndex(AtomicCityBusScheduler.Location aLocation)
      {
         List<int> cLocationIndex = new List<int>();
         List<AtomicCityBusScheduler.Location> cLocation = this.cLocation;
         for (int i = 0; i < cLocation.Count; i++)
         {
            if (aLocation == cLocation[i])
```

```
            {
                cLocationIndex.Add(i);
            }
        }
        if (cLocationIndex.Contains(0) & cLocationIndex.Contains(cLocation.Count - 1))
        {
            cLocationIndex.Remove(0);
        }
        return cLocationIndex;
    }

    public List<Trip> cTripNearestTime(AtomicCityBusScheduler.Location aLocation, int time)
    {
        List<Trip> cTripNearestTime = new List<Trip>();
        List<int> cLocationIndex = this.cLocationIndex(aLocation);
        for (int i = 0; i < cTrip.Count; i++)
        {
            Trip aTrip = cTrip[i];
            if (cLocationIndex.Exists(delegate(int index) { return aTrip.c1ArrivalTime[index] < time; }))
            {
                cTripNearestTime.Add(aTrip);
            }
        }
        int j = cLocationIndex[cLocationIndex.Count - 1];
        cTripNearestTime.Sort(delegate(Trip t1, Trip t2) { return t1.c1ArrivalTime[j].CompareTo(t2.c1ArrivalTime[j]); });
        return cTripNearestTime;
    }

    public List<int> startandEndTime(AtomicCityBusScheduler.Location startDest, AtomicCityBusScheduler.Location
endDest, int aArrivalTime)
    {
        List<Trip> cTripNearestTime = this.cTripNearestTime(endDest, aArrivalTime);
        int index = cTripNearestTime.Count - 1;
        Trip bestTrip = cTripNearestTime[index].Copy();
        JoinedTrip bestJoinedTrip = new JoinedTrip();
        bool found = false;
        while (!found)
        {
            index--;
            Trip previousTrip = cTripNearestTime[index];
            if (!(previousTrip.overlaps(bestTrip)))
            {
                found = true;
                bestJoinedTrip = previousTrip.Append(bestTrip);
            }
        }
        index = cTripNearestTime.Count - 2;
        Trip secondBestTrip = cTripNearestTime[index].Copy();
        JoinedTrip secondBestJoinedTrip = new JoinedTrip();
        found = false;
        while (!found)
        {
            index--;
            Trip previousTrip = cTripNearestTime[index];
            if (!(previousTrip.overlaps(secondBestTrip)))
            {
                found = true;
                secondBestJoinedTrip = previousTrip.Append(secondBestTrip);
```

```
            }
        }

        List<int> bestStartandEndTime = bestJoinedTrip.startandEndTime(startDest, endDest, aArrivalTime);
        List<int> secondBestStartEndTime = secondBestJoinedTrip.startandEndTime(startDest, endDest, aArrivalTime);
        bestStartandEndTime.AddRange(secondBestStartEndTime);
        return bestStartandEndTime;
    }

    public AtomicCityBusScheduler.Location joinStartandEnd(Route startRoute, AtomicCityBusScheduler.Location
startDest, AtomicCityBusScheduler.Location endDest)
    {
        List<AtomicCityBusScheduler.Location> csDest = new List<AtomicCityBusScheduler.Location>();
        List<AtomicCityBusScheduler.Location> ceDest = new List<AtomicCityBusScheduler.Location>();
        csDest.AddRange(startRoute.cLocation);
        ceDest.AddRange(this.cLocation);
        List<AtomicCityBusScheduler.Location> commonDest = new List<AtomicCityBusScheduler.Location>();
        foreach (AtomicCityBusScheduler.Location str in csDest)
        {
            if (ceDest.Contains(str))
            {
                commonDest.Add(str);
            }
        }
        int endIndex = -1;
        List<AtomicCityBusScheduler.Location> cEndDest = this.cLocation;
        for (int i = cEndDest.Count - 1; endIndex < 0; i--)
        {
            if (cEndDest[i] == endDest)
            {
                endIndex = i;
            }
        }
        int startIndex = -1;
        for (int i = endIndex - 1; (startIndex < 0) & (i >= 0); i--)
        {
            if (cEndDest[i] == endDest)
            {
                endIndex = i;
            }
            if (commonDest.Contains(cEndDest[i]))
            {
                startIndex = i;
            }
        }
        return cEndDest[startIndex];
    }

    public TripPattern addTripPattern(TripPattern aTripPattern)
    {
        cTripPattern.Add(aTripPattern);
        aTripPattern.aRoute = this;
        return aTripPattern;
    }

    public bool ContainsEnd(AtomicCityBusScheduler.Location endDest)
    {
        for (int i = 1; i < cLocation.Count; i++)
```

