CS 202 Iditarod Challenge 6

Kelby Hubbard

April 21, 2020

- Repository Link: `https://github.com/krhubbard2/CS202/tree/master/Iditarod6`
- Git Commits: `https://github.com/krhubbard2/CS202/commits`
- This homework took approximately 4 hours to complete.

1 Design

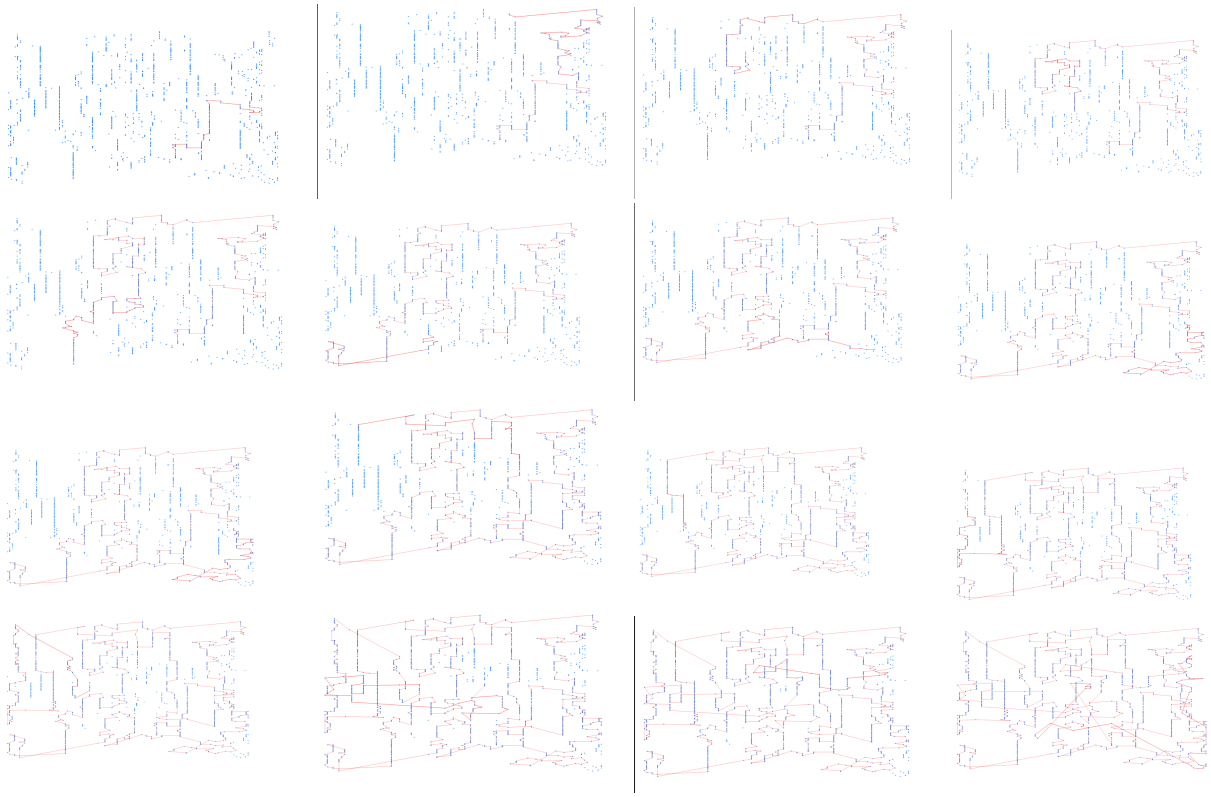
The design for this program was fairly easy to follow. Originally I was going to create 16 different SVG output files, but found it easier to rather just have one SVG output file which had 16 different SVGs inside it as SVG can handle these kind of operations.

2 Post Mortem

This took a little longer than expected trying to figure out how to keep all cities on each seperate SVG and keep the old path traveled, without adding in any extra lines. But once solved it went quite quickly.

