We will see a different kind of embedding into trees, called tree decomposition, in which each vertex of the guest graph may be mapped to a set of nodes rather than a single node. This will lead to the notion of *treewidth*. Graphs that have good decompositions will be seen to have faster algorithms for MIS and several other NP-complete problems. Tree decompositions are useful in many areas, including probabilistic inference. We will see a connection to solving linear systems defined on graphs which arise in engineering calculations.

As we will see in the exercises, tree decompositions and standard embeddings into trees are related.

1 Tree Decomposition

Let G be an undirected unweighted graph. Then a tree decomposition for G consists of a tree T and a map $f : V(G) \rightarrow \{\text{subtrees of } T\}$, such that $(u, v) \in E(G) \Rightarrow f(u), f(v)$ have a non empty intersection.

For each vertex u define its load V_u as the set of vertices of G whose subtrees contain u. The width of the decomposition (T, f) is $\max_u |V_u| - 1$. The treewidth of a graph is the minimum possible depth, over all possible choices of T, f. For a collection of vertices U we will use V_U to denote $\cup_{u \in U} V_u$. While we will refer to the decomposition by (T, f), the decomposition is also specified completely by giving T and all the V_u.

Example: For any graph G, consider a tree decomposition in which T is a single node, and all vertices of G are mapped to subtrees consisting of a single node. This is a trivial example, but it demonstrates that every graph does have a tree embedding.

Example: Find tree embeddings for P_n, C_n, complete binary tree. Make the width as small as possible.

It turns out that every tree has a tree embedding of treewidth 1, in fact that is why we subtracted 1 from the definition of width.

Lemma 1 *A connected graph has treewidth* 1 *if and only if it is a tree.*

Proof: Suppose G is a tree. Then it has a (G, f) decomposition as follows. Root G at any vertex r. Assign $V_r = \{r\}$, and for $u \neq r$ assign $V_u = \{u, p\}$ where p is the parent of u. Each vertex is thus mapped to a subtree consisting of itself and its children, and hence each child subtree intersects with its parent tree.

We prove the converse later. ■

First we establish the decomposition properties provided by (T, f).

Lemma 2 *Suppose (T, f) is a tree decomposition of G. Suppose t is a node in T, and on removal of t we get subtrees $T_1, \ldots, T_k$. Let G_i denote the subgraph induced by $V_{T_i} - V_t$. Then G_i have no vertices in common, nor do any edges connect them. Further G_i are also obtainable from G by removing the vertices in V_t and their incident edges.*

Proof: We show that (T_i, f) is a decomposition for G_i, from which the result follows. G_i only contains vertices u such that $f(u)$ intersects with $V_{T_i} - V_t$. Suppose $f(u)$ is only partially contained in T_i, i.e. $f(u)$ contains a node $x \in T_i$ as well as a node y outside of T_i. But in T there is a unique path from x to y and this passes through t. Since $f(u)$ is a subtree of T, t must also belong to $f(u)$. But then $u \in V_t$, and thus $u \notin V_{T_i} - V_t$. Thus u is not in G_i. Thus G_i only contains vertices whose trees are completely contained in T_i. If u, v have an edge in G_i, then they have an edge in G, and hence their trees intersect. Thus (T_i, f) is a decomposition for G_i.

Since T_i are disjoint, it follows that G_i cannot share vertices. Also, because the subtrees of vertices in distinct G_i are fully contained in T_i, they cannot intersect. Thus they would not have an edge between them in G.

The only vertices and edges absent in G_i together are the vertices V_t and their incident edges. Hence removing the vertices in V_t from G should give G_i. ■

Lemma 3 *Suppose (T, f) is a tree decomposition of G. Suppose $e = (x, y) \in E(T)$. On removing e suppose we get subtrees T_x, T_y with x, y in each respectively. Let G_x be the graph induced by $V_{T_x} - (V_x \cap V_y)$, and similarly G_y. Then (T_x, f) is a decomposition for G_x, and similarly for G_y. Further, G_x, G_y are vertex disjoint, nor is there any edge in G connecting the graphs together. Further, G_x, G_y are obtained by removing $V_x \cap V_y$ from G.*

Proof: Replace (x, y) with two edges $(x, w), (w, y)$. Let $V_w = V_x \cap V_y$. Extend f to include w. The new tree and extended f also form a tree decomposition for G. Now remove the node w and use the previous lemma. ■

Lemma 4 *Let (T, f) be a tree decomposition for a graph G. Suppose T contains an edge (x, y) such that $V_x \subseteq V_y$. Then let T' be the graph obtained by merging x into y, i.e. y acquires all of x's neighbours and V_y remains unchanged. Then (T', f) is also a tree decomposition for G.*

Proof: Check that all the properties are maintained. ■

The above lemma defines a notion of a non-redundant decomposition, i.e. a decomposition which does not contain such edges. Note further that the contraction operation does not increase the width.

Proof of Lemma 1: (contd.) Suppose a graph G containing a cycle C has a decomposition (T, f). Consider the subtree T_C of T induced by subtrees $f(v)$ for $v \in V(C)$. (T_C, f) is a tree decomposition for C because each vertex v is mapped to a subtree $f(v)$, and $(u, v) \in E(C) \Rightarrow (u, v) \in E(G)$. Thus $f(u), f(v)$ intersect in T and hence in T_C. In T_C fewer trees intersect any node than in T. Hence the width of (T_C, f) is no larger than that of (T, f). Thus it suffices if we prove that (T_C, f) has treewidth at least 2.

So without loss of generality let us assume that G is a cycle, having (T, f) as its decomposition. Further assume it has width 1, i.e. $|V_t| \leq 2$ for all $t \in V(T)$. Without loss of generality, we can assume that T is non-redundant. Let (x, y) be any edge, and consider what happens after we remove it. Because of non-redundancy, we know that neither V_x nor V_y can equal $V_x \cap V_y$. Thus $V_{T_x} - (V_x \cap V_y)$ and $V_{T_y} - (V_x \cap V_y)$ must both be non empty. Thus removing (x, y) splits G into parts. But because of non-redundancy $|V_x \cap V_y| = 1$, and hence in G we have removed only one vertex. But a cycle cannot split by removing just one vertex. Thus we have a contradiction. ■

Exercise: Show that every graph of treewidth k must have a vertex of degree at most k.
Exercise: Every graph of n vertices and treewidth k must have a tree decomposition of width k and at most n nodes.

2 Computing Treewidth

This is NP-complete.

3 MIS

We wish to find a MIS for a weighted graph G with weights w having tree decomposition (T, f) of width k. We present an algorithm which has a divide-and-conquer flavour (and it may also be viewed as dynamic programming), and it is also brute force in certain parts. The general strategy works not only for MIS, but for many other problems.

The basic idea is to find an MIS (or whatever else we want to find) independently for different regions of the graph, and somehow stitch together the different solutions. For simplicity, suppose our graph is divided into two subgraphs G_1, G_2, which overlap in the set of boundary vertices B such that the edges in the graph are either entirely within $G1$, or entirely within G_2. Suppose we solve our problem (say MIS) separately on G_1 and on G_2. In general, the solutions, say S_1, S_2 will not agree on the boundary nodes B because a certain node in B may be included in S_1 but excluded in S_2. If we somehow knew what values S_B the optimal solution S takes on B, then we can ask for an MIS for G_1, G_2 which is consistent with S_B. This is the gist of the approach, though two points must be noted:

1. On G_1, G_2 we are no longer solving the original problem, but a boundary constrained problem in which the part S_B of the solution on B is fixed before hand. A boundary constrained MIS problem is really an MIS problem for a smaller graph; however in general (say for the node disjoint path problem or the graph colouring problem), the constrained problem can be quite different. However, usually it is not much worse than the original problem. Note also that the solution to the constrained problems is not obtained directly but recursively.

2. We dont really know S_B. So we consider all possible n_B ways in which the solution S could appear on the boundary, and solve boundary constrained problems for each such choice. We then stitch together the solutions for each choice S_B, and return that stitched solution that has the best objective value (the weight of the MIS, or whatever we are optimizing). Typically n_B will be exponential in the size of B, and

so we will solve that many boundary constrained problems. However, if the graph has low treewidth, then the size of the boundary will be much smaller than the size of the graph. Thus this approach will take time exponential in the size of the boundary, as opposed to the naive approach which will require time exponential in the size of G.

Before we develop this idea further, we need an observation about the structure implied in the tree decomposition of G. Let r be any node of T which we designate as root. For any other node u of T let $T(u)$ denote the subtree under u. Let H_u denote the graph induced by vertices $V_{T(u)}$. The vertices V_u contain the boundary vertices of H_u in the following sense.

Lemma 5 *No vertex in $V_{T(u)} - V_u$ has an edge in G to nodes outside $V_{T(u)}$.*

Proof: Similar to the proof of Lemma 2. ■

For each independent (in G) subset c of the boundary vertices V_u, we will find an MIS of H_u containing exactly the set c from V_u. Such an MIS will be denoted as S_u^c, and its weight, A_u^c. It will be seen that these can be computed by a recursive algorithm, and $\max_c\{A_r^c\}$ will give us the weight of the MIS for G.

The basis of the recursion is easy: when u a leaf, then $V_{T(u)} = V_u$, and so A_u^c for each independent set $c \subseteq V_u$ is just w_c. Since $|V_u| \leq k+1$, all such values, for any u can be computed in time[1] $\tilde{O}(2^{k+1})$.

The combine step is based on the following lemma. In what follows, we will use addition to mean set union, and products to mean intersection. Note that the condition "S contains exactly the set c from a set X" is simply expressed as $SX = c$.

Lemma 6 *Let u be a node in T, and $u_1, \ldots, u_d$ its children. Let S be an MIS for H_u such that $SV_u = c$. Then $S = c + \sum_i S_i$ where S_i is an MIS for H_{u_i} such that $S_i V_{u_i} = c_i$ for some set c_i such that $c_i V_u = V_{u_i} c$.*

Before we prove the lemma, note that the lemma assures us that S_u^c indeed can be built from some $S_{u_i}^{c_i}$. It does not tell us precisely which c_i, but we can simply try out all choices.

[1] We use $\tilde{O}(f)$ to denote $O(fg)$ where g may be any polynomial factor.

Proof: The basic idea is to "project" S onto $V_{T(u_i)}$. For this first note that $V_{T(u)} = V_u + \sum_i V_{T(u_i)}$. Noting that $S \subseteq V_{T(u)}$, we multiply the above by S and get

$$S = SV_{T(u)} = SV_u + \sum_i SV_{T(u_i)} = c + \sum_i S_i$$

where we define $S_i = SV_{T(u_i)}$. Noting that $V_{T(u_i)} = V_{u_i} + V_{T(u_i)} - V_{u_i}$, and also that $S_i \subseteq V_{T(u_i)}$, we have

$$S_i = S_i V_{T(u_i)} = S_i V_{u_i} + S_i(V_{T(u_i)} - V_{u_i})$$

We now define $S_i V_{u_i} = c_i$. The point to note is that S has a fixed part c in the region V_u, and so this must affect c_i as well. Thus

$$V_u c_i = V_u S_i V_{u_i} = V_u S V_{T(u_i)} V_{u_i} = V_u S V_{u_i} = c V_{u_i}$$

where $V_{T(u_i)} V_{u_i} = V_{u_i}$ because $V_{u_i} \subseteq V_{T(u_i)}$.

Finally, suppose S_i is not a MIS for H_{u_i} satisfying $S_i V_{u_i} = c_i$. Suppose instead there exists an MIS S_i' of H_{u_i} such that $S_i' V_{u_i} = c_i$ and weight $w_{S_i'} > w_{S_i}$. As for S_i we know that

$$S_i' = S_i' V_{u_i} + S_i'(V_T(u_i) - V_{u_i})$$

But $S_i V_{u_i} = c_i = S_i' V_{u_i}$. Thus S_i, S_i' differ only in which elements they pick out of $V_T(u_i) - V_{u_i}$. By Lemma 5 there is no edge from $V_T(u_i) - V_{u_i}$ to any vertex outside of H_{u_i}. Thus $S' = S - S_i + S_i'$ will also be an independent set for H_u satisfying $S' V_u = c$, and will have higher weight than S. This cannot happen and hence S_i' must not exist. ■

Thus we have shown that $S_i = S_{u_i}^{c_i}$ for some c_i which satisfy $c_i V_u = V_{u_i} c$. We can try out all possible choices of c_i and pick the $S_{u_i}^{c_i}$ of maximum weight. Notice that we need only consider c_i that are independent and satisfying $c_i V_u = V_{u_i} c$. Since c_i contains only $k+1$ vertices, there are at most 2^{k+1} choices to consider. Letting A_i denote the weight of S_i, we have

$$A_i = \max_{c_i} \{ A_{u_i}^{c_i} | c_i V_u = V_{u_i} c, c_i \text{ is independent} \}$$

Thus determining A_i takes time $\tilde{O}(2^{k+1})$. If we write $S = c + \sum_i S_i$ as a disjoint union, it will be useful for finding its weight. Different S_i are mutually disjoint, so it suffices to write $S = c + \sum_i S_i - cS_i$. But

$$cS_i = cc_i + cS_i(V_{T(u_i)} - V_{u_i}) = cc_i + cS_i(V_{T(u_i)} - V_{u_i}) = SV_u S_i(V_{T(u_i)} - V_{u_i}$$

Now u_i is between u any vertex in $T(u_i) - \{u_i\}$, and so vertices common to V_u and $V_{T(u_i)} - V_{u_i}$ must also be present in V_{u_i}, i.e $V_u V_{T(u_i)} = V_{u_i} V_u$. But then we have $cS_i = cc_i$. Thus we get $S = c + \sum_i S_i - cc_i$, with $cc_i \subseteq S_i$. Thus we can write

$$A_u^c = w_c + \sum_i A_i - w_{cc_i}$$

The time to find A_u^c for a single c is $\tilde{O}(d2^{k+1}) = \tilde{O}(2^{k+1})$. Because there are at most 2^{w+1} choices for c, the total time is $\tilde{O}(4^{k+1})$. Evaluating this for every node will produce a factor of n or so, which will be ignored in the $\tilde{O}$ notation.[2] This total time should be compared to the $\tilde{O}(2^n)$ time for the brute force method.

4 Chordal Graphs

A graph is said to be a chordal graph if it has maximum number of edges for a given tree decomposition (T, f). In other words, $f(u)$, $f(v)$ intersect if and only if (u, v) is an edge.

Examples: Trees, complete graphs.

Non-examples: Cycle on 4 or more vertices. A simple way to show that this is not chordal is to show that it does not have a perfect elimination ordering, as discussed below.

Chordal graphs have a so called "perfect elimination ordering". To understand this, let us consider the elimination problem.

Many engineering problems are solved using the so called finite element method. We have some physical object, say a beam, or a plate that is being heated. The goal is to find the stresses or the temperatures as the object is stimulated by a load or by placing heating elements nearby. It is customary to consider the value of the physical quantity (temperature, stress) at some finite number of points strategically located in the object. Based on the physics of the problem, for each point, an equation is formed which relates the magnitude of the quantity at point i with magnitudes of points geometrically close by (determined suitably). The equations are linear, and may be considered to be $Ax = b$ where x_i is the unknown value of the magnitude of the quantity at point i. The system may be solved by Gaussian elimination, and for numerical considerations, it is best to eliminate unknown

[2]There will not be separate n, d factors, but simply a single factor for the number of edges in H, and this must be at most n assuming H is non-redundant.

x_i using the ith equation.[3] At the start of the elimination process the matrix A is symmetric, and sparse. Thus to minimize work when solving it using Gaussian elimination, it is desirable to maintain sparsity, or minimize fill-in, i.e. non-zero elements introduced into the matrix during the elimination.

The non-zero elements in A define a graph: there is an edge (i, j) if $a_{ij} \neq 0$. While eliminating unknown x_i we make a_{kj} become non-zero if $a_{ki}, a_{ij} \neq 0$. In other words we introduce fill-in at a_{ij} if edge (i, j) was not originally present but edges $(i, j), (i, k)$ were present. After eliminating vertex i we will in general have the complete graph at all its neighbours, and all the edges might correspond to fill-in.

Consider now what happens if G is chordal. Assume that we have a non-redundant tree-decomposition (T, f) of it. Consider any leaf u, we know V_u must contain some v that does not appear in any other vertex of T. So we choose to eliminate v from G. This causes a clique to be placed between all its neighbours; however, the neighbours were already mutually connected because they are present in V_u and this is a chordal graph! Thus we have only modified elements in the matrix which were already non-zero, and have thus introduced no fill-in.

The new graph has the same tree decomposition, except for removing the leaf u. But the maximality property holds for other vertices of T, and hence the new graph is also chordal. So we get a non-redundant representation for it and repeat the process. So in the entire elimination process no fill-in is introduced.

Lemma 7 *A graph is chordal if and only if it has a perfect elimination ordering.*

Proof: We have proved the only if direction above. The if direction is an exercise. ■

5 Separability

An ϵ-separator for an n vertex graph is a subset S of vertices such that after removal of S each connected component contains at most ϵn vertices.

[3] Because we expect a_{ii} to be large, this strategy ensures that there is no overflow during elimination, or in general reduces loss of precision.

The choice $\epsilon = 2/3$ is a very common choice.

Suppose s is a function $s : N \to N$. An n vertex graph is said to be s-separable, if either $n = 1$ or G has $2/3$ separator of size $s(n)$, and subgraphs of G are also s-separable.

Theorem 1 *A graph of treewidth k is $k+1$-separable, i.e. $s(n) = k$ is a constant function.*

Proof: First consider special case $k = 1$, i.e. G is a tree. Here we can in fact find a single vertex separator using the algorithm below. Say G has been rooted, and `Separate` is called with the root.

```
Separate(r){
If no child of r has subtrees of size at least 2n/3
   return r.
Else,
   s = child of r with largest subtree underneath it
       among all children.
   return separate(s)
}
```

`Separate` is to be called with the precondition $P(r)$: the subtree under r has size at least $2n/3$. The precondition is true on the first call.

So we must prove correctness ensuring the precondition.

Suppose in a certain execution no subtree of size at least $2n/3$ is found. Because of the precondition, the subtrees together have size at least $2n/3$. Hence the component of the tree containing the parent of r can have size at most $n/3$. Thus removal of r will indeed leave all components of size at most $2n/3$ as needed.

Suppose a subtree of size $2n/3$ is found. Then we recurse on that. But this guarantees the precondition.

Finally, the program must terminate because the size of the subtree on which `Separate` is called keeps decreasing, and it cannot decrease below $2n/3$.

For general k, a simple idea is to remove the vertices V_r rather than r. Then we recurse on the child u such $V_{T(u)} - V_r$ has the largest size. It suffices to remove $k+1$ vertices, the maximum number of vertices in any node r. ■[4]

[4]Consider the chordal graph whose tree decomposition is a 4 leaf star, in which the root

Theorem 2 *A k separable graph of n nodes, degree d, has bisection width $O(dk \log n.)$*

Proof: The proof is constructive. We will apply separation and put resulting pieces into a bin of capacity $n/2$ till the bin becomes exactly full. We will then show that not many edges go from the vertices in the bin to those outside.

At any non terminal step of the algorithm we will have accumulated some *pieces* (subgraphs resulting from the separation) in the bin, and also a set of separator nodes outside the bin. We will also have determined that some pieces must definitely not to be put into the bin, and we will have one subgraph G which contains more vertices than the unfilled bin capacity. The subgraphs in the bin will have edges only to the separator nodes. We next separate G and break it a separator set S and several into pieces. We will try to put each piece into the bin in turn. Either all pieces fit, in which case we put in as many nodes from S as needed to fill the bin. Or, we might find a piece which cannot fit. Then we recurse on that piece.

At the end we will have some subgraphs inside the bin, and some separator nodes as well. Each subgraph can have edges only to separator vertices, inside or outside the bin. Thus, to separate the vertices in the bin from those outside, we might at most have to remove all the edges to all the separator nodes. But in each separation step, the size of the graph on which we recurse drops to at most $2/3$ the original size. Thus there can be at most $\log_{3/2} n$ iterations, and hence $k+1$ times that many separator vertices. Thus the total number of edges is $O(dk \log n)$. ■

The above theorem can be used to bound the treewidth of a graph. For example, for the n node hypercube has bisection width $n/2$, and degree $\log n$. Supposing the hypercube is s separable and has treewidth $TW(n)$ we know that $s(n) \leq TW(n) + 1$. Thus from the above theorem we know that the bisection width $n/2 = O((\log n)s(n) \log n) = O((\log n)TW(n) \log n)$. Or in other words, $TW(n) = \Omega(n/\log^2 n)$.

Can we match this? Somewhat. We can find a tree embedding of width $O(n/\sqrt{\log N})$ as follows. Do a BFS of the hypercube Q_r where $r = \log n$ from any node. There are $\binom{r}{i}$ nodes in level i. The decomposition tree is

contains the vertices $\{a, b, c, d\}$, and the leaves respectively contain $\{a, b, c, 1\}$, $\{b, c, d, 2\}$, $\{c, d, a, 3\}$ and $\{d, a, b, 4\}$. This has treewidth 3, but no 3-separator.

simply the path P_{r+1}, with nodes $0, \ldots, r$. We set V_i to contain all vertices in level $i, i-1$ of the BFS tree, for $i = 1$ to r. Thus the edge $(i-1, i)$ is the subtree for vertices in level $i-1$, whereas vertices in level r have node r of P_{r+1} as the subtree. Since edges in the BFS tree only go only from level i to level $i+1$ we have the overlap between subtrees as desired.

The maximum number of nodes placed in any tree node is $\binom{r}{i} + \binom{r}{i+1}$. Binomial coefficients peak in the middle, and hence this quantity is $O(\binom{r}{r/2}) = O(2^r/\sqrt{r}) = O(n/\sqrt{\log n})$.

The last connection...

Theorem 3 *If a n node graph is k separable, then it has treewidth at most $O(k \log n)$.*

We need the notion of a *separator tree* for the graph. This is a tree in which the root holds the separator of the entire graph. The subtrees at the root are the separator trees for the subgraphs resulting from the separation. The leaves hold single vertices which do not get included in any separator. The tree has height at most $\log_{3/2} n$ because at each stage the size of the subgraphs reduces to at most 2/3 the original.

Proof: We construct a decomposition (T, f) as follows. We choose T to be the same as the separator tree S. The subtree for a vertex is simply the subtree beneath where it appears in S. Each vertex in any node of S can have edges to only separator vertices contained in the path from that node to the root of S. Thus the intersection property is satisfied for (T, f). Each node u of T receives subtrees from all vertices in nodes in the path from u to the root in S. The path length is $O(\log n)$ and the number of vertices in any node of S at most $k+1$. ■

5.1 Planar Separator Theorem

Planar graph: that which can be drawn on the plane without crossings.

Thm: A planar graph has $O(\sqrt{n})$ separator.

Proof: Classical proof later. Today, a non standard proof.

Koebe's theorem: Every planar graph G=(V,E) can be drawn in the plane such that each vertex u is placed at the center of a circle C_u, The circles are non-overlapping but may touch, circles touch iff corresponding vertices are connected by an edge in G.

Works also on a sphere!

Centerpoint theorem: Let P be a collection of points in 3 dimensions. There exists a point c such that every plane through c partitions P such that at most 1/4 of the points are on each side.

Wont prove this either!

Circles can be found such that center of the sphere is a centerpoint of the vertices.

Given such a drawing, consider a random plane throug the center. Clearly it partitions the point set 1/4-3/4 by definition. It turns out that the expected number of circles intersected by the plane is $O(\sqrt{n})$. So such a plane must exist.

Consider a circle of radius r on the sphere of unit radius. What is the probability that a random plane touches it? The random plane may be chosen by picking a north pole at random, and taking its equator. Where should the north pole be so as to touch the circle? Within a band r of the equator defined by the circle center. So probability = band area/sphere surface area $= cr$. The expected number of intersections is thus $\sum_i cr_i$. But we know that areas of the circle must add up to at most 4π, thus $\sum_i r_i^2 = O(1)$. So we have to maximise the sum of r_is subject to the sum squares being a constant. When does this happen?

When all r_i are equal. Hence $r_i = 1/\sqrt{n}$. Thus the expected number of intersections, $O(\sum_i r_i) = O(n/\sqrt{n}) = O(\sqrt{n})$.

Exercises

1. Show that any tree can be split into disconnected forests T_1, T_2 with at most 2/3 the number of nodes as the original by removing one node.

2. Show that by removing $O(\log n)$ nodes an n node tree can be split into forests each with 1/2 the nodes.

3. Show that a graph with treewidth k and degree 3 can has bisection width $O(k \log n)$.

4. Suppose G contains the complete graph on r vertices, which may be considered to be numbered $\{1, \ldots, r\}$. Show that the treedecomposition of r must contain a node u such that $V_u \supseteq \{1, \ldots, r\}$.

5. Show that a every cycle of length 4 or more in a chordal graph must contain a chord, i.e. an edge connecting to non-adjacent vertices in the cycle. Hint: suppose there exists a cycle without such an edge – argue that no vertex in the cycle can be before the others in an elimination ordering.

6. Let G be a graph with tree decomposition (T, f). Suppose a, b, c are vertices in G such that $(a, b), (b, c), (c, a)$ are edges in G. Show that $f(a), f(b), f(c)$ must have at least common vertex t of T.