Dijkstra's Algorithm

Let's consider how we can adapt Dijkstra's Algorithm to solve the problem we called "Problem 1"

Problem 1: Given a communications network in which each connection has a known bandwidth, find a path from A to B that maximizes the minimum bandwidth along the path, using each link at most once.

Here's Dijkstra's Algorithm again, with the comments removed:

```
Cost(A) = 0
Cost(v) = ∞  for all v != A
Pred(v) = ""        for all v
Reached = {}
Candidates = {A}

while Candidates != {} :
       let x be the vertex in Candidates with minimum Cost value
       add x to Reached, and remove x from Candidates
       for each vertex y such that y is a neighbour of x
                                          and y is not in Reached:
          if Cost(x) + w(x,y) < Cost(y):
             if Cost(y) == ∞:
                  add y to Candidates
             Cost(y) = Cost(x) + w(x,y)
             Pred(y) = x

return Cost, Pred
```

We can look for the similarities and differences between the Shortest Path problem (which Dijkstra's Algorithm solve) and the Max Bandwidth problem. The key to the correctness of Dijkstra's Algorithm is the observation that when we choose the candidate x with the lowest Cost, we know that there cannot be an undiscovered path to x with a lower cost because any such path would have to go through one of the other candidates, and their costs *are already at least as large as the known cost of x*.

We need to find a similar structure of the the Max Bandwidth problem. Consider the initial situation: we are starting at vertex A, and we can see the bandwidth on the edges that join A to its neighbours. Let x be the neighbour with the highest-bandwidth connection from A. Any other path from A to x must go through one of A's other neighbours ... the bandwidth of this path cannot exceed the bandwidth of its first edge ... and that is $\leq$ the bandwidth on the edge that goes directly from A to x. Thus the first choice is correct. We can extend the same reasoning to prove that if we always choose the candidate with the maximum bandwidth, our paths from A to the other vertices will all be optimal.

For the original problem, the cost of a path is the sum of the weights of the edges in the path – this is easy to compute by adding the weight of each edge (when we choose it) to the weight of the path we are extending. For the Max Bandwidth problem, when we extend a path by adding one more edge the bandwidth of the extended path is either the same as it was before, or it reduces because the new edge has a lower bandwidth than the path it is extending. This just means we take the minimum of the existing path's bandwidth and the new edge's bandwidth.

So what do we have so far?

- At each stage, we choose the candidate with the *highest* bandwidth instead of the *lowest* cost.
- Instead of adding the edge weights in a path, we keep track of the minimum edge weight in the path.

It turns out that is all we need ... plus revising the initialization. In the Shortest Path problem, we start with Cost(A) = 0 and Cost(v) = ∞ for all v $\neq$ A ... which makes sense because we want to start with high estimates of the cost of reaching each vertex, and drive those costs down as we find better and better paths. But for the new problem we want to start with low estimates of the best achievable bandwidth for each vertex, and drive those bandwidths up as we find better and better paths. So we can replace the Cost() vector with BW() and initialize it like this: BW(A) = ∞ and BW(v) = 0 for all v $\neq$ A

(Note that in both versions of the algorithm, it is not necessary to use ∞ in the initialization – any sufficiently large value will do.)

Now we can state the Max Bandwidth version of Dijkstra's Algorithm. I have highlighted the changes – you can see that the structure of the algorithm has not changed at all.

```
BW(A) = ∞
BW(v) = 0   for all v != A
Pred(v) = ""        for all v
Reached = {}
Candidates = {A}

while Candidates != {} :
       let x be the vertex in Candidates with maximum BW value
       add x to Reached, and remove x from Candidates
       for each vertex y such that y is a neighbour of x
                                          and y is not in Reached:
          if min(BW(x) , w(x,y)) > BW(y):
             if BW(y) == 0 :
                  add y to Candidates
             BW(y) = min(BW(x) , w(x,y))
             Pred(y) = x

return BW, Pred
```

You should work through this algorithm on a small example to see it in action.

Challenge:

Modify Dijkstra's Algorithm to solve Problem 3. I have provided a sample set of cities and flights called "flights.txt". Each line has four tab-separated values :

- departure city – a string
- destination city – a string
- departure time – an integer
- arrival time – an integer

The current time is 0. Your task is to find the sequence of flights that will get you from "Harrow" to "Port Alma" at the earliest possible time.