3 Iditarod Challenge 6

3.1 RL1304 Greedy Output



3.2 Git Commit Messages

Date	Message
2020-04-21	Started Iditarod6
2020-04-21	Create 16 snapshots of SVG in progress
2020-04-21	Debug
2020-04-21	Screenshots

3.3 Source Code

```
// Kelby Hubbard
// CS202
// April 21, 2020
// Iditarod Challenge #6

#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"
#include "svg.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
  {
    city.setFileName(fileName);
    string line;
    bool loop = true;

    while (loop)
    {
      //If reading file hits an error or EOF
      if (!ifile)
      {
        if (ifile.eof())
        {
          loop = false;
        }
        else
        {
          loop = true;
        }
      }
      //If file opens correctly
      else
      {
        getline(ifile, line);
        string nodeStart = "NODE_COORD_SECTION";

        //Start of node listings
        if (line == nodeStart)
        {
          bool loop1 = true;
          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
92                  else if (i == 1)
93                  {
94                    node.setLatitudeY(val1);
95                  }
96                  //Longitude
97                  else if (i == 2)
98                  {
99                    node.setLongitudeX(val1);
100                 }
101               }
102               city.setCityNode(node);
103             }
104           }
105         }
106       }
107     }
108   }
109 }
110
111
112 int main(int argc, char** argv){
113
114   CityNode node0(0, 0, 0);
115   TspSolver solve;
116
117   //Make CityList for FL3795
118   cout << "CITY: RL1304\n";
119   CityList rl;
120   readTSP("rl1304.tsp", node0, rl);
121   CityPath svgrl;
```

```
  solve.solveGreedy(rl, svgrl);
  cout << "Generating Greedy SVG Output\n";
  svgGraph(rl, svgrl, "rl1304GreedyOutput.svg");


  return 0;
}
```

3.4 SVG Source Code

```
#include "svg.hpp"

void svgGraph(CityList &list, CityPath &path, string outputName){
  double minX = list.minX();
  double maxX = list.maxX();
  double minY = list.minY();
  double maxY = list.maxY();

  string line;
  vector<string> previous;
  vector<string> citydots;


  ofstream svgOut(outputName);
  if (svgOut.is_open())
  {

        line = "<html xmlns=\"http://www.w3.org/1999/xhtml\"\n";
        line += "xmlns:svg=\"http://www.w3.org/2000/svg\">\n";
        line += "<body>\n <svg:svg width=\"2250px\" height=\"2500px\">\n";
        svgOut << line;

        double x = 0;
        double y = 0;
        double oldX = 0;
        double oldY = 0;
        double startX = 0;
        double startY = 0;
        string dot = "";
        string lines = "";


      //Dots
      for (auto i = 0; i < path.size(); i++){
      oldX = x;
      oldY = y;

      if (i == 0){
        startX = x;
        startY = y;
      }


      dot = "<svg:circle cx=\"";
      x = 1980 - 1920 * (list.getCityLon(path.getPath(i)) - minX) / (maxX - minX);
      y = 2080 - 1080 * (list.getCityLat(path.getPath(i)) - minY) / (maxY - minY);

      //Cities
```

```
      dot += std::to_string(x);
      dot += "\" cy=\"";
      dot += std::to_string(y);
      dot += "\" r=\"";
      dot += '3';
      dot += "\" fill=\"#0078ff\"/>\n";

      svgOut << dot;

      citydots.push_back(dot);
    }

    for (auto n = 0; n < path.size(); n++){

      if(n != 0){
        oldX = x;
        oldY = y;
        x = 1980 - 1920 * (list.getCityLon(path.getPath(n)) - minX) / (maxX - minX);
        y = 2080 - 1080 * (list.getCityLat(path.getPath(n)) - minY) / (maxY - minY);
        dot = "<svg:line x1=\"";
        dot += std::to_string(oldX);
        dot += "\" y1=\"";
        dot += std::to_string(oldY);
        dot += "\" x2=\"";
        dot += std::to_string(x);
        dot += "\" y2=\"";
        dot += std::to_string(y);
        dot += "\" style=\"stroke:rgb(255,0,0);stroke-width:1\" />\n";
        previous.push_back(dot);

        if (n % 82 == 0)
        {
          dot = "</svg:svg>\n";
          dot += "<svg:svg width=\"2250px\" height=\"2500px\">\n";
          for (auto a = 0; a < citydots.size(); a++)
          {
            svgOut << citydots[a];
          }
          for (auto b = 0; b < previous.size(); b++){
            svgOut << previous[b];
          }
        }

        if (n % 1304 == 0){
          dot = "</svg:svg>\n";
          dot += "<svg:svg width=\"2250px\" height=\"2500px\">\n";
          for (auto a = 0; a < citydots.size(); a++)
          {
            svgOut << citydots[a];
          }
          for (auto b = 0; b < previous.size(); b++){
            svgOut << previous[b];
          }
        }
        svgOut << dot;
      }
    }


        //Connect last city to first
        dot = "<svg:line x1=\"";
        dot += std::to_string(x);
```

```
        dot += "\" y1=\"";
        dot += std::to_string(y);
        dot += "\" x2=\"";
        dot += std::to_string(startX);
        dot += "\" y2=\"";
        dot += std::to_string(startY);
        dot += "\" style=\"stroke:rgb(255,0,0);stroke-width:1\" />\n";

        dot += "</svg:svg>\n <svg:svg width=\"2250px\" height=\"2500px\">\n";
        dot += "</svg:svg>\n </body>\n </html>";
        svgOut << dot;
  }
}
```

