Shortest Path Algorithms

Dijkstra's Algorithm for the Single Pair Shortest Path Problem

In a weighted directed graph with no negative weights, in the worst case, it takes just as long to solve the single pair shortest path problem using Dijkstra's algorithm as it takes to solve the single source problem. However, it is possible that it will take less time. Suppose, for example, that the minimum weight path from s to t is less than the maximum weight of any path from x. Then, as soon as we move the node t to the "fully processed" set, we are done, since we don't care about the values for the remaining nodes that are not fully processed.

Equivalent Weightings

Suppose G is a directed graph. Let W_1 and W_2 be two edge weightings on G, with no negative cycles. If σ is any path in G, we let $|\sigma|_i$ be the sum of the edges of σ, using the weight W_i. If x, y are nodes of G, we define $d_i(x, y)$ to be the the smallest value of $|\sigma|_i$ for any path σ from x to y.

We say that W_1 and W_2 are *equivalent with respect to shortest paths* if the two weight functions give the same paths for all shortest path problems. That is, if σ is a path from x to y, then $|\sigma|_1 = d_1(x, y)$ if and only if $|\sigma|_2 = d_2(x, y)$. If h satisfies the condition that $h(x) \leq W_1(x, y) + h(y)$ for every edge (x, y), then W_2 will have no negative values.

Computing an Equivalent Weighting

We can compute a weighting W_2 equivalent to W_1, if we are given a function h on G. For any edge (x, y) of G, define

$$W_2(x, y) = W_1(x, y) - h(x) + h(y)$$

Johnson's Algorithm

In Johnson's algorithm, given a weighted directed graph (G, W_1) with no negative cycles, we first define G^* to be the augmented graph consisting of all the nodes and edges of G, together with one additional node s^*, plus an edge from s^* to every node of G. We then extend the weighting W_1 to G^* by defining $W_1(s^*, x) = 0$ for all $x \in G$. Let $f(x)$ be the length of the shortest path from s^* to x in G^*. Note that $f(x) \leq 0$ for all x. Use $h(x) = -f(x)$ to define an equivalent weight function W_2. All values of W_2 will be non-negative. We can then apply Dijkstra's algorithm.

The A* Algorithm

Let $G_1 = (G, W_1)$ be a weighted directed graph with no negative cycles, and let s, t be distinguished *source* and *target* nodes. Let h be a function (the *heuristic* on G which satisfies the condition that $h(x) \leq W_1(x,y) + h(y)$ for any edge (x,y) of G. Let $W_2(x,y) = W_1(x,y) - h(x) + h(y)$. We can then solve the single pair shortest problem in G_1 by solving the same problem for $G_2 = (G, W_2)$.

Will it be faster? That depends on the heuristic. Remember: there is no general way to find a heuristic; it has to come out of the specific application in some way, or be given to you by the person (me in this case) who gives you the problem.

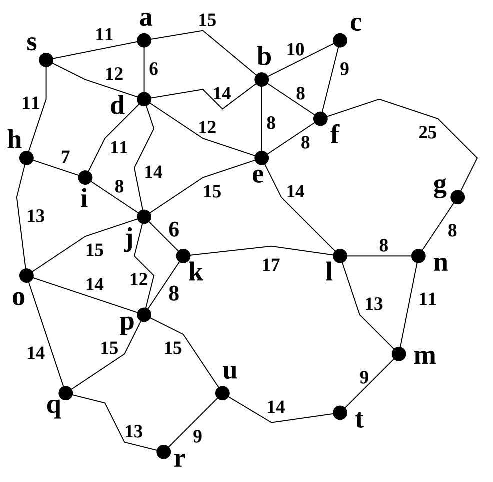
Assignment

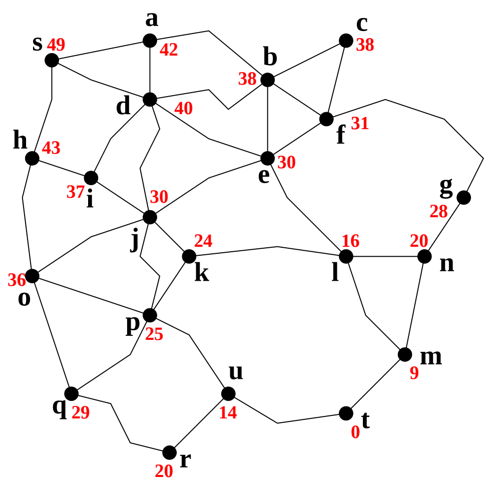
Consider the graph G illustrated below. The first figure shows G has a weighted graph, where the number on an edge $\{x, y\}$ is equal to both $W_1(x,y)$ and $W_1(y,x)$. That is, we can think of G_1 as a *symmetric* weighted graph. How much time would it take to find the shortest path from s to t? Your heap would eventually contain all the nodes. (Or maybe all but one; I'm not sure.) So, you pay for the single source solution, even though you don't need it.

