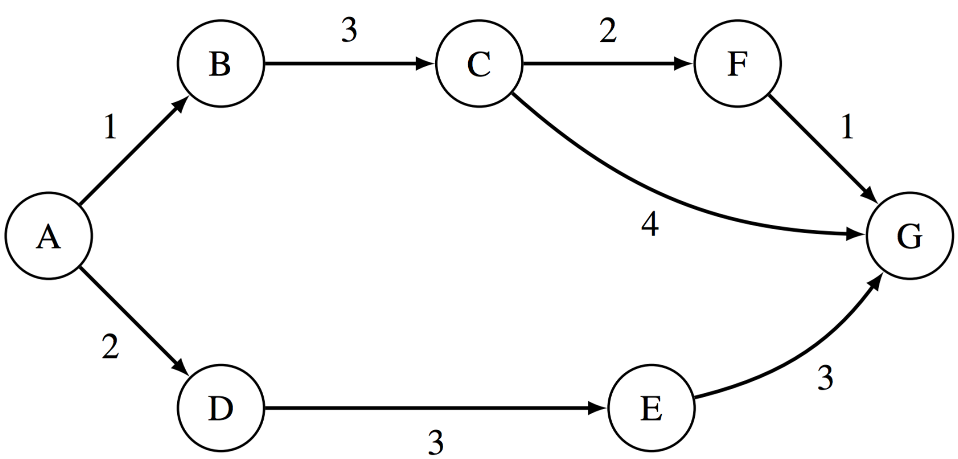
CS 61B Spring 2018 — Graphs & Sorting

Discussion 12: April 10, 2018

1 Dijkstra's Algorithm

For the graph below, let `g(u, v)` be the weight of the edge between any nodes `u` and `v`. Let `h(u, v)` be the value returned by the heuristic for any nodes `u` and `v`.



Edge weights	Heuristics
$g(A,B)=1$	$h(A,G)=8$
$g(B,C)=3$	$h(B,G)=6$
$g(C,F)=4$	$h(C,G)=5$
$g(C,G)=4$	$h(F,G)=1$
$g(F,G)=1$	$h(D,G)=6$
$g(A,D)=2$	$h(E,G)=3$
$g(D,E)=3$	
$g(E,G)=3$	

1.1 Run Dijkstra's algorithm to find the shortest paths from A to every other vertex. You may find it helpful to keep track of the priority queue and make a table of current distances.

$$A \to B = 1$$
$$A \to C = 4$$
$$A \to D = 2$$
$$A \to E = 5$$
$$A \to F = 6$$
$$A \to G = 7$$

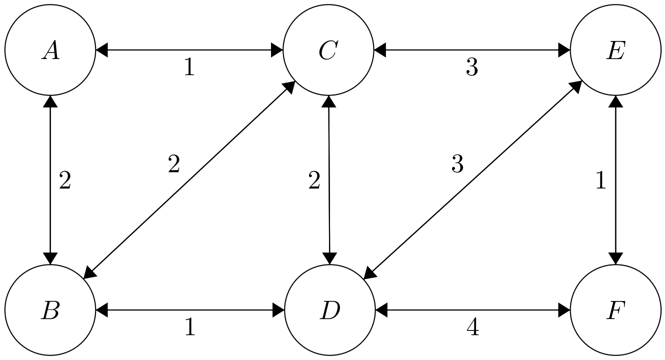
1.2 Given the weights and heuristic values for the graph below, what path would A* search return, starting from A and with G as a goal?

A* would return $A - D - E - G$.

1.3 Is the heuristic admissible? Why or why not?

A heuristic is admissible if all of its estimations `h(x)` are optimistic. No it's not, because the actual shortest path from $A \to G$ is of cost 7 if we take the northern route, but the heuristic estimates it will cost 8.

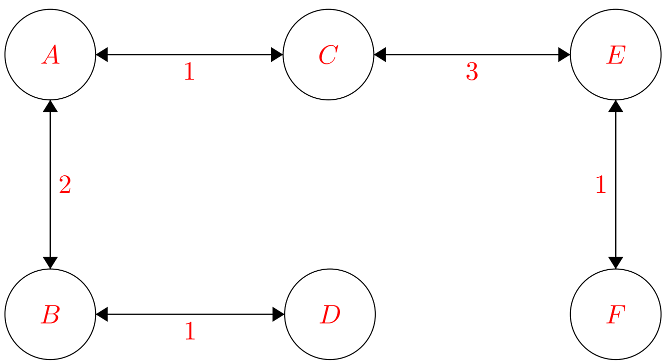
2 Minimum Spanning Trees



2.1 Perform Prim's algorithm to find the minimum spanning tree. Pick A as the initial node. Whenever there is more than one node with the same cost, process them in alphabetical order.

2.2 Use Kruskal's algorithm to find a minimum spanning tree.

In this case, Prim and Kruskal's output the same MST. This is not always the case.



2.3 There are quite a few MSTs here. How many can you find?

There are three choices to use an edge of weight 2 that can be used interchangeably and there are two choices of using an edge of weight 3 that can be used interchangeably. So there are 3 * 2 = 6 possible MST's. This math does not always lead to this. The key thing to note is that we could replace one of the weight 2 edges with another weight 2 edge and the entire graph would be spanning. Same for the weight 2 edges.

3 Mechanical Sorting

3.1 Show the steps taken by each sort on the following unordered list:

`0, 4, 2, 7, 6, 1, 3, 5`

(a) Insertion sort

```
0 | 4 2 7 6 1 3 5
0 4 | 2 7 6 1 3 5
0 2 4 | 7 6 1 3 5
0 2 4 7 | 6 1 3 5
0 2 4 6 7 | 1 3 5
0 1 2 4 6 7 | 3 5
0 1 2 3 4 6 7 | 5
0 1 2 3 4 5 6 7 |
```

(b) Selection sort

```
0 | 4 2 7 6 1 3 5
0 1 | 2 7 6 4 3 5
0 1 2 | 7 6 4 3 5
0 1 2 3 | 6 4 7 5
0 1 2 3 4 | 6 7 5
0 1 2 3 4 5 | 7 6
0 1 2 3 4 5 6 | 7
0 1 2 3 4 5 6 7 |
```

(c) Merge sort

[0 4 2 7 6 1 3 5]
[0 4 2 7] [6 1 3 5]
[0 4] [2 7] [6 1] [3 5]
[0] [4] [2] [7] [6] [1] [3] [5]
[0 4] [2 7] [1 6] [3 5]
[0 2 4 7] [1 3 5 6]
[0 1 2 3 4 5 6 7]

(d) Use heapsort to sort the following array (hint: draw out the heap). Draw out the array at each step:

`0, 6, 2, 7, 4`

```
7 6 2 0 4 (turns the array into a valid heap)
6 4 2 0 7 ('delete' 7, then sink 4)
4 0 2 6 7 ('delete' 6, then sink 0)
2 0 4 6 7 ('delete' 4, then sink 2)
0 2 4 6 7 ('delete' 2)
0 2 4 6 7 ('delete' 0)
```