Prof. Friedrich Eisenbrand
Dr. Yuri Faenza

Discussions from: September 24, 2013

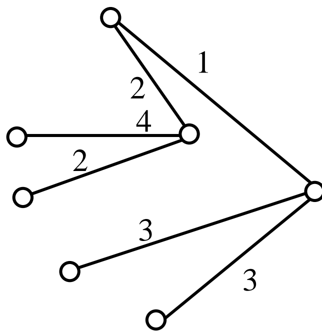
Combinatorial Optimization

Fall 2013

Assignment Sheet 1

Exercise 1
Take the complete graph with three vertices and all edges of cost -1.

Exercise 2



Exercise 3
This is a special case of next exercise.

Exercise 4
There are several ways to show this. The proof we give does not rely on the knowledge of the algorithm for computing a minimum spanning tree. Assume that $w \in \mathbb{Z}$ (a similar argument works for generic w). Let $e_1, \dots, e_m$ be any ordering of the edges of the graph, and let $w'(e_i) = w(e_i) + 2^{-i}$. Since

$$2^i > 2^1 + \cdots + 2^{i-1} \quad \text{for each } i \in \mathbb{N}, \tag{1}$$

we have that $w(T) < w'(T) < w(T) + 1$ for each subgraph T of G. In particular, if $w(T') < w'(T')$ for each pair of trees T, T', then $w'(T) < w(T) + 1 \le w(T) < w'(T')$ (recall that w is integral). Even more in particular, a minimum spanning tree in (G, w') is one of the minimum spanning trees in (G, w).

Now we show that no two spanning tree of G (actually, the argument works for any two subgraphs) have the same cost wrt w'. In fact, let T and T' be two subgraphs of G and consider the fractional part of their cost, which we denote respectively by $\mu(T)$ and $\mu(T')$. Note that $\mu(T) = \sum_{i: e_i \in E(T)} 2^{-i}$ and similarly for T'. Let e_i be the first edge where they differ, and γ the fractional contribution given to T (and T') by the edges among $e_1, \dots e_{i-1}$ that they

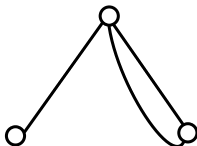
have. Wlog let $e_i \in E(T) \setminus E(T')$. Then $\mu(T) \geq \gamma + 2^{-i}$ and, from (1), $\mu(T') < \gamma + 2^{-i}$, proving the claim. In particular, there is a unique spanning tree of minimum cost.

Exercise 5
We show the contrapositive. Let $G = (V, E)$ be a connected graph with a cycle. From the greedy algorithm for the minimum spanning tree problem, we now that there exists a tree $T = (V, E')$ with $E' \subseteq E$. Note that $|E'| < |E|$, else T has a cycle, a contradiction. Using the "only if" part, we know that $|E| > |E'| = |V| - 1$, and the thesis follows.

Exercise 6
Let u and v be the two nodes of odd degree. Construct a *walk* (i.e. we take a sequence of adjacent nodes, and we are allowed to pass multiple times of the same node) starting from u, and picking each time an edge that has not been taken before in the walk. Since the number of edges is finite, we eventually arrive at some node w where no edge available to extend the walk. Suppose first $w = u$. Since we started from u, this implies that the walk used an even number of edges incident on u, but this contradicts the fact that the degree of u is odd and the path cannot be extended. If $w \neq u, v$, then the walk used an an odd number of edges incident on w, again a contradiction. It must then be $w = v$. It is intuitive and routine to prove (but you may want to check it explicitly) that, given a walk connecting two nodes, there also exists a path connecting them.

Exercise 7
In a simple graph all nodes have degree between 0 and $n-1$. Note that the events "there is a node of degree 0" and "there is a node of degree $n-1$" are mutually exclusive. Hence, suppose there is no node of degree 0 (the other case goes similarly). Then by the pigeon-hole principle (where the nodes are the pigeons and the values of degrees from 1 to $n-1$ are the holes), there exist two nodes with the same degree.

If the edges can appear multiple times, then the following is a counterexample.



Exercise 8

FYI, the algorithm ALGO is known as *Prim's algorithm*, while the GREEDY one seen in class is usually called *Kruskal's algorithm*.

(b). A general comment on the algorithm: at each repetition of the `WHILE`, the connected component U of v_1 is increased by exactly one.