3.5 SVG Header

```
// Kelby Hubbard
// CS202
// April 21, 2020
// Iditarod Challenge #5

#ifndef SVG_HPP_
#define SVG_HPP_

#include "citylist.hpp"
#include "citypath.hpp"
#include <fstream>
using std::ofstream;

void svgGraph(CityList &list, CityPath &path, string outputName);



#endif
```

3.6 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.  */
```

```
19   CityNode(unsigned int node, double latY, double lonX);
20
21   //Sets _nodeNumber
22   void setNodeNumber(unsigned int node);
23   //Sets _latitude
24   void setLatitudeY(double lat);
25   //Sets _longitude
26   void setLongitudeX(double lon);
27
28   //returns _nodeNumber
29   unsigned int getNodeNumber();
30   //returns _latitude
31   double getLatitudeY();
32   //returns _longitude
33   double getLongitudeX();
34
35 private:
36   unsigned int _nodeNumber;
37   double _latitude;
38   double _longitude;
39   double _graphX;
40   double _graphY;
41 };
42
43
44
45
46
47 #endif
```

3.7 CityNode Source

```
1  // Kelby Hubbard
2  // CS202
3  // April 10, 2020
4  // Iditarod Challenge #4
5  #ifndef CITYNODE_HPP_
6  #define CITYNODE_HPP_
7
8  #include <string>
9  using std::string;
10
11 class CityNode
12 {
13 public:
14   //Default Constructor
15   CityNode();
16
17   /*This constructor should typically be used. Adds node number,
18     the lat, and long of node to the node.  */
19   CityNode(unsigned int node, double latY, double lonX);
20
21   //Sets _nodeNumber
22   void setNodeNumber(unsigned int node);
23   //Sets _latitude
24   void setLatitudeY(double lat);
25   //Sets _longitude
26   void setLongitudeX(double lon);
27
28   //returns _nodeNumber
29   unsigned int getNodeNumber();
```

```
  //returns _latitude
  double getLatitudeY();
  //returns _longitude
  double getLongitudeX();

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



#endif
```

3.8 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>

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

  //Constructor automatically adding node to _cityList vector
  CityList(CityNode node);

  //Adds node to _cityList vector
  void setCityNode(CityNode node);

  //Prints whole _cityList vector
  void printAllCityNodes();

  /*Prints specific vector position of _cityList depending
    on node you want printed*/
  void printSpecCityNode(unsigned int node);

  //Sets file/city name
  void setFileName(string name);

  //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
```

