CPSC 335
Spring 2013
Project #4 — minimum spanning trees

Introduction

In this project you will implement an efficient minimum spanning tree algorithm. This involves implementing a graph representation and using an appropriate data structure. The input graph will represent the layout of the Disneyland resort.

Objective

The goal is to implement one of the three minimum spanning tree (MST) algorithms we've covered. Those three algorithms are

1. Prim's algorithm, using only arrays and/or lists;
2. Prim's algorithm, accelerated by a *heap* (a.k.a. *priority queue*); or
3. Kruskal's algorithm, using the *union-find* data structure.

For full credit, you must implement one of the $O(m \log n)$-time MST algorithms (2 or 3). If you cannot do this, you may instead implement algorithm 1, which is simpler but slower, running in $O(mn)$ time. If you go this route you will lose some credit, but still be eligible for credit on the other parts of the project.

Our graph has $n = 84$ vertices and $m = \frac{n(n-1)}{2} = 3,486$ edges. So all these algorithms should finish very quickly if they are implemented properly.

Data structures

The fast version of Prim's algorithm depends on an efficient priority queue to achieve its $O(m \log n)$ running time.

Similarly, Kruskal's algorithm depends on a union-find structure with $O(\log n)$-time operations (or better) to run in $O(m \log n)$ time. It also depends on an $O(n \log n)$ time sorting algorithm to presort the edges.

So if you pick either of these algorithms, your code needs to use the corresponding data structure. Data structures are not the focus of this course or project, so you may reuse an existing implementation of them. The C++ STL has a priority queue and optimal sorting algorithm, but it does not have a union-find structure. Alternatively you may be able to find a priority queue or union-find implementation on the Internet. As always, if you incorporate someone else's code, make sure to cite them appropriately.

You must write the code for Prim's or Kruskal's algorithm yourself.

The input

The input to your algorithm will be a graph whose vertices correspond to points of interest at Disneyland, and whose edges are weighted according to an estimate of the walking distance between them. The vertices are the 84 locations marked on the official Disneyland map available at

`http://disneyland.disney.go.com/media/dlr_nextgen/SiteCatalog/PDF/DisneylandParkMap_2011060311.pdf`

and also provided in TITANium. You are also provided with a text data file `disneyland.txt` that stores a graph in the following format:

```
[number of points]
[id 1] [x-coordinate 1] [y-coordinate 1] [name 1]
[id 2] [x-coordinate 2] [y-coordinate 2] [name 2]
...
[id n] [x-coordinate n] [y-coordinate n] [name n]
```

All the fields, except for names, are positive integers. Names are contiguous strings of letters. The coordinates correspond to pixels in my PDF viewer; they don't relate directly to real-world units of measure.

The actual lines of the file corresponding to the template above are

```
84
1 957 685 RailRoadMainStreet
2 1009 593 MainStreetCinema
...
84 1222 422 TomorrowlandTerrace
```

The fields are separated by whitespace, which should make them easy to parse. In particular, the fields can be parsed conveniently using the C++ stream extraction operator >>.

You will need to write code to load the file and convert it into a graph. Each point should correspond to a vertex in your graph. The graph should be complete, meaning that there is an edge between every pair of distinct vertices. You can define the weight of each edge by the Euclidean distance metric. If an edge connects the coordinates (x_1, y_1) and (x_2, y_2), its weight should be

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Graph representation

Your MST code will need to interact with a graph representation: either an adjacency list or adjacency matrix. Our graph is complete, so either representation should be reasonably efficient. You must write your own graph representation code as part of the project.

Code to write

Your program should

1. Load the Disneyland graph.

2. Compute an MST for that graph, and measure how long that takes.
3. Print out
   (a) The tree's total weight.
   (b) The list of edges in the tree.
   (c) The amount of time that the MST algorithm took.

