**Exercises 4**

**1. Bottleneck spanning tree**

Let $G = (V, E)$ a connected graph with positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of $G$; we define the bottleneck edge of $T$ to be the edge of $T$ with the greatest cost.

A spanning tree of $G$ is a minimum-bottleneck spanning tree if there is no spanning tree with a cheaper bottleneck edge.

a) Is every minimum-bottleneck spanning tree of $G$ a minimum spanning tree of $G$?

b) Is every minimum spanning tree of $G$ a minimum-bottleneck spanning tree of $G$?

**Solution.**

a) This is false. A counterexample is simpler when we allow edges with the same cost (Figure 1).

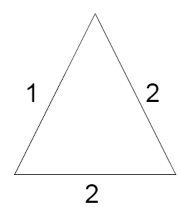


Figure 1: a minimum spanning tree must pick the edge with weight 1, but a minimum-bottleneck spanning tree might not

Now, let's build a counterexample for graphs with distinct edge costs. The idea is that a minimum-bottleneck spanning tree cares only about the most expensive edge (Figure 2).

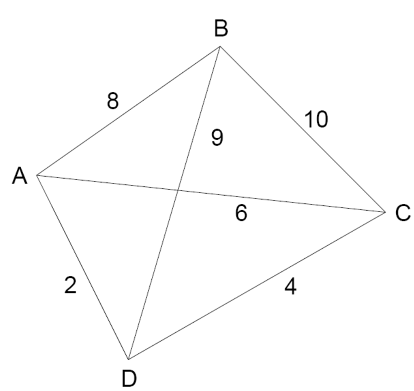


Figure 2: a MST of $G$ contains the edges $(A, D)$, $(C, D)$, $(A, B)$, with total cost 14, while a minimum-bottleneck tree can be $(A, B)$, $(A, C)$, $(C, D)$, with total cost 18.

b) This is true. Suppose that $T_1$ is a MST that is not a minimum-bottleneck spanning tree. Then $T_1$ uses an edge $e$ heavier than any edge in a minimum-bottleneck spanning tree $T_2$. Removing $e$ from $T_1$, we get two connected components. $T_2$, being a spanning tree, must have an edge connecting these two components, and such an edge has a cost strictly lower than $e$. Thus, such an edge can be added to $T_1$, obtaining a spanning tree with a weight strictly smaller than $T_1$. This is a contradiction with the assuption that $T_1$ is a MST.

**2. Cycle detection**
**Input**: a directed graph $G = (V, E)$.
**Output**: Does $G$ have a cycle?
Give a linear algorithm (in $|V| + |E|$) for this problem.

**Solution.**
This problem can be solved efficiently using depth-first-search. Let us briefly remind this generic procedure for systematically exploring a graph. We need an array, called `visited`, which keeps track of the vertices encountered so far.

```
Dfs(graph G)
{
 for each v ∈ V
       visited[v]=false;
 for each v ∈ V
       if (visited[v]=false)
          explore(v)
}

explore(v)
{
  pre-visit(v);
  visited[v]=true;
  for each edge (v,u)
      if (visited[u]=false)
         explore(u);
  post-visit(v);
}
```

In the code above, `pre-visit` and `post-visit` are two routines that can be customized to solve different problems. If we want only to explore a graph using DFS we don't need these routines.

Now, returning to our problem, we need another array, let's call it `path`, such that `path[v]=true` if the vertex $v$ has been encountered before on the path currently explored. The modified `explore` procedure is below.

```
explore(v)
{
  path[v]=true;
  visited[v]=true;
  for each edge (v,u)
      if (path[u]=1) we have a cycle
      else if (visited[u]=0)
              explore(u);
  path[v]=false;
}
```

Of course, we must initialize in the procedure DFS `path[v]=false` for all vertices $v$.

The running time of the whole algorithm will be $O(|V|+|E|)$.

**3. Mobile phone stations**

**Input**: the locations of $n$ houses along a straight line

We want to place cell phone base stations along the road so that every house is within 4 miles of one of the base stations.

**Output**: a minimal set of base stations.

**Solution.**

The algorithm is simple: pick the leftmost house not covered by a base station and place a base station 4 miles to the left.

Now let's prove that this approach is optimal. Suppose that our greedy algorithm places $m$ base stations at the locations $b_1 < b_2 < ... < b_m$ and assume that this solution is not optimal. Let then $OPT = \{b'_1 < b'_2 < ... < b'_k\}$ an optimal choice of base stations, with $k < m$.

Observe that $b'_1 \leq b_1$. This is because we put the first base station $b_1$ as far to the right as possible. Maybe $b'_2 \leq b_2$? Suppose that this does not hold, so we have $b'_1 \leq b_1 < b_2 < b'_2$. Since we needed to place a base station at $b_2$, there must be a house situated at a location $h$ such that $b_1 + 4 < h$. Then $b_2 = h + 4$. But this implies $b'_1 + 4 < h < b'_2 + 4$, so the location $h$ cannot be covered neither by $b'_1$, nor by $b'_2$, contradiction with the fact that $OPT$ is a valid solution. Thus $b'_2 \leq b_2$. Similarly $b'_3 \leq b_3, ..., b'_k \leq b_k$. But then there is no need to place another phone station at the location $b_{k+1}$, as all the houses that might be deserved by it are already covered by $b'_k$. Thus, our greedy strategy produces an optimal solution.

**4**. Prove that, in a Huffman coding scheme, if some symbol occurs with frequency strictly higher than 2/5, there is some codeword of length

1. Show also that if all symbols occur with frequency strictly less than 1/3 then there is no codeword of length 1.

**Solution.** Let $n$ be the number of letters and $f_1 \geq f_2 \geq ...f_n$ their relative frequencies. The statement is obvious for $n \leq 3$, so let's consider now the case $n = 4$. Suppose that there is no codeword of length 1. This implies that the Huffman tree looks like the one in Figure 3, because otherwise $f_1$ would have a codeword of lenght 1.

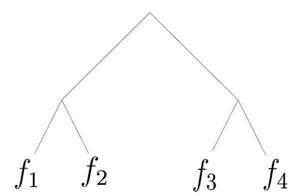


Figure 3

So $f_3 + f_4 \geq f_1 > 2/5$ (because the node corresponding to $f_3 + f_4$ is at a higher level than $f_1$). Then $f_3 \geq 1/5$, so $f_2 \geq 1/5$. Suming up we have:

$$1 = f_1 + f_2 + f_3 + f_4 > \frac{2}{5} + \frac{1}{5} + \frac{2}{5} = 1,$$

contradiction.

For the case $n > 4$ we make the following observation: we saw that, for a tree with 4 nodes, $f_1$ resists all merges until the last one, so it resists all the previous merges.

Now for the second part of the problem. First notice that $n \geq 4$ since all the frequencies are strictly less than 1/3.

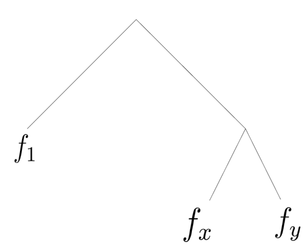


Figure 4

Again, suppose that $f_1$ has a codeword of length 1. So $f_1$ resists all merges except for the last one. When, during the construction of the code, there are only 3 nodes left, the tree looks like in Figure 4, where $f_x$ and $f_y$ might be cumulated frequencies resulted from previous merges. Then $f_x \leq f_1 < 1/3$, $f_y \leq f_1 < 1/3$, so $f_1 + f_x + f_y < 1$, contradiction.

**5. Matching points and intervals**

We are given $n$ points $x_1, ..., x_n$ and $n$ intervals, $I_1, ..., I_n$. The objective is to associate to each point $x_i$ an interval $I_k$ such that $x_i \in I_k$.

**Solution.** First we sort the intervals in increasing order of their final point. Suppose that the ordering is $I_1, ..., I_n$. The algorithm is the following: take $I_1$ and let $x_j$ the leftmost point contained in $I_1$ (if such a point does not exists, then the problem has no solution). Associate $x_j$ with $I_1$ and repeat the process.

We must prove that, if there is a matching between points and intervals, our algorithm finds such a matching. Denote by $M_k$ the matching we obtain after $k$ steps. We prove by induction on $k$ that, at any step, $M_k$ is included in a perfect matching.

First let's examine the case $k = 1$. Let $M$ be a perfect matching and suppose that $I_1$ is matched with $x_i$ in our greedy approach, while in $M$, $I_1$ is matched with $x_j$ and $x_i$ is matched with $I_q$. From the greedy choice we made, $x_i \leq x_j$ and the endpoint of $I_q$ is farther to the right than the endpoint of $I_1$. Then we can modify $M$ by matching $I_1$ with $x_i$ and $I_q$ with $x_j$.

Now to prove the induction step. Suppose that, after $k$ steps of our algorithm, the solution so far can be included in some perfect matching $M$. Let $(x_p, I_{k+1})$ the association made at the step $k + 1$ of the greedy algorithm and suppose that, in $M$, $x_q$ is mapped to $I_{k+1}$ and $I_l$ is mapped to $x_p$. Then neither $x_q$ nor $I_l$ appear previously in $M_k$. Also, from the greedy choice we made, $x_p \leq x_q$. Then we can modify $M$ by mapping $x_p$ to $I_{k+1}$ and $x_q$ to $I_l$.