1: Minimum Spanning Trees

The following algorithm can be shown to compute a minimum spanning tree (MST) of a connected undirected graph $G = (V, E)$, where each edge has a real valued weight. Initially let each node of V be a tree of size one. Let the set of trees be denoted $\mathcal{T}$. Repeatedly pick an arbitrary tree $t \in \mathcal{T}$ and grow t by joining t with a tree $t' \in \mathcal{T}$ that is connected to t by an edge e of minimum weight. The edge e will be part of the computed MST. This generic algorithm forms the basis for several efficient algorithms for computing a MST.

a) Adapt the algorithm such that it for a planar graph $G = (V, E)$ finds a MST in time $O(|V|)$. The algorithm should proceed in $O(\log |V|)$ phases. After each phase the algorithm should contract each subtree to a single node and remove multiple edges and self loops.

Let $G = (V, E)$ be a connected undirected graph, with $n = |V|$ and $m = |E|$, where each edge has a real valued weight. In the following we consider a variant of the generic algorithm that works in a sequence of phases. In each phase small subtrees of a MST are identified. After a phase the subtrees found in the phase are contracted to single nodes. For each phase we have two parameters s_i and k_i, for $i \geq 0$. The number of trees at the start of phase i is denoted s_i, where $s_0 = n$. The parameter $k_i = 2^{2m/s_i}$ is denoted the heap limit of phase i.

In phase i, initially all nodes are subtrees of size one and are unmarked. An unmarked subtree t is grown until one of the following three conditions is true, in which case t is marked.

- t is adjacent to $> k_i$ nodes outside t.
- t has been joined with a marked tree, or
- t is not adjacent to another tree,

A phase ends when all nodes have been marked.

b) Show how each phase of the algorithm can be implemented to take time $O(s_i \log k_i + m)$ using Fibonacci heaps.

c) Show that the number of trees after a phase is at most $2m_i/k_i$, and that the heap limit $k_{i+1} \geq 2^{k_i}$.

d) Argue that there is at most $O(\log^* n)$ iterations, and that the running time of the algorithm becomes $O(m \log^* n)$, where $\log^* n = \min\{i \mid \log^{(i)} n \leq 1\}$, $\log^{(1)} = \log n$ and $\log^{(i+1)} n = \log \log^{(i)} n$, for $i > 1$.

2: Augmenting Paths of Maximum Bottleneck Capacity

Show how to calculate in time $O(|E|+|V| \log |V|)$ an augmenting path of maximum bottleneck capacity in a graph $G = (V, E)$ with capacity c.

3: Ghostbusters

A group of n Ghostbusters is battling n ghosts. Each Ghostbuster is armed with a proton pack, which shoots a stream at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The Ghostbusters decide upon the following strategy. They will pair off with the ghosts, forming n Ghostbuster-ghost pairs, and then simultaneously each Ghostbuster will shoot a stream at his or her chosen ghost. As we all know, it is *very* dangerous to let streams cross, and so the Ghostbusters must choose pairings for which no streams will cross.

Assume that the position of each Ghostbuster and each ghost is a fixed point in the plane and that no three positions are on the same line.

a) Argue that there exists a line passing through one Ghostbuster and one ghost such the number of Ghostbusters on one side of the line equals the number of ghosts on the same side. Describe how to find such a line in $O(n \log n)$ time.

b) Give an $O(n^2 \log n)$ time algorithm to pair Ghostbusters with ghosts in such a way that no streams cross.

4: Self Adjusting Trees *

In this exercise we consider search trees that do not maintain any explicit balance information. Each node is represented by a record consisting of three fields: an element from some totally ordered universe, a pointer to the left child, and a pointer to the right child (non existing children are represented by nil). The elements are maintained in inorder. The crucial operation on a tree is the following operation:

– splay(i, T) reorganize T such that the node containing element i becomes the root. If $i \notin T$ then either $\max\{k \in T | k < i\}$ or $\min\{k \in T | k > i\}$ becomes the root.

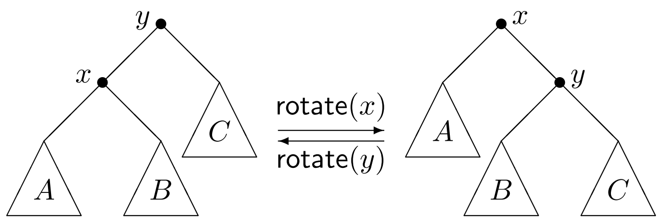
Using the splay operation we can now implement the following three operations:

– Member(i, T) determines if $i \in T$ by performing splay(i, T) and returning true if and only if the new root of T contains i.

– Insert(i, T) inserts element i in T as follows. First perform splay(i, T). Assume the new root v contains $\max\{k \in T | k < i\}$ (the case $\min\{k \in T | k > i\}$ is handled symmetrically). Create a new root with element i, left child v, and right child being the right child of v. The right child of v becomes nil and the left child of v remains unchanged.

– Delete(i, T) deletes $i \in T$ as follows. First perform splay(i, T) such that the root contains i. Let T_L and T_R be respectively the subtree rooted at the left child and the right child of the root. Remove the root and let $T = T_L$. Perform splay(i, T), i.e. the right child of the root of T is nil, and make T_R the right child of the root of T.

The splay operation is implemented by first performing an ordinary search in T for i. The node found is then moved to the root by performing a sequence of rotations:



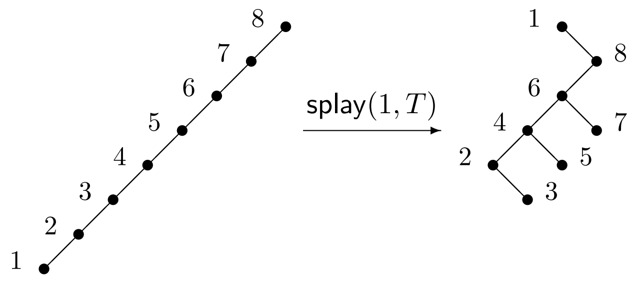
More specifically, let x be the node found. The node x is move to the root by iterating the following steps:

(i) If x has a parent but no grandparent, apply $\mathsf{rotate}(x)$.

(ii) If x has a parent y and a grandparent, and both x and y are left children (or both right children), then apply $\mathsf{rotate}(y)$ followed by $\mathsf{rotate}(x)$.

(iii) If x has a parent y and a grandparent, and x is a left child and y a right child (or x right child and y a left child), then apply $\mathsf{rotate}(x)$ followed by $\mathsf{rotate}(x)$.

The example below illustrates the effect of applying $\mathsf{splay}(1, T)$ to the tree to the left.



Let $|T(v)|$ denote the number of nodes in the subtree $T(v)$ rooted at a node v. Let $\mu(v) = \log|T(v)|$. To do an amortized analysis of the operations we define the potential of a tree by the function

$$\Phi(T) = \left\lceil \sum_{v \in T} \mu(v) \right\rceil .$$

a) Show that each application of (ii) and (iii) to a node x implies a change in potential $\Delta\Phi(T) \leq 3(\mu(x') - \mu(x)) - 1$, where x' refers to x after applying the two rotations. Prove first that $2\log(a+b) - \log a - \log b > 1$, for all $a, b > 0$.

b) Show that each splay operation takes amortized time $O(\log n)$.

c) Show that each of the operations Insert, Delete and Member takes amortized time $O(\log n)$.