CS 202 Iditarod Challenge 4

Kelby Hubbard

April 14, 2020

- Repository Link: `https://github.com/krhubbard2/CS202/tree/master/Iditarod4`
- Git Commits: `https://github.com/krhubbard2/CS202/commits`
- This homework took approximately 12 hours to complete.

1 Design

The design for this program was a struggle. I had to go back and rethink my layout a few times and really think about how I wanted the classes to be and communicate with each other. I focused on getting CityPath class written first and fully compatible then focused on integrating TspSolver functions appropriately.

2 Post Mortem

This homework took me quite a bit of time. I ran into a few segmentation faults which I really struggled with taking me hours of debugging to finally fix. The issue was in one of my classes I ended up calling an incorrect vector location causing a memory issue. Other than that I still found this homework assignment rather challenging integrating the proper functions and tying together all 4 classes properly. But the outcome was a great, debugged, and finished project.

3 Iditarod Challenge 4

3.1 Sample Output

Listing 1: Sample Program Output

```
************ S O L V E  G R E E D Y ************

Path traveled: 1072 1030 1073 987 812 888 1032 814
   1043 1042 1041 1040 855 1301 1300 654  ...
184 183 182 181 180 658 23 810 333 332 331 36 1072
Total distance: 313998

************* S O L V E  R A N D O M L Y *************

Path traveled: 932 153 221 854 917 803 752 62 220 762
   20 992 662 195 680 1072 915 332 696 457 135 1010
   636 354 1191 76 362 1247 1295 847 666 519 656 815
   947 167 440 763 214 642 756
...
637 1121 154 749 248 335 771 809 589 1034 952 995 530
   65 781 372 857 442 48 1182 310 1096 294 850 1062
   394 517 932
Total distance: 9.22863e+06

*********** S O L V E  M Y W A Y ***************

Path traveled: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
   16 17 18 19 20 21 22 23 24 25 26 27 28
...
1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290
    1291 1292 1293 1294 1295 1296 1297 0
Total distance: 3225480
```

3.2 Git Commit Messages

Date	Message
2020-04-10	Started Iditarod4
2020-04-10	Start TspSovler and CityPath classes
2020-04-10	Write declaration of member fucntions.
2020-04-12	Write CityPath Class
2020-04-12	Write solveGreedy(in progress)
2020-04-12	SolveGreedy: Pick a random starting city – add it to CityPath
2020-04-12	Write member function SolveGreedy
2020-04-12	Cleaned up code
2020-04-12	Write member function SolveRandomly
2020-04-12	Write member function SolveMyWay
2020-04-12	Debugging finished.
2020-04-12	Final notes.

3.3 Source Code

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4

#include <iostream>
using std::cout;
using std::endl;
#include <fstream>
using std::ifstream;
#include <string>
using std::string;
#include <sstream>
using std::istringstream;
#include "citylist.hpp"
#include "citynode.hpp"
#include "citypath.hpp"
#include "tspsolver.hpp"


void readTSP(string fileName, CityNode& node, CityList& city)
{
  ifstream ifile(fileName);
  //Throw error if it can't open file
  if (!ifile)
  {
    cout << "Couldn't open file." << endl;
  }
  else
  {
```

```
31      city.setFileName(fileName);
32      string line;
33      bool loop = true;
34
35      while (loop)
36      {
37        //If reading file hits an error or EOF
38        if (!ifile)
39        {
40          if (ifile.eof())
41          {
42            loop = false;
43          }
44          else
45          {
46            loop = true;
47          }
48        }
49        //If file opens correctly
50        else
51        {
52          getline(ifile, line);
53          string nodeStart = "NODE_COORD_SECTION";
54
55          //Start of node listings
56          if (line == nodeStart)
57          {
58            bool loop1 = true;
59            while(loop1)
60            {
61              //If reading file hits EOF
62              if (line == "EOF")
63              {
64                loop1 = false;
65                loop = false;
66              }
67              else
68              {
69                loop1 = true;
70              }
71              getline(ifile, line);
72
73              //Ensure line is an int (node / info)
74              istringstream iss(line);
75              int val;
76              iss >> val;
77              if(iss)
78              {
79                istringstream iss1(line);
80                //Grab each section of string
81                for (int i = 0; i < 3; i++)
82                {
83                  double val1;
84
85                  iss1 >> val1;
86                  //Node number
87                  if (i == 0)
88                  {
89                    node.setNodeNumber(val1);
90                  }
91                  //Latitude
```

```
                else if (i == 1)
                {
                  node.setLatitudeY(val1);
                }
                //Longitude
                else if (i == 2)
                {
                  node.setLongitudeX(val1);
                }
              }
              city.setCityNode(node);
            }
          }
        }
      }
    }
  }
}
int main()
{
  CityNode node0(0, 0, 0);

  //Make CityList for BRD14051
  CityList brd;
  readTSP("brd14051.tsp", node0, brd);

  //Make CityList for FL3795
  CityList fl;
  readTSP("fl3795.tsp", node0, fl);

  //Make CityList for FNL4461
  CityList fnl;
  readTSP("fnl4461.tsp", node0, fnl);

  //Make CityList for RL1304
  CityList rl;
  readTSP("rl1304.tsp", node0, rl);

  //Make CityList for U2152
  CityList u;
  readTSP("u2152.tsp", node0, u);

  TspSolver testSolve;
  cout << "*****************************************************************\n"
       << "**************** S O L V E  G R E E D Y *********************\n"
       << "*****************************************************************\n";
  testSolve.solveGreedy(rl);
  cout << "*****************************************************************\n"
       << "**************** S O L V E  R A N D O M L Y *********************\n"
       << "*****************************************************************\n";
  testSolve.solveRandomly(rl);
  cout << "*****************************************************************\n"
       << "**************** S O L V E  M Y W A Y ***************************\n"
       << "*****************************************************************\n";
  testSolve.solveMyWay(rl);




  return 0;
}
```

3.4 CityNode Header

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4
#ifndef CITYNODE_HPP_
#define CITYNODE_HPP_

#include <string>
using std::string;

class CityNode
{
public:
  //Default Constructor
  CityNode();

  /*This constructor should typically be used. Adds node number,
    the lat, and long of node to the node.  */
  CityNode(unsigned int node, double latY, double lonX);

  //Sets _nodeNumber
  void setNodeNumber(unsigned int node);
  //Sets _latitude
  void setLatitudeY(double lat);
  //Sets _longitude
  void setLongitudeX(double lon);

  //returns _nodeNumber
  unsigned int getNodeNumber();
  //returns _latitude
  double getLatitudeY();
  //returns _longitude
  double getLongitudeX();

private:
  unsigned int _nodeNumber;
  double _latitude;
  double _longitude;
  double _graphX;
  double _graphY;
};




#endif
```

3.5 CityNode Source

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4
#ifndef CITYNODE_HPP_
#define CITYNODE_HPP_

```

```
#include <string>
using std::string;

class CityNode
{
public:
  //Default Constructor
  CityNode();

  /*This constructor should typically be used. Adds node number,
    the lat, and long of node to the node.  */
  CityNode(unsigned int node, double latY, double lonX);

  //Sets _nodeNumber
  void setNodeNumber(unsigned int node);
  //Sets _latitude
  void setLatitudeY(double lat);
  //Sets _longitude
  void setLongitudeX(double lon);

  //returns _nodeNumber
  unsigned int getNodeNumber();
  //returns _latitude
  double getLatitudeY();
  //returns _longitude
  double getLongitudeX();

private:
  unsigned int _nodeNumber;
  double _latitude;
  double _longitude;
  double _graphX;
  double _graphY;
};




#endif
```

3.6 CityList Header

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4
#ifndef CITYLIST_HPP_
#define CITYLIST_HPP_

#include <iostream>
using std::cout;
using std::endl;
#include <string>
using std::string;
#include <vector>
using std::vector;
#include "citynode.hpp"
#include <cmath>

```

```
18 class CityList
19 {
20 public:
21   //Default Constructor
22   CityList();
23
24   //Constructor automatically adding node to _cityList vector
25   CityList(CityNode node);
26
27   //Adds node to _cityList vector
28   void setCityNode(CityNode node);
29
30   //Prints whole _cityList vector
31   void printAllCityNodes();
32
33   /*Prints specific vector position of _cityList depending
34     on node you want printed*/
35   void printSpecCityNode(unsigned int node);
36
37   //Sets file/city name
38   void setFileName(string name);
39
40   //returns _fileName
41   string getFileName();
42
43   //Returns Euclidean distance between two cities (node 1 & node 2).
44   double distance(int first, int second);
45
46   //Returns CityNode
47   CityNode getCityNode(int node);
48
49   //returns  _latitude from CityNode
50   double getCityLat(int node);
51
52   //returns _longitude from CityNode
53   double getCityLon(int node);
54
55   //Returns _cityList vector size (For use in TspSolver)
56   int cityListSize();
57
58   //Returns _cityList [n] value as an int
59   int listVectorSpecific(int n);
60
61   int getCityNodeInt(int node);
62
63   //Removes node from _cityList (For use in TspSolver)
64   void removeCityList(int node);
65
66 private:
67   vector<CityNode> _cityList;
68   string _fileName;
69   CityNode node;
70 };
71
72
73
74
75
76
77
78
79
80
81
82
83 #endif
```

3.7 CityList Source

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4
#include "citylist.hpp"