Sample output

```
Loaded 84 sites
Kruskal's algorithm took 0.01 seconds
Tree edges:
  (2)[MainStreetCinema] <-> (10)[CarnationCafe] weight=43.3244
  (2)[MainStreetCinema] <-> (12)[MainStreetConeShop] weight=58.8303
  (3)[FireEngine] <-> (5)[HorselessCarriage] weight=19.2094
  (4)[HorseDrawnStreetcars] <-> (1)[RailRoadMainStreet] weight=39.4462
  (4)[HorseDrawnStreetcars] <-> (2)[MainStreetCinema] weight=74.2159
  (4)[HorseDrawnStreetcars] <-> (6)[Omnibus] weight=47.0956
  (5)[HorselessCarriage] <-> (1)[RailRoadMainStreet] weight=16.9706
  (7)[DisneyGallery] <-> (6)[Omnibus] weight=24.0832
  (7)[DisneyGallery] <-> (8)[GreatMomentsWithMrLincoln] weight=21.0238
  (9)[BlueRibbonBakery] <-> (11)[GibsonGirlIceCreamParlor] weight=18.1108
  (10)[CarnationCafe] <-> (9)[BlueRibbonBakery] weight=17.0294
  (12)[MainStreetConeShop] <-> (13)[PlazaInn] weight=67.0298
  (13)[PlazaInn] <-> (71)[AstroOrbitor] weight=46.0435
  (14)[MainStreetRefreshmentCorner] <-> (11)[GibsonGirlIceCreamParlor] weight=20
  (15)[RailRoadNewOrleansSquare] <-> (19)[FrenchMarketRestaurant] weight=70.0928
  (16)[HauntedMansion] <-> (15)[RailRoadNewOrleansSquare] weight=67.6018
  (17)[PiratesOfTheCaribbean] <-> (20)[CafeOrleans] weight=35.6931
  (18)[BlueBayouRestaurant] <-> (21)[RoyalStreetVeranda] weight=39.0512
  (18)[BlueBayouRestaurant] <-> (39)[IndianaJones] weight=77.1038
  (20)[CafeOrleans] <-> (19)[FrenchMarketRestaurant] weight=42.0476
  (21)[RoyalStreetVeranda] <-> (17)[PiratesOfTheCaribbean] weight=22.0907
  (22)[BigThunderMountainRailroad] <-> (26)[SailingShipColumbia] weight=33.2415
  (23)[PiratesLairOnTomSawyerIsland] <-> (34)[DavyRockettsExplorerCanoes] weight=65.0077
  (24)[FrontierlandShootinExposition] <-> (30)[RanchoDelZocaloRestaurant] weight=34.8281
  (26)[SailingShipColumbia] <-> (25)[MarkTwainRiverboat] weight=19.2354
  (27)[BigThunderRanch] <-> (22)[BigThunderMountainRailroad] weight=89.0898
  (28)[GoldenHorseshoeStage] <-> (25)[MarkTwainRiverboat] weight=58.1378
  (28)[GoldenHorseshoeStage] <-> (29)[StageDoorCafe] weight=27.0185
  (28)[GoldenHorseshoeStage] <-> (30)[RanchoDelZocaloRestaurant] weight=67.624
  (29)[StageDoorCafe] <-> (42)[BengalBarbecue] weight=34.2053
  (31)[RiverBelleTerrace] <-> (42)[BengalBarbecue] weight=31.3209
  (32)[BigThunderRanchBBQ] <-> (27)[BigThunderRanch] weight=54.6717
  (33)[SplashMountain] <-> (35)[WinnieThePooh] weight=61.8547
  (33)[SplashMountain] <-> (37)[HungryBearRestaurant] weight=54.1295
  (34)[DavyRockettsExplorerCanoes] <-> (36)[HarbourGalley] weight=55.109
  (36)[HarbourGalley] <-> (16)[HauntedMansion] weight=32.28
  (37)[HungryBearRestaurant] <-> (34)[DavyRockettsExplorerCanoes] weight=48.0104
  (38)[EnchantedTikiRoom] <-> (14)[MainStreetRefreshmentCorner] weight=77.7946
  (38)[EnchantedTikiRoom] <-> (43)[TikiJuiceBar] weight=19.105
  (40)[JungleCruise] <-> (42)[BengalBarbecue] weight=42.5206
  (41)[TarzansTreehouse] <-> (21)[RoyalStreetVeranda] weight=64.0078
  (41)[TarzansTreehouse] <-> (31)[RiverBelleTerrace] weight=29.1548
  (43)[TikiJuiceBar] <-> (40)[JungleCruise] weight=49.6488
```

```
  (44)[ChipNDaleTreehouse] <-> (49)[MickeysHouse] weight=37.1214
  (45)[RailroadToontown] <-> (70)[TroubadourTavern] weight=24.7588
  (46)[DonaldsBoat] <-> (48)[GoofysPlayhouse] weight=22.0227
  (46)[DonaldsBoat] <-> (57)[PrincessFantasyFaire] weight=36
  (47)[GadgetsGoCoaster] <-> (44)[ChipNDaleTreehouse] weight=25.0799
  (48)[GoofysPlayhouse] <-> (52)[ToontownDining] weight=36.7151
  (49)[MickeysHouse] <-> (50)[MinniesHouse] weight=25.9615