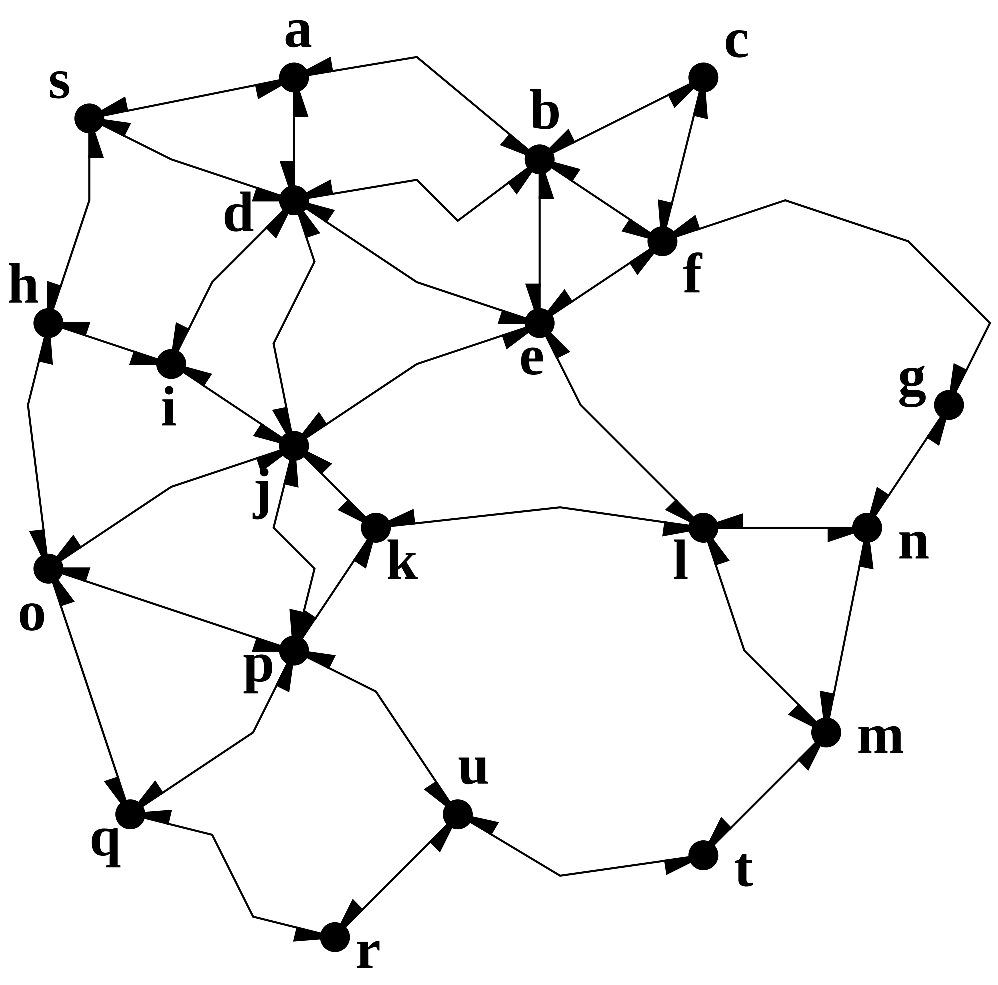
The second figure shows the heuristic, in red. (Don't ask how I came up with it. I just made up numbers that work.)

Here is your assignment. Note that in the large copy of G, each edge has been replaced by two edges, one in each direction.

1. Mark the large copy with the second weight function, W_2.
2. Use Dijkstra's algorithm for $G_2 = (G, W_2)$ to find the shortest path, in G_1, from s to t. It is amazingly fast.






a
b
c
s
d
f
h
e
g
i
j
k
l
n
o
p
m
u
q
t
r

Edit Distance

The dynamic programming algorithm for edit distance can be replaced by Dijkstra's algorithm. The running is then much smaller, if the two words are close.