CityList::CityList() {}

CityList::CityList(CityNode node)
{
  _cityList.push_back(node);
}

void CityList::setCityNode(CityNode node)
{
  _cityList.push_back(node);
}

CityNode CityList::getCityNode(int node)
{
  return _cityList[node];
}

int CityList::getCityNodeInt(int node)
{
  return _cityList[node].getNodeNumber();
}

double CityList::getCityLat(int node)
{
  return _cityList[node].getLatitudeY();
}

double CityList::getCityLon(int node)
{
  return _cityList[node].getLongitudeX();
}

void CityList::printAllCityNodes()
{
  for (auto a : _cityList)
  {
    cout << "City/File: " << getFileName() << " Node: " << a.getNodeNumber()
         << " Lat: " << a.getLatitudeY() << " Lon: " << a.getLongitudeX()
         << endl;
  }
}

void CityList::printSpecCityNode(unsigned int node)
{
  /*Function assumes you typed in the node you want printed NOT the vector
    position. Hence node-1 to take into account vector position [0]*/
  cout << "City/File: " << getFileName() << " Node: "
       << _cityList[node-1].getNodeNumber() << " Lat: "
       << _cityList[node-1].getLatitudeY() << " Lon: "
       << _cityList[node-1].getLongitudeX() << endl;
}

void CityList::setFileName(string name)
{
```

```
  _fileName = name;
}
string CityList::getFileName()
{
  return _fileName;
}
  //Returns Euclidean distance between two cities (node 1 & node 2).
double CityList::distance(int first, int second)
{
  //x = long y = lat
  double d, x1, x2, y1, y2;
  x1 = getCityLon(first);
  x2 = getCityLon(second);
  y1 = getCityLat(first);
  y2 = getCityLat(second);

  d = sqrt(pow((x2-x1),2) + pow((y2-y1),2));

  return d;

}
int CityList::cityListSize()
{
  return _cityList.size();
}
int CityList::listVectorSpecific(int n)
{
  return _cityList[n].getNodeNumber();
}
void CityList::removeCityList(int node)
{
  _cityList.erase(_cityList.begin() + node);
}
```

3.8 CityPath Header

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4

#ifndef CITYPATH_HPP_
#define CITYPATH_HPP_
#include <iostream>
#include <vector>
using std::vector;
#include "citynode.hpp"

class CityPath
{
public:
  CityPath(const int& s);

  CityPath() : _size(0) {};

  int cityPathSize();
```

```
21
22    CityPath(CityPath &copy) {_connections = copy._connections;};
23
24    int getPath(const int& n) const;
25
26    void addPath(const int& n);
27
28    void deletePath(const int& n);
29
30    void deleteAllPaths();
31
32    int size() const { return _connections.size(); }
33
34    vector<int> get_connections();
35
36    void setConnections(vector<int>& vec);
37
38  private:
39    //Stores optimal connections from city to city (vector<CityNode> _cityList)
40    vector<int> _connections;
41
42    int _size;
43  };
44
45
46
47
48
49
50
51  #endif
```

3.9 CityPath Source

```
1   // Kelby Hubbard
2   // CS202
3   // April 10, 2020
4   // Iditarod Challenge #4
5
6   #include "citypath.hpp"
7
8   CityPath::CityPath(const int& s) : _size(s) {}
9
10  int CityPath::cityPathSize()
11  {
12    return _connections.size();
13  }
14
15  void CityPath::addPath(const int& n)
16  {
17    _connections.push_back(n);
18  }
19
20  void CityPath::deletePath(const int& n)
21  {
22    if (_connections.size() == 1) {
23      _connections.erase(_connections.begin());
24      return;
25    }
26    _connections.erase(_connections.begin() + n);
27  }
28
29  void CityPath::deleteAllPaths()
30  {
```

```
  _connections.clear();
}

int CityPath::getPath(const int& n) const
{
  return _connections[n];
}

vector<int> CityPath::get_connections()
{
  return _connections;
}

void CityPath::setConnections(vector<int>& vec)
{
  for (auto a : vec)
  {
    _connections.push_back(a);
  }
}
```

3.10 TspSolver Header

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4

#ifndef TSPSOLVER_HPP_
#define TSPSOLVER_HPP_

#include "citylist.hpp"
#include "citypath.hpp"
#include "citynode.hpp"
#include <random>


class TspSolver
{
public:
  void solveRandomly(CityList& list);
  void solveGreedy(CityList& list);
  void solveMyWay(CityList& list);
  unsigned int getRandInt(int low, int high);

private:


};









#endif
```

3.11 TspSolver Source

```
// Kelby Hubbard
// CS202
// April 10, 2020
// Iditarod Challenge #4

#include "tspsolver.hpp"

unsigned int TspSolver::getRandInt(int low, int high)
{
  std::random_device rd;
  std::mt19937 gen1(rd());
  std::uniform_int_distribution<int> dist(low,high);
  return dist(gen1);
}

void TspSolver::solveRandomly(CityList& list)
{
  CityPath marked;
  CityPath unmarked;
  vector<int> temp;
  vector<int> finalVec;

  //Starting distance
  double dist = 0;

  //Best Distance
  double bestDist = 1e12;

  //M times repeated -- Change m to whatever
  int m = 10;

  for (int i = 0; i < m; i++)
  {
    temp.clear();
    dist = 0;
    unmarked.deleteAllPaths();
    //Fill unmarked path
    for (int i = 0; i < list.cityListSize(); i++)
    {
      unmarked.addPath(i);
    }
    temp.clear();
    //Set starting city
    int startNode = getRandInt(0, list.cityListSize() - 1);
    //Add starting city to temp
    temp.push_back(startNode);
    //Delete starting city from unmarked
    unmarked.deletePath(startNode);

    //While cities unmarked remain
    while (unmarked.size() != 0)
    {
      //Randomly connect cities
      int startNode = getRandInt(0, unmarked.size() - 1);
      temp.push_back(unmarked.getPath(startNode));
      unmarked.deletePath(startNode);
    }

    //Starting city is also ending city
    temp.push_back(temp[0]);
```

```
61
62      //Find distance traveled
63      for (auto i = 0; i < temp.size() - 1; i++)
64      {
65        dist += list.distance(temp[i], temp[i + 1]);
66      }
67
68      if (dist < bestDist)
69      {
70        bestDist = dist;
71        finalVec = temp;
72      }
73    }
74
75    marked.setConnections(finalVec);
76    dist = 0;
77    cout << "Path traveled: ";
78    for (int i = 0; i < marked.size() - 1; i++)
79    {
80      dist += list.distance(marked.getPath(i), marked.getPath(i+1));
81      cout << marked.getPath(i) << " ";
82    }
83    cout << marked.getPath(marked.size() - 1) << endl;
84    cout << "Total distance: " << dist << endl;
85  }
86
87  void TspSolver::solveGreedy(CityList& list)
88  {
89    CityPath marked;
90    CityPath unmarked;
91
92    //Fill unmarked path
93    for (int i = 0; i < list.cityListSize(); i++)
94    {
95      unmarked.addPath(i);
96    }
97
98    //Starting distance
99    double dist = 0;
100
101   //Set starting city
102   int startNode = getRandInt(0, list.cityListSize() - 1);
103   //Add starting city to marked
104   marked.addPath(startNode);
105   //Delete starting city from unmarked
106   unmarked.deletePath(startNode);
107
108   int smallest;
109   int del;
110   // Loop for all nodes
111   for (int i = 0; i < list.cityListSize(); i++)
112   {
113     dist = 1e12;
114
115     //If last city
116     if (unmarked.size() == 1)
117     {
118       marked.addPath(unmarked.getPath(0));
119       unmarked.deletePath(0);
120       break;
121     }
122
123     //Find closest city comparing all cities remaining (unmarked)
```

```
124       for(auto z = 0; z < unmarked.cityPathSize(); z++)
125       {
126         if (dist > list.distance(marked.getPath(i), unmarked.getPath(z)))
127         {
128           dist = list.distance(marked.getPath(i), unmarked.getPath(z));
129
130           //Smallest city found
131           smallest = unmarked.getPath(z);
132           //Delete the found city
133           del = z;
134         }
135       }
136
137       marked.addPath(smallest);
138       unmarked.deletePath(del);
139
140     }
141
142     marked.addPath(marked.getPath(0));
143
144     dist = 0;
145     cout << "Path traveled: ";
146     for (int i = 0; i < marked.size() - 1; i++)
147     {
148       dist += list.distance(marked.getPath(i), marked.getPath(i+1));
149       cout << marked.getPath(i) << " ";
150     }
151     cout << marked.getPath(marked.size() - 1) << endl;
152     cout << "Total distance: " << dist << endl;
153 }
154
155 void TspSolver::solveMyWay(CityList& list)
156 {
157     // Connect cities in node order (ending with starting node)
158     CityPath marked;
159
160     for (int i = 0; i < list.cityListSize(); i++)
161     {
162       marked.addPath(i);
163     }
164
165     int dist = 0;
166     cout << "Path traveled: ";
167     for (int i = 0; i < marked.size() - 1; i++)
168     {
169       dist += list.distance(marked.getPath(i), marked.getPath(i+1));
170       cout << marked.getPath(i) << " ";
171     }
172     cout << marked.getPath(marked.size() - 1) << endl;
173     cout << "Total distance: " << dist << endl;
174
175 }
```