3.9 CityList Source

```
1  // Kelby Hubbard
2  // CS202
3  // April 10, 2020
4  // Iditarod Challenge #4
5  #include "citylist.hpp"
6
7  CityList::CityList() {}
8
9  CityList::CityList(CityNode node)
10 {
11   _cityList.push_back(node);
12 }
13
14 void CityList::setCityNode(CityNode node)
15 {
16   _cityList.push_back(node);
17 }
```

```
18
19 CityNode CityList::getCityNode(int node)
20 {
21   return _cityList[node];
22 }
23
24 int CityList::getCityNodeInt(int node)
25 {
26   return _cityList[node].getNodeNumber();
27 }
28
29 double CityList::getCityLat(int node)
30 {
31   return _cityList[node].getLatitudeY();
32 }
33
34 double CityList::getCityLon(int node)
35 {
36   return _cityList[node].getLongitudeX();
37 }
38
39 void CityList::printAllCityNodes()
40 {
41   for (auto a : _cityList)
42   {
43     cout << "City/File: " << getFileName() << " Node: " << a.getNodeNumber()
44          << " Lat: " << a.getLatitudeY() << " Lon: " << a.getLongitudeX()
45          << endl;
46   }
47 }
48
49 void CityList::printSpecCityNode(unsigned int node)
50 {
51   /*Function assumes you typed in the node you want printed NOT the vector
52     position. Hence node-1 to take into account vector position [0]*/
53   cout << "City/File: " << getFileName() << " Node: "
54        << _cityList[node-1].getNodeNumber() << " Lat: "
55        << _cityList[node-1].getLatitudeY() << " Lon: "
56        << _cityList[node-1].getLongitudeX() << endl;
57 }
58
59 void CityList::setFileName(string name)
60 {
61   _fileName = name;
62 }
63
64 string CityList::getFileName()
65 {
66   return _fileName;
67 }
68
69   //Returns Euclidean distance between two cities (node 1 & node 2).
70 double CityList::distance(int first, int second)
71 {
72   //x = long y = lat
73   double d, x1, x2, y1, y2;
74   x1 = getCityLon(first);
75   x2 = getCityLon(second);
76   y1 = getCityLat(first);
77   y2 = getCityLat(second);
78
79   d = sqrt(pow((x2-x1),2) + pow((y2-y1),2));
80
```

```
  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);
}
```

3.10 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();

  CityPath(CityPath &copy) {_connections = copy._connections;};

  int getPath(const int& n) const;

  void addPath(const int& n);

  void deletePath(const int& n);

  void deleteAllPaths();

  int size() const { return _connections.size(); }

  vector<int> get_connections();

  void setConnections(vector<int>& vec);

private:
  //Stores optimal connections from city to city (vector<CityNode> _cityList)
  vector<int> _connections;

  int _size;
};
```

```
#endif
```

3.11 CityPath Source

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

#include "citypath.hpp"

CityPath::CityPath(const int& s) : _size(s) {}

int CityPath::cityPathSize()
{
  return _connections.size();
}

void CityPath::addPath(const int& n)
{
  _connections.push_back(n);
}

void CityPath::deletePath(const int& n)
{
  if (_connections.size() == 1) {
    _connections.erase(_connections.begin());
    return;
  }
  _connections.erase(_connections.begin() + n);
}

void CityPath::deleteAllPaths()
{
  _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);
  }
}
```

3.12 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
```

3.13 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);
}
```

```
15
16 void TspSolver::solveRandomly(CityList& list)
17 {
18   CityPath marked;
19   CityPath unmarked;
20   vector<int> temp;
21   vector<int> finalVec;
22
23   //Starting distance
24   double dist = 0;
25
26   //Best Distance
27   double bestDist = 1e12;
28
29   //M times repeated -- Change m to whatever
30   int m = 10;
31
32   for (int i = 0; i < m; i++)
33   {
34     temp.clear();
35     dist = 0;
36     unmarked.deleteAllPaths();
37     //Fill unmarked path
38     for (int i = 0; i < list.cityListSize(); i++)
39     {
40       unmarked.addPath(i);
41     }
42     temp.clear();
43     //Set starting city
44     int startNode = getRandInt(0, list.cityListSize() - 1);
45     //Add starting city to temp
46     temp.push_back(startNode);
47     //Delete starting city from unmarked
48     unmarked.deletePath(startNode);
49
50     //While cities unmarked remain
51     while (unmarked.size() != 0)
52     {
53       //Randomly connect cities
54       int startNode = getRandInt(0, unmarked.size() - 1);
55       temp.push_back(unmarked.getPath(startNode));
56       unmarked.deletePath(startNode);
57     }
58
59     //Starting city is also ending city
60     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 }
```