1. The happy line problem models conversation between pairs of people waiting in line. A person may either converse with no one, or with the person immediately in front, or the person immediately behind (in the line), but may not converse with both. The input provides a happy-score for each consecutive pair of people, that shows how much happiness is generated when that pair converses. A person conversing with no one generates 0 happiness. The goal is to determine the highest possible attainable total happiness for a given line – the sum of happy-scores over all conversing pairs of people in the line.

A. Given the line shown below as a graph, with edges between consecutive nodes providing the input happy-scores, circle the pairs of nodes that should converse to generate maximum happiness.

B. It is claimed that the optimal solution to a happy line problem should always include as a conversing pair, the two consecutive people having the highest happiness score. Is this correct? Explain.

That's so wrong! Have a look at part A.

2. A partial execution of Kruskal's minimum spanning tree algorithm, applied to the weighted graph shown below on the left, using Union-Find trees (weighted union and path compression), has generated the union-find tree structures shown on the right. Show the resulting tree structure upon completion of the execution.
3. Two finite state machines are shown below as graphs. Each edge is labeled with the cost of the associated transition. Determine for each machine the minimum constant that gives a valid amortized cost for a transition, assuming that in both cases the initial state is given by A. Write your answers next to the respective graphs.

![Graph 1](image1)

![Graph 2](image2)

4. Use the depth first search method to topologically sort the graph shown below. The execution should have `disSweep` make its first call to `dfs` at the node C, and otherwise process the nodes in an order such that it generates two depth-first search trees. Show the two resulting trees, with each node labeled with `discover` and `finish` times, and show the resulting sorted order.

![Graph 3](image3)

Topological sorted order: E, F, A, D, C, B

5. Consider an execution of Dijkstra's shortest path algorithm – using a priority queue – with the graph shown below provided as input, and the node S chosen as the source vertex. At the point in the execution that two nodes belong to the shortest path tree, list the fringe vertices, showing for each its associated key in the priority queue.

![Graph 4](image4)

<table>
<thead>
<tr>
<th>fringe nodes</th>
<th>keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
</tbody>
</table>