Find the edit distance between *fluorouracil* and *florouricil*. The matrix obtained by dynamic programming is shown in Figure 1. Figure 2 shows values at those nodes which will be visited and placed in the heap when we use Dijkstra's algorithm.

		f	*l*	*o*	*r*	*o*	*u*	*r*	*i*	*c*	*i*	*l*
	0	1	2	3	4	5	6	7	8	9	10	11
f	1	**0**	1	2	3	4	5	6	7	8	9	10
l	2	1	**0**	1	2	3	4	5	6	7	8	9
u	3	2	**1**	1	2	3	4	5	6	7	8	9
o	4	3	2	**1**	2	2	3	4	5	6	7	8
r	5	4	3	2	**1**	2	3	4	5	6	7	8
o	6	5	4	3	2	**1**	2	3	4	5	6	7
u	7	6	5	4	3	2	**1**	2	3	4	5	6
r	8	7	6	5	4	3	2	**1**	2	3	4	5
a	9	8	7	6	5	4	3	2	**2**	3	4	5
c	10	9	8	7	6	5	4	3	3	**2**	3	4
i	11	10	9	8	7	6	5	4	4	3	**2**	3
l	12	11	10	9	8	7	6	5	5	4	3	**2**

Figure 1

		f	*l*	*o*	*r*	*o*	*u*	*r*	*i*	*c*	*i*	*l*
	0	1	2	3								
f	1	**0**	1	2	3							
l	2	1	**0**	1	2	3						
u	3	2	**1**	1	2	3						
o		3	2	**1**	2	2	3					
r			3	2	**1**	2	3					
o				3	2	**1**	2	3				
u					3	2	**1**	2	3			
r						3	2	**1**	2	3		
a							3	2	**2**	3		
c								3	3	**2**	3	
i										3	**2**	3
l											3	**2**

Figure 2

Figure 3 shows the heuristic h obtained by considering only the length of the remaining suffix. The adjusted edit distance between the two words is now only 1. When we use Dijkstra's algorithm using adjusted weights, we only compute the values shown in Figure 4.

		f	*l*	*o*	*r*	*o*	*u*	*r*	*i*	*c*	*i*	*l*
	1	2	3	4	5	6	7	8	9	10	11	12
f	0	1	2	3	4	5	6	7	8	9	10	11
l	1	0	1	2	3	4	5	6	7	8	9	10
u	2	1	0	1	2	3	4	5	6	7	8	9
o	3	2	1	0	1	2	3	4	5	6	7	8
r	4	3	2	1	0	1	2	3	4	5	6	7
o	5	4	3	2	1	0	1	2	3	4	5	6
u	6	5	4	3	2	1	0	1	2	3	4	5
r	7	6	5	4	3	2	1	0	1	2	3	4
a	8	7	6	5	4	3	2	1	0	1	2	3
c	9	8	7	6	5	4	3	2	1	0	1	2
i	10	9	8	7	6	5	4	3	2	1	0	1
l	11	10	9	8	7	6	5	4	3	2	1	0

Figure 3

		f	*l*	*o*	*r*	*o*	*u*	*r*	*i*	*c*	*i*	*l*
	0	2										
f	0	**0**	2									
l	2	0	**0**	2								
u		2	**0**	1	3							
o			2	**0**	2							
r				2	**0**	2						
o					2	**0**	2					
u						2	**0**	2				
r							2	**0**	2			
a								2	**1**	3		
c									3	**1**	3	
i										3	**1**	3
l											3	**1**

Figure 4

Can the King Quit in a Week or Less?

The Grand Vizier of Wonderland has been summoned by the King. "I'm tired of this job," he complains.

"According to the ancient Word-Ladder rules laid down by our founder in 1877, you can only change one letter a day." The Vizier stroked his beard. "I can take you from `KING` to `QUIT` in eight days."

```
KING
SING
SANG
SAND
SAID
SLID
SLIT
SUIT
QUIT
```

"My stable boy could do that well!" thundered the King, "I don't want to sit around here more than another week!"

Simple Supergraphs

Suppose a weighted directed graph G is a subsgraph of a larger graph S, which we call the "supergraph," and suppose that, because of its regular structure, it is very easy to solve shortest path problems in S. Then we can use the weights of the shortest paths in S as heuristics for applying the A^* algorithm to G.

For example, let G be the word-ladder graph of 4-letter common English words, and S the word-ladder graph of all strings of length 4 over the Roman alphabet, which consists of $26^4 = 456976$ words, each of which has degree 100. The shortest distance between two words in S is simply the number of places where a letter differs, which is never more than 4. For the King's problem, $h(x)$ is the distance from x to `QUIT`, and thus $h(\texttt{KING}) = 4$.

Design an efficient algorithm for solving the four letter English word-ladder problem. Assume that your program has access to a dictionary of all four letter English words, where *find* can be executed quickly.