A *path decomposition* of a graph (V, E) is a tree decomposition where the tree is a path, i.e., where each tree node has at most one child. We can then just list the sequence of bags: $(V_1, V_2, \ldots, V_r)$. A path decomposition is *nice* when $|V_1| = |V_r| = 1$ and at every next bag exactly one vertex is either added or removed, i.e., for every $i \in \{1, 2, \ldots, r-1\}$ there is a vertex $v \in V$ such that either $V_{i+1} = V_i \cup \{v\}$ or $V_{i+1} = V_i \setminus \{v\}$.

Let a graph $G = (V, E)$ be given with positive weights $w(e)$ for every edge $e \in E$ together with a nice path decomposition of G of width k. For any subset X of V the *weight* of that subset is the sum of the weights of edges (u, v) with $u \in X$ and $v \notin X$. The goal is to find the maximum weight possible over all subsets X, i.e.,

$$\max_{X \subseteq V} \sum_{(u,v) \in \{(u,v) \in E | u \in X, v \notin X\}} w(u, v).$$

Hint: use the notation $w_i(A, B)$ to denote the maximum total weight of edges between points from $\bigcup_{1 \leq j \leq i} V_j$ that are in different sets, when the partition is made consistent with a partition (A, B) of V_i.

1. (1 point) Give the maximum total weight over all partitions (A, B) of V_1 and explain.

Solution: Since $|V_1| = 1$ we know that $V_1 = \{v\}$ for some $v \in V$. Therefore, there are two partitions, and thus two possible solutions: $w_1(\{v\}, \emptyset) = w_1(\emptyset, \{v\}) = 0$.

2. (3 points) For $1 \leq i < r$, express the solution to $i+1$ in terms of those for i in case $V_{i+1} = V_i \cup \{v\}$ and argue why this is correct.

Solution: First observe that there can be no edge (u, v) with $u \in \left(\bigcup_{1 \leq j < i} V_j\right) \setminus V_i$; this follows from the property of a "separating tree node" for V_i and because $v \in V_{i+1}$. Thus all neighbors of v in this subgraph $\bigcup_{1 \leq j \leq i+1} V_j$ are in both V_{i+1} and V_i.

For a partition (A, B) of V_{i+1}, we can therefore compute the additional weight by the cut pairs involving v considering only vertices from V_i, and compute w recursively:

$$w_{i+1}(A, B) = \begin{cases} w_i(A \setminus \{v\}, B) + \sum_{u \in B} w(u, v) & \text{if } v \in A \\ w_i(A, B \setminus \{v\}) + \sum_{u \in A} w(u, v) & \text{otherwise} \end{cases} \quad (1)$$

3. (2 points) For $1 \leq i < r$, express the solution to $i+1$ in terms of those for i in case $V_{i+1} = V_i \setminus \{v\}$ and argue why this is correct.

Solution: For a partition (A, B) of V_{i+1}, we define

$$w_{i+1}(A, B) = \max\{w_i(A \cup \{v\}, B), w_i(A, B \cup \{v\})\} \quad (2)$$

Since the partition of V_i needs to be consistent with (A, B) we can only decide on where to put v. There are just two possible options: we take the best (maximum) one.

4. (2 points) Using the above recursive formulations, give the pseudocode of an iterative dynamic programming algorithm. As always, give the algorithm that uses the least space and run time.

   Hint: Compute $w_i(A, B)$ for all partitions (A, B) of V_i.

Solution:

1. For $i = 1$ to r do:
   (a) For all $A \subseteq V_i$ compute $w_i(A, V_i \setminus A)$ and store the answers in a table M of size $O(2^{|V_i|})$. For these computations use the definitions above, but instead of a recursive call, lookup the value in the table L for the previous iteration.
   (b) Let $L \leftarrow M$.
2. Eventually, return $\max_{A \subseteq V_r} w_r(A, V_r \setminus A)$ where the latter is the highest value in L.

Following examples on DP over a tree decomposition, you would expect a separate table for every "tree" node. However, this is not required, since every node is visited exactly once. Directly after that, the values are not needed anymore. (Subtract half a point if this is not done.)

5. (1 point) What is a tight upper bound on the space required by this dynamic programming algorithm?

Solution: The size of the two tables is $O(2^k)$, since k is the width and thus $k \geq |V_i| - 1$ for all i. (The inefficient version is $O(r \cdot 2^k)$.)

6. (1 point) What is a tight upper bound on the run-time complexity of this dynamic programming algorithm?

Solution: The outer for loop is done r times, the inner for loop exactly once per table entry, so $O(2^k)$. Filling in a table entry requires computing $\sum_{u \in B} w(u, v)$ which can be done in $O(k)$ (e.g., if edges are stored in a two dimensional matrix). This comes down to a total running time of $O(2^k \cdot k \cdot r)$.

End of assignment