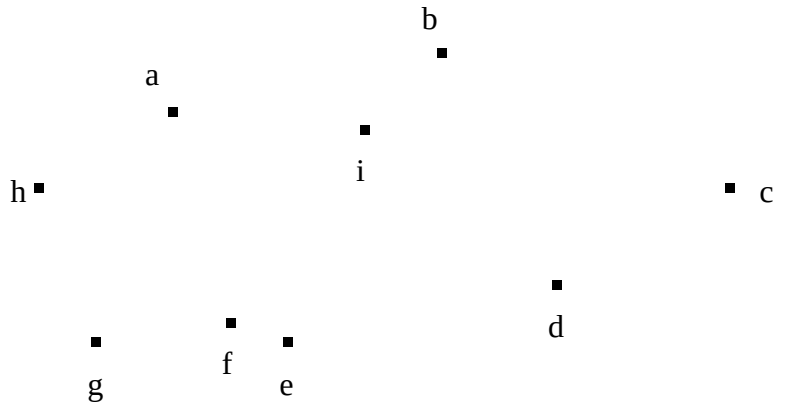
1. Suppose you had a map of settlements on the planet X



We want to build roads that allow us to travel between any pair of cities. Because resources are scarce, we want the total length of all roads build to be minimal.

a) Which pair of cities would you connect first? Why these cities?

b) What cities would you connect next?

c) Outline in English the algorithm you are following in (a) and (b).

d) What would be some characteristics of the resulting "graph" after all the cities are connected?

e) What would be the run-time of your algorithm?

f) Does your algorithm come up with the overall best (globally optimal) result?

2. Prim's algorithm for determining the minimum-spanning tree of a graph is an example of a *greedy algorithm*. Unlike divide-and-conquer and dynamic programming algorithms, greedy algorithms DO NOT divide a problem into smaller subproblems. Instead a greedy algorithm builds a solution by making a sequence of choices that look best ("locally" optimal) at the moment without regard for past or future choices (no backtracking to fix bad choices).

a) What greedy criteria does Prim's algorithm use to select the next vertex and edge to the partial minimum spanning tree?

b) What data structure could be used to efficiently determine that selection?

c) Below is the updated (http://www.pythonworks.org/) textbook's Prim's algorithm code, how might it be improved?

The version of Prim's algorithm in listing 6.11 contains typos...missing parenthesis on getDistance in line 7. The cost method is actually called getCost (lines 11,13, and 14).

```
00001: def prim(G,w,start):
00002:     PQ = PriorityQueue()
00003:     for v in G:
00004:         v.setDistance(sys.maxint)
00005:         v.setPred(None)
00006:     start.setDistance(0)
00007:     PQ.buildHeap([(v.getDistance(),v) for v in G])
00008:     while not PQ.isEmpty():
00009:         u = PQ.delMin()
00010:         for v in u.getAdj():
00011:             if v in PQ and u.getCost(v) < v.getDistance():
00012:                 if v.getPred() != None:
00013:                     G.findEdge(v,v.getPred()).remove()
00014:                 v.setPred(u)
00015:                 v.setDistance(u.getCost(v))
00016:                 PQ.decreaseKey(v,u.getCost(v))
```