We first show that the graph $T = (V, \tilde{E})$ obtained at the end of the algorithm is a tree. Let $u \in V$. We show that there is a path between u and v in T. In fact, let U be the connected

component of v_1 in T. If $U = V$ the claim follows, so suppose not. Since G is connected, there exists a path between $u \in U$ and $z \in V \setminus U$. Then each edge of this path connecting U to $V \setminus U$ is candidate for being taken by the algorithm in the last step, a contradiction. Hence T is connected. We now argue that T has no cycle. Suppose it had, and let $e_1, \ldots, e_k$ be the edges of the cycle, taken in this order. When the edge $e_k = wz$ was added, w, z were already in the same connected component of $(V, \tilde{E})$, hence the algorithm could not have taken it, a contradiction.

We now argue that it is a tree of minimum cost. Sort the edges of T as $e_1, e_2, \ldots, e_{n-1}$, so that e_i is inserted in $\tilde{E}$ at the i-th iteration of the algorithm, and let T' be a minimum spanning tree of G such that the minimum i such that $e_i \notin E(T')$ is maximized. Let $e_i = uz$ be the first edge of $E(T) \setminus E(T')$ – one must exists, else we are done. Call U_i the set U at at the beginning of the i-th iteration, and assume wlog that $u \in U_i$ and $z \notin U_i$. Since T' is a tree, there exists a path between z and each node of U_i. Take such a path, and let $e = wx$ be an edge of this path with $w \notin U_i$ and $x \in U_i$. Note that $e \neq e_1, \ldots, e_i$, hence $w(e) \geq w(e_i)$, else ALGO would choose e at the i-th iteration. We claim that $T'' = T' \setminus \{e\} \cup \{e_i\}$ is a minimum spanning tree. Clearly $w(T'') = w(T') - w(e) + w(e_i) \leq w(T')$, and $E(T'') = E(T') = |V| - 1$. Moreover, pick any two nodes $p, q \in V$, and consider the path connecting them in T'. If this path does not cross e, then it is still a path in T''. If conversely it does, then we can obtain a walk between p and q by replacing the edge e with the path between w and z in T'' (which exists by construction and is the same we have in T'), the edge zu, and the path between u and x. Hence T'' is a minimum spanning tree that contains $e_1, \ldots, e_i$, a contradiction.

(c). We present an $O(|V|^2)$ implementation using the adjacency matrix A, with edge costs in each entry (and $+\infty$ if no edge appears). At each repetition of the `WHILE` we keep an array L of length $O(|V|)$ that contains, for each $v \in V \setminus U$, the cheapest edge (or any of them in case of ties) between U and v, or $+\infty$ if none. We can select the next edge to insert in $\tilde{E}$ (hence, the next node to join U) in $O(|V|)$ by search through L. We can also update L in time $O(|V|)$. In fact, let u be the node inserted in U: for each $v \in V \setminus U$ we just compare the cost of $L[v]$ with the cost $A[v][u]$, and keep the minimum. Since $O(|V|)$ repetitions are needed, the thesis follows.

(d). We now show an implementation in time $O(|E| \log |V|)$ using the adjacency list. This requires some basic knowledge on binary heap that can be obtained e.g. by looking at the Wikipedia page on this topic. We recall here some definitions and facts, specializing them to our context:

- A *binary heap* is a data structure organized as a binary tree where each node has an associated number, called *key*, and the value of each son is greater or equal than the value of the parent;
- Given a set of n entries with keys, they can be organized in a binary heap in time $O(n \log n)$.
- One can remove the minimum element of the heap (the root) and restore the heap structure in time $O(\log n)$.
- One can update the key of a node and restore the heap structure in time $O(\log n)$.

The implementation is as follows: we keep the nodes of $V \setminus U$ stored as a heap, where the key is the edge of minimum cost connecting $V \setminus U$ to U. Consider a generic step of the algorithm, where an edge vu is added to $\tilde{E}$ and consequently a node u is added to U. We eliminate the root, restore the heap structure, and then for each node adjacent to u check if the newly introduced edge decreases the value of the heap and, in this case, update the key and restore the heap structure. Let us analyze the complexity. Sorting the heap requires $O(|V|\log|V|)$. Each elimination of the root plus restoring of the heap structure requires time $O(\log|V|)$, and it is performed at most $|V|$ times. Each update of the key plus restoring of the heap structure requires $\log|V|$, and in total it is performed at most once per edge, giving the total complexity of $O(|E|\log|V|)$.