  (50)[MinniesHouse] <-> (52)[ToontownDining] weight=44.1022
  (51)[RogerRabbitsCarToonSpin] <-> (45)[RailroadToontown] weight=64.1405
  (53)[AliceInWonderland] <-> (68)[PixieHollow] weight=68.4471
  (54)[BibbidiBobbidiBoutique] <-> (64)[PinocchiosDaringJourney] weight=2
  (55)[CaseyJrCircusTrain] <-> (59)[KingArthurCarrousel] weight=24.4131
  (55)[CaseyJrCircusTrain] <-> (69)[VillageHausRestaurant] weight=32.7567
  (56)[Dumbo] <-> (62)[MrToadsWildRide] weight=26.9072
  (56)[Dumbo] <-> (70)[TroubadourTavern] weight=62
  (57)[PrincessFantasyFaire] <-> (70)[TroubadourTavern] weight=19.0263
  (60)[MadTeaParty] <-> (53)[AliceInWonderland] weight=29
  (60)[MadTeaParty] <-> (62)[MrToadsWildRide] weight=43.0116
  (60)[MadTeaParty] <-> (67)[StoryBookLandCanalBoats] weight=42.5206
  (61)[MatterhornBobsleds] <-> (58)[itsasmallworld] weight=66.6108
  (61)[MatterhornBobsleds] <-> (67)[StoryBookLandCanalBoats] weight=55.0091
  (62)[MrToadsWildRide] <-> (63)[PeterPansFlight] weight=22.1359
  (63)[PeterPansFlight] <-> (65)[SleepingBeauty] weight=55.0364
  (65)[SleepingBeauty] <-> (59)[KingArthurCarrousel] weight=29
  (66)[SnowWhite] <-> (54)[BibbidiBobbidiBoutique] weight=23.3238
  (69)[VillageHausRestaurant] <-> (64)[PinocchiosDaringJourney] weight=18.0278
  (71)[AstroOrbitor] <-> (68)[PixieHollow] weight=51.4782
  (72)[Autopia] <-> (74)[Monorail] weight=38.0132
  (72)[Autopia] <-> (75)[RailRoadTomorrowland] weight=92.0054
  (72)[Autopia] <-> (78)[Innoventions] weight=52.4309
  (73)[AstroBlasters] <-> (68)[PixieHollow] weight=45.2769
  (73)[AstroBlasters] <-> (81)[StarTours] weight=52.8015
  (76)[FindingNemoSubmarines] <-> (74)[Monorail] weight=21.0238
  (76)[FindingNemoSubmarines] <-> (84)[TomorrowlandTerrace] weight=64.622
  (77)[CaptainEO] <-> (82)[ReddRockettsPizzaPort] weight=29.1548
  (78)[Innoventions] <-> (83)[SpiritOfRefreshment] weight=40.0125
  (79)[SpaceMountain] <-> (77)[CaptainEO] weight=29.0689
  (80)[Starcade] <-> (77)[CaptainEO] weight=34.1321
  (82)[ReddRockettsPizzaPort] <-> (83)[SpiritOfRefreshment] weight=22.561
  (84)[TomorrowlandTerrace] <-> (81)[StarTours] weight=81.0555
total weight = 3502.85
```

Deliverables

Produce a written project report. Your report should include the following:

1. Your name(s) and an indication that the submission is for project 4.
2. Your source code.
3. Output from your program. As stated earlier, your program should print the MST edges, MST weight, and running time of the MST algorithm.

4. Answers to the following questions:
   (a) Which algorithm did you implement? Why?
   (b) Did you implement a fast $O(m \log n)$-time algorithm or the simpler $O(mn)$-time algorithm? If you implemented the faster version, how difficult did you find it to be? If you implemented the simpler version, why was that?
   (c) Print out a copy of the Disneyland map and use a pen to draw in the edges chosen by your algorithm in Main Street (sites 1 – 14) and Tomorrowland (68 – 84). (You may fill in all the edges if you wish, but it's not required since that seemed too tedious.) Is your graph acyclic, as it should be? Does it look like a minimum-weight spanning tree to you? *Note:* you don't need to turn in your filled-in map, just use it to answer this question.
   (d) Does the total weight of your tree match that of the sample output above? What about the set of edges? If there are any discepancies, why do you think that is?

Due Date

The project deadline is Monday, 4/29, 9 AM. Late submissions will not be accepted.