Homework 4 Solutions

Chris Wu

The following solution were prepared by me (Chris), so if you find a typos email me at wu@cs.nyu.edu and not the professor. One comment is that these solutions are *complete* and contain many steps that are written for explanatory purposes so don't worry if you didn't write everything here.

Question 1

This is the solution for the non-programming portion. A good approach to this would be a greedy algorithm. For any cell tower, the range of houses it covers is an 8 mile interval. We move through the houses from left to right. The idea is that when we come upon an unmarked house, we'll place a tower 4 miles to it's right. Then we'll mark all houses that fall in that 8 mile range.

Notice that we don't really need to keep a boolean array of variables. As long as we don't move left, we can implicitly say that any house on our left has been 'processed' and those are the right are not.

Assume the input is an array H of the n house locations ordered left to right. The output will be the queue of locations for the cell towers, L. Here's the pseudocode:

```
i = 0
while i < n do
  L.enqueue(H[i] + 4)
  x = H[i]
  while H[i] < x + 8 do
    i++
  end while
end while
```

What's the running time? It's clearly linear since we do only constant work for each iteration and we only make one pass through the data.

Question 2

Here are the caches during the 3 times of eviction policies:

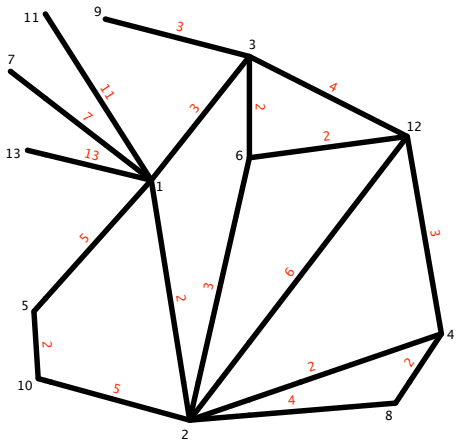
Request	Farthest-in-future	FIFO Queue	FIFO with hit moves
d	abc	abc	abc
a	abd	bcd	bcd
d	abd	cda	cda
b	abd	cda	cad
a	abd	dab	adb
c	abd	dab	dba
a	acd	abc	bac
e	acd	abc	bca
d	ecd	bce	cae
b	ecd	ced	aed
c	ecb	edb	edb
e	ecb	dbc	dbc
f	ecb	bce	bce
a	efb	cef	cef
d	eab	efa	efa
b	edb	fad	fad
e	edb	adb	adb
a	edb	dbe	dbe

The Farthest-in-future policy causes 8 misses. The FIFO causes 15 as does the modified FIFO.

Question 3

Part a)

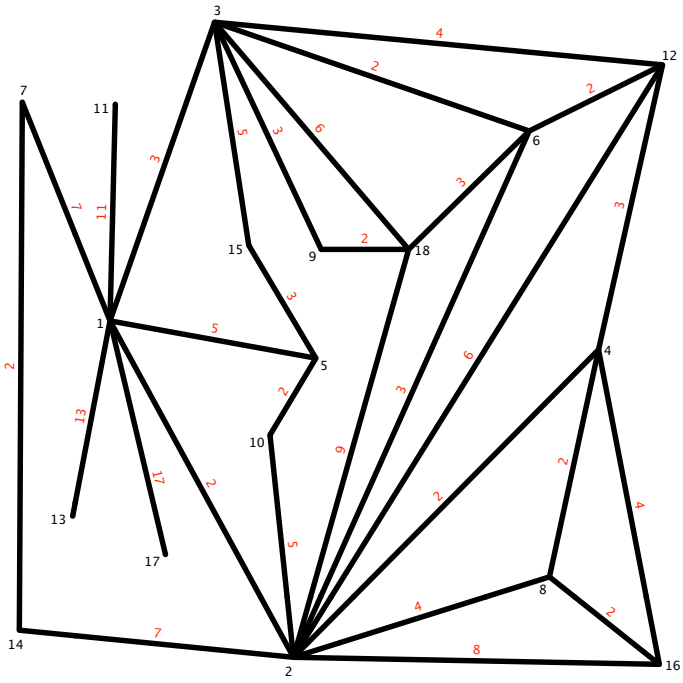
Here is G_{13}:



The vertex names are in black, the edges weight in red. The odd placement was simply to make it planar and more readable.

Part b)

Here is the graph for G_{18}:



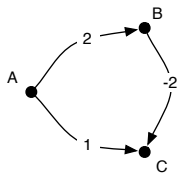
The matrix for Prim's algorithm is:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0 X	∞ 2 X	∞ 3	∞ 2 X	∞ 5	∞ 3	∞ 7	∞ 4 2 X	∞	∞ 5	∞ 11	∞ 6 3	∞ 13	∞ 7	∞	∞ 8 4 2	∞ 17	∞ 9
		X			2 X			3 X			2 X			5	X		6 3 2
				X		 X			2 X				 2	3 X			X
										X		X	X			X	

Question 4

Part a)

This will do



If we run Dijkstra's algorithm on this starting at A, the algorithm will prematurely declare that AC is the shortest path.

Part b)

Nothing. The problem with negative edges in Dijkstra is that the algorithm makes an assumption about paths it has not yet seen. Namely, that they are longer. This is a correct assumption if the edges are non-negative.

Prim's doesn't have that problem since it makes decisions based on the partial order of the edges and whether or not their addition creates cycles.

Another way to see that Prim's is ok with negative edge weights is to add a giant constant to all the edge weights. If we add a constant larger than the absolute value of the smallest negative edge then we'll have a non-negative weighted graph. Prim's handles that fine and return an MST that will be the same MST and for original graph we had.

Part c)

A	B	C	D	E	F	G
0	∞	∞	∞	∞	∞	∞
X	7	1	10	13		
		X		11		17
	X		9		16	
			X			16
				X		15
						X
					X	

Question 5

h	e	l	o	!	" "	t	i	s	m	y	w	r	d
2	2	5	2	2	5	3	3	2	1	1	1	1	1

All the frequencies are divided by 31 but since they're all over the same denominator,

I'll just ignore them.

I'm not going to draw all the tree, just the sets of nodes of the forest as they grow. This is only one possible solution since all ties can be broken arbitrarily leading to many many possible solutions.

{h}{e}{l}{o}{!}{“ ”}{t}{i}{s}{m}{y}{w}{r}{d}
{h}{e}{l}{o}{!}{“ ”}{t}{i}{s}{m}{y}{w}{r,d}
{h}{e}{l}{o}{!}{“ ”}{t}{i}{s}{m}{y,w}{r,d}
{h}{e}{l}{o}{!}{“ ”}{t}{i}{s}{m,y,w}{r,d}
{h}{e}{l}{o}{!}{“ ”}{t}{i}{m,y,w}{s,r,d}
{h,e}{l}{o}{!}{“ ”}{t}{i}{m,y,w}{s,r,d}
{h,e}{l}{o,!}{“ ”}{t}{i}{m,y,w}{s,r,d}
{h,e}{l}{o,!}{“ ”}{t,i}{m,y,w}{s,r,d}
{h,e}{l}{o,!}{“ ”}{t,i}{m,y,w,s,r,d}
{h,e,o,!}{l}{“ ”}{t,i}{m,y,w,s,r,d}
{h,e,o,!}{l,“ ”}{t,i}{m,y,w,s,r,d}
{h,e,o,!}{l,“ ”}{t,i,m,y,w,s,r,d}
{h,e,o,!,l,“ ”}{t,i,m,y,w,s,r,d}

By this run of the algorithm, we'd get this tree: