Algorithmic Discrete Mathematics 5. Exercise Sheet

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Mathematics
PD Dr. Ulf Lorenz
Dipl.-Math. David Meffert

SS 2012
13. and 14. Juni 2012
Version of June 20, 2012

Groupwork

Exercise G1 (Spanning trees I)
Let $G = (V, E)$ be a connected, undirected, weighted graph with weight function $w\colon E \to \mathbb{N}$. Prove the following statements about spanning trees:

(a) Any edge $e \in E$ that is not contained in any cycle of G is contained in every minimal spanning tree of G.

(b) Every graph G with pairwise disjoint weights on the edges contains a unique minimal spanning tree.

Solution:

(a) We prove it by contradiction, so assume that there exists a minimal spanning tree $S = (V, E')$ not containing $e = (u, v)$. Since S is connected by definition there is a path p from u to v only using edges in $E' \subseteq E$. We get a cycle C by adding e to the path p. By construction this is a cycle in V and contains e. This contradicts e being contained in no cycle of V.

(b) Again we use proof by contradiction, so assume there exist two different minimal spanning trees $S = (V, E_1)$ and $S' = (V, E_2)$. Let the edges $E_1 := \{e_1, \ldots, e_{n-1}\}$ and $E_2 := \{e'_1 \ldots e'_{n-1}\}$ be ordered increasingly. This means for $1 \le i < j \le n-1$ we have $w(e_i) < w(e_j)$ and $w(e'_i) < w(e'_j)$, because all edges have disjoint weights. Since $E_1 \ne E_2$ there exists a minimal $k \in \{1, \ldots, n-1\}$ with $e_k \ne e'_k$ and w.l.o.g. we assume $w(e_k) < w(e'_k)$. This means that e_k is not contained in E_2, because otherwise that would be a contradiction to E_2 being ordered with increasing weights. Now add that edge e_k to E_2 and get a graph $G' = (V, E_2 \cup \{e_k\}))$. Since S' is a tree the graph G' contains a cycle C. We want to eliminate another edge $e \ne e_k \in C$ to get another spanning tree. We distinguish two cases:
1.case: There exists and $e \ne e_k$ with $w(e) > w(e_k)$. Then resulting spanning tree has a smaller weight than the minimal spanning tree S'. This is a contradiction.
2.case: We have $w(e) \le w(e_k)$ for all $e \in C$ with $e \ne e_k$. Since all edges have disjoint weights we may conclude $w(e) < w(e_k)$ for all those edges. By minimality and definition of k all those edges e have to be contained in E_1 and since $e_k \in E_1$ also holds the spanning tree S contains a cycle. This is a contradiction, too.

Exercise G2 (Spanning trees II)
Let $G = (V, E)$ be a undirected, connected, weighted graph.

(a) Let (u, v) be an arbitrary edge of a minimal spanning tree of the graph G. Prove that there is a cut of G, such that (u, v) is a light crossing edge for that cut.

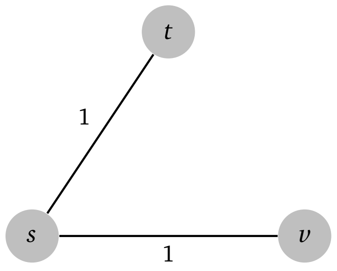
(b) Prove that, if for every cut in G there exists a unique light crossing edge for that cut, then G contains a unique minimal spanning tree. Also prove that the converse is not true in general.

Solution:

(a) Let S be a minimal spanning tree containing the edge (u, v). Let $(C, V - C)$ be a cut which is crossed by (u, v). The cut C could contain the vertex u and all vertices which are in the same connected component as u the subgraph $S \setminus \{(u, v)\}$. Assuming that (u, v) is no light crossing edge for that cut we get a another edge (x, y) with smaller weight than (u, v) in that cut. Then $S' = (S \setminus \{(u, v)\}) \cup \{(x, y)\}$ is also a spanning tree and it has a smaller sum of weights than S. This contradicts S being a minimal spanning tree.

(b) Assume that for every cut in G there is a unique light crossing edge for that cut. Now let S and S' be two minimal spanning trees of G. We want to proof that S and S' have the same edges and therefore are equal. Let (u, v) an arbitrary edge of S. By part (a) we get a cut $(C, V - C)$ and (u, v) is a light crossing edge of that cut. Since without any edges crossing this cut S' can't be a spanning tree there exists edges in S' crossing this cut. We claim that one of these edges is a light crossing edge for the cut C. Then by uniqueness of the light crossing edge for that cut (u, v) is also an element of S' and since $(u, v) \in S$ was arbitrarily chosen we get $S = S'$. So lets assume there is no light crossing edge for the cut C, which is contained in S'. We add (u, v) to S' and call the resulting graph G'. This graph contains a cycle containing a crossing edge $(x, y) \in S'$ and (u, v). Since we assumed (x, y) to have a bigger weight than (u, v) we can delete (x, y) from G' to get a spanning tree of G which has smaller weight than S'. This is a contradiction to S' being a minimal spanning tree for G.

A counterexample for the converse statement is the following graph:

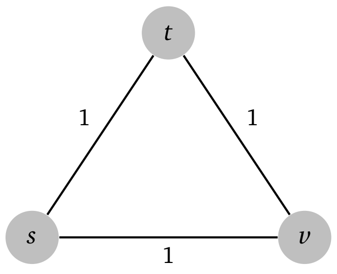


Of course this graph has a unique minmal spanning tree, but if we take then cut induced by the vertex set $C = \{s\}$ then the light crossing edges are not unique.

Exercise G3 (Spanning trees III)
Let $e = (i, j)$ be an edge of minimal weight in an undirected, connected, weighted graph $G = (V, E)$. Prove the existence of a minimal spanning tree containing that edge (i, j). Does every minimal spanning tree have to contain that edge?

Solution: Let $S = (V, E')$ be a minimal spanning tree of G. If e is contained in E' we are done. Otherwise look at the Graph $G' = (V, E' \cup \{e\})$. By construction this graph contains a cycle C. Removing any edges $e' \in C$ with $e' \neq e$ yields a new spanning tree S'. We claim that this tree is also minimal. Assume it is not. Then the edge e' we eliminated has to have smaller weight than e. This is a contradiction to e being an edge with minimal weight.

Not every minimal spanning tree has to contain the minimal edge e. Lets for example take $e = (s, t)$. Then $S =$



$(s, t, v; (s, v), (t, v)$ is a minimal spanning tree not containing e.

Homework

Exercise H13 (Finding anagrams) (10 points)
In this exercise we want to work with anagrams. Two words A and B are anagrams of each other, if they contain the same characters with same cardinality of these characters and without respecting the difference of small and capital characters. Examples are the words 'meat'and 'team', 'dog' and 'god' or 'plates' and 'staples'. An example for words not being anagrams could be 'mama' and 'beer'.

Now assume we have a dictionary containing many words and we have to find out which words are anagrams of each other. Construct an algorithm solving this problem. You can describe it pseudocode or just by words. The description should be sufficiently precise, such that a fellow student of yours would be able to implement the algorithm by just using you description. What is the worst case runtime of your algorithm? On which property (parameter) of the set of given words does the algorithms runtime also depend?

Solution: This is just the idea of the algorithm. Let n be the number of words we want to investigate and m be length of the longest word we have.

Step1: For each of the words we sort its characters in lexicographic order (head -> aedh, sheep -> eehps), but still save the original word and connected this information to each other, such that in the end we can still see what the original word was. This process needs runtime $O(n \cdot m \log m)$ by using a sorting algorithm like Heap-Sort for example.

Step2: Now we sort this permuted words in alphabetic order. While words like 'sheep' and 'deep' where far away from each other the original list, the words 'eehsp' and 'eedp' are now very close together. This steps takes a runtime of $= O(n \log n)$ by using Heap-Sort again.

Step3: In the last step we compare all the words which are directly after each other in the list we got in step2. If two words are the same we return the saved original words and say that they are anagrams of each other. So for example we startet with 'meat' and 'team'. Both of the words will become 'aemt' in the first step. In the second step they will be put directly after each other in the list and by the comparing in the third step we see that they are anagrams of each other. The runtime of the comparison step is at most $O(n \cdot m)$.

For the total runtime we get $O(n \cdot m \log m + n \log n + n \cdot m) = O(n \cdot m \log n)$ under the assumption that $m \leq n$. (which is a realistic assumption)

Exercise H14 (Trees and forests) (10 points)

(a) Prove that you can use an algorithm finding minimal spanning trees, to find a forest with maximal weight.

(b) Prove that conversely you can use an algorithm finding maximal forests, to find a minimal spanning tree, if the graph is connected.

Solution:

(a) Let $G = (V, E)$ be a graph with edge weights c_e. We construct a complete graph $K_n = (V, E_n)$ with the following weights c'_e:

$$\begin{aligned} c'_e &:= -c_e \quad \text{for } e \in E \text{ with } c_e \geq 0 \\ c'_e &:= 1 \quad \text{for } e \in E_n \setminus \{e \in E \mid c_e \geq 0\}. \end{aligned}$$

Let S be a minimal spanning tree in K_n with the weights C'_e then we claim $F = S \setminus \{e \in S \mid c'_e = 1\}$ is a maximal forest in G. Since F is a subgraph of a tree it is a Forest. Additionally we have $c_{e_1} < c_{e_2}$, iff $c'_{e_1} = -c_{e_1} > -c_{e_2} = c'_{e_2}$. So for c_e being minimal in K_n it is maximal in G. This proves the maximality of the forest.

(b) Let $G = (V, E)$ be a graph with edge weights c_e. Set $M := \max\{|c_e|, e \in E\} + 1$. Now set $c'_e = M - c_e$ and determine a maximal forest F with these new weights. Since by contruction all weights c'_e are positive and we assumed G to be connected F is connected, too. If it would not be connected we can add a edge without getting a cycle and since the weights c'_e of that edge is positive the weight of that forest would be bigger. Hence F is a spanning tree. The minimality of this forest(tree) with weights c_e follows from the relation $c_{e_1} > c_{e_2}$ iff $c'_{e_1} = M - c_{e_1} < M - c_{e_2} = c'_{e_2}$.

Exercise H15 (Updates on spanning trees) (10 points)

Let S be a minimal spanning tree of an undirected, connected, weighted graph $G = (V, E)$. Find an algorithm which determines a minimal spanning tree in the new graph G' arising from the following modifications:

(a) Delete an edge $(i, j) \in E$. Assume that the resulting graph $G' = (V, E')$ is still connnected.

(b) Add an edge $(i, j) \in E$. Assume that G is not a complete graph.

Prove the correctness of your algorithms and determine its runtimes.

Solution:

(a) If the deleted edge did not belong to S we are done. If $(i, j) \in S$ this induces a cut $(C, V - C)$ where C and $V - C$ are the resulting connected components of the graph $S \setminus \{(i, j)\}$. By looking at all the crossing edges of that cut and taking the one with minimal weight we get the new minimal spanning tree. There exists at least one crossing edge since G' is still connected. The worst case runtime for this procedure is $O(m)$ with m being the number of edges.

(b) Let P be the unique path from i to j and $(k, l) \in E$ an edge with maximal weight on that path. If we have $c_{ij} \geq c_{kl}$ then S is still a minimal spanning tree. Otherwise we get the new minimal spanning tree be substituting the edge (k, l) by (i, j). Since the length of P is smaller than n this method needs at most runtime $O(n)$ with n being the number of vertices.