```
            {
                if (endDest == cLocation[i])
                {
                    return true;
                }
            }
            return false;
        }

        public bool ContainsStart(AtomicCityBusScheduler.Location startDest)
        {
            for (int i = 0; i < cLocation.Count - 1; i++)
            {
                if (startDest == cLocation[i])
                {
                    return true;
                }
            }
            return false;
        }
    }

}
```

Trip:

```
using System;
using System.Collections.Generic;
//using System.Linq;
using System.Web;
using System.Text;

namespace AtomicCityBusScheduler
{

   public class Trip
   {
      public string name;
      public Route aRoute;
      public List<int> c1ArrivalTime;
      public List<int> cWaitTime;
      public Trip()
      {
         c1ArrivalTime = new List<int>();
         cWaitTime = new List<int>();
      }

      public List<AtomicCityBusScheduler.Location> cRouteLocation
      {
         get
         {
            return aRoute.cLocation;
         }
      }

      public Boolean isAtDestNearTime(AtomicCityBusScheduler.Location aLocation, int aTime, int aTolerance)
      {
         for (int i = 0; i < cRouteLocation.Count; i++)
         {
            if (cRouteLocation[i] == aLocation)
            {
               int time = this.c1ArrivalTime[i];
               if ((time < aTime) & (time >= aTime - aTolerance))
               {
                  return true;
               }
            }
         }
         return false;
      }

      public Boolean overlaps(Trip bTrip)
      {
         int lastTime = c1ArrivalTime[c1ArrivalTime.Count - 1];
         int firstTime = c1ArrivalTime[0];
         if ((lastTime > bTrip.c1ArrivalTime[0]) & (lastTime < bTrip.c1ArrivalTime[c1ArrivalTime.Count - 1]))
         {
            return true;
         }
         if ((firstTime > bTrip.c1ArrivalTime[0]) & (firstTime < bTrip.c1ArrivalTime[c1ArrivalTime.Count - 1]))
         {
            return true;
```

```
    }
    return false;
}

public List<int> cTime(AtomicCityBusScheduler.Location aLocation)
{
    List<int> cTimeDest = new List<int>();

    for (int i = 0; i < cRouteLocation.Count; i++)
    {
        if (cRouteLocation[i] == aLocation)
        {
            cTimeDest.Add(c1ArrivalTime[i]);
        }
    }

    return cTimeDest;
}

public Trip Copy()
{
    Trip answer = new Trip();
    answer.aRoute = aRoute;
    answer.c1ArrivalTime = new List<int>();
    answer.c1ArrivalTime.AddRange(c1ArrivalTime);
    answer.cWaitTime = new List<int>();
    answer.cWaitTime.AddRange(cWaitTime);
    return answer;
}

public JoinedTrip Append1(Trip aTrip)
{
    JoinedTrip answer = new JoinedTrip();
    answer.cLocation.AddRange(this.cRouteLocation);
    answer.c1ArrivalTime.AddRange(this.c1ArrivalTime);
    answer.cWaitTime.AddRange(this.cWaitTime);
    if (this.cRouteLocation[this.cRouteLocation.Count - 1] == aTrip.cRouteLocation[0])
    {
        int nLocation = aTrip.cRouteLocation.Count;
        for (int i = 1; i < nLocation; i++)
        {// skip first location to avoid duplication.
            answer.cLocation.Add(aTrip.cRouteLocation[i]);
            answer.c1ArrivalTime.Add(aTrip.c1ArrivalTime[i]);
            answer.cWaitTime.Add(aTrip.cWaitTime[i]);
        }
    }
    else
    {
        answer.cLocation.AddRange(aTrip.cRouteLocation);
        answer.c1ArrivalTime.AddRange(aTrip.c1ArrivalTime);
        answer.cWaitTime.AddRange(aTrip.cWaitTime);
    }

    return answer;
}
public JoinedTrip Append(Trip aTrip)
{
    JoinedTrip answer = new JoinedTrip();
```

