Algorithms Example Sheet 5: Further Reading

(optional) Faster algorithms for the MST

Exercise 5.F.1 [Boruvka's algorithm] One of the earliest known algorithms for computing the MST is Boruvka's algorithm. This works by starting with all single vertices and no edges and then for each vertex pairing it up with the smallest incident edge. Then it contracts the connected components and repeats the same thing.

(a) Prove that Boruvka's algorithm finds the MST.

(b) Explain why the number of connected components in each iteration are at most half as many as in the previous iteration. Deduce that Boruvka's algorithm needs $\mathcal{O}(\log V)$ iterations.

(c) Explain how you would implement each step in $\mathcal{O}(E)$ time.

(d) Deduce that the running time of Boruvka's algorithm is $\mathcal{O}(E \log V)$.

Exercise 5.F.2 [Yao's MST algorithm] (++) In this problem we will illustrate some of the ideas behind Yao's $\mathcal{O}(E \log\log V)$ time algorithm. The essential idea behind this algorithm is to note that if we implement Boruvka's algorithm we scan all the edges in the graph in each pass, and avoiding this repetition should get us an improvement. Assume that the graph G is connected and hence $E \geq V - 1$.

(a) Suppose for each vertex, the edges adjacent to that vertex are stored in non-decreasing order of weights. Show a slight variant of Boruvka's algorithm that runs in time $\mathcal{O}(E + V \log V)$ time.

(b) (*k-partial sorting*) Given a parameter k and a list of N numbers, give an $\mathcal{O}(N \log k)$ time algorithm that partitions this list into k groups $g_1, g_2, \ldots, g_k$ of size at most $\lceil N/k \rceil$ each such that all elements in g_i are smaller than those in g_{i+1} for each i.

(c) Adapt your algorithm from the first part above to handle the case where the edges adjacent to each vertex are not completely sorted but only k-partially-sorted. Ideally, your run-time should be $\mathcal{O}\left(\frac{m}{k} \log V + V \log V\right)$.

(d) Use the two parts above (setting $k = \log V$), preceded by some additional rounds of Boruvka, to give an $\mathcal{O}(m \log\log V)$ time MST algorithm.

[Source: CMU Adv. Algo Ex 2]

Project 1 [Fredman-Tarjan $\mathcal{O}(E \log^* V)$ algorithm] (+) Read section 1.3 from these notes and answer the following questions:

(a) Define the *iterated logarithm* $\log^* n$ for $n \in \mathbb{N}$. (see wikipedia)

(b) (optional) Plot this function to confirm that this stays small even for large values of n.

(c) Describe a single iteration of the Fredman-Tarjan algorithm.

(d) What is the time complexity of each round?

(e) Prove that the number of trees in each iteration is at most $2m/K$.

(f) Describe the choice of K_i in each iteration. How does this lead to the overall $\mathcal{O}(E \log^* V)$ time complexity?

Project 2 [Fast second-best MST] Read about *lowest common ancestors* in a tree and how these can be computed in $\log(V)$-time using the technique known as *binary lifting*.

(a) Argue that two vertices u and v have a unique lowest-common ancestor.

(b) Explain the *binary lifting technique*.

(c) Argue about the time complexity of the binary lifting technique.

(d) Extend the binary lifting technique to find the minimum in the path between two vertices u and v.

(e) Use the extended binary lifting technique to find the second-best MST in time $\mathcal{O}(E \log V)$.

Maximum flow

Exercise 5.F.3 [Edmonds-Karp algorithm] (++) The Edmonds