```
        answer.cLocation.AddRange(this.cRouteLocation);
        answer.c1ArrivalTime.AddRange(this.c1ArrivalTime);
        answer.cWaitTime.AddRange(this.cWaitTime);
        answer.cLocation.AddRange(aTrip.cRouteLocation);
        answer.c1ArrivalTime.AddRange(aTrip.c1ArrivalTime);
        answer.cWaitTime.AddRange(aTrip.cWaitTime);
        int index = this.c1ArrivalTime.Count-1;
        answer.cWaitTime[index] = aTrip.c1ArrivalTime[0] - this.c1ArrivalTime[index];

        return answer;
    }
}
```

TripPattern:

```
public class TripPattern : Trip
{
    public int repeatMinutes;
    public int firstRepeatTime;
    public int lastRepeatTime;
    public List<Trip> cTrip;
    public TripPattern()
        : base()
    {
    }

    public List<Trip> setcTrip()
    {
        cTrip = new List<Trip>();
        for (int time = firstRepeatTime; time <= lastRepeatTime; time = time + repeatMinutes)
        {
            Trip aTrip = new Trip();
            List<int> cTime = new List<int>();
            foreach (int aArrivalTime in c1ArrivalTime)
            {
                cTime.Add(time + aArrivalTime);
            }
            aTrip.c1ArrivalTime = cTime;
            aTrip.cWaitTime = this.cWaitTime;
            aTrip.aRoute = this.aRoute;
            aTrip.name = name;
            cTrip.Add(aTrip);
        }
        return cTrip;
    }

    public List<Trip> yesterdaycTrip()
    {
        cTrip = new List<Trip>();
        for (int time = firstRepeatTime; time <= lastRepeatTime; time = time + repeatMinutes)
        {
            Trip aTrip = new Trip();
            List<int> cTime = new List<int>();
            foreach (int aArrivalTime in c1ArrivalTime) { cTime.Add(time + aArrivalTime); }
            aTrip.c1ArrivalTime = cTime;
            aTrip.cWaitTime = this.cWaitTime;
            aTrip.aRoute = this.aRoute;
            aTrip.name = name;
            cTrip.Add(aTrip);
        }
        return cTrip;
    }

}

public class JoinedTrip : Trip
{
    public List<AtomicCityBusScheduler.Location> cLocation;
    public JoinedTrip()
        : base()
    {
```

```
            cLocation = new List<AtomicCityBusScheduler.Location>();
        }

        public List<int> startandEndTime(AtomicCityBusScheduler.Location startDest, AtomicCityBusScheduler.Location
endDest, int aArrivalTime)
        {
            int endIndex = -1;
            for (int i = c1ArrivalTime.Count - 1; endIndex < 0; i--)
            {
                if ((cLocation[i] == endDest) & (c1ArrivalTime[i] < aArrivalTime))
                {
                    endIndex = i;
                }
            }
            int startIndex = -1;
            for (int i = endIndex - 1; (startIndex < 0) & (i >= 0); i--)
            {
                if (cLocation[i] == endDest)
                {
                    endIndex = i;
                }
                if ((cLocation[i] == startDest) & (c1ArrivalTime[i] < c1ArrivalTime[endIndex]))
                {
                    startIndex = i;
                }
            }
            int startTime = c1ArrivalTime[startIndex] + cWaitTime[startIndex];
            int endTime = c1ArrivalTime[endIndex];
            List<int> startandEndTime = new List<int>();
            startandEndTime.Add(startTime);
            startandEndTime.Add(endTime);
            return startandEndTime;
        }

    }

}
```

Location:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace AtomicCityBusScheduler
{
    public class Location
    {
        public string name;
        public string address;
        public double latitude;
        public double longitude;
        public List<Route> cRoute;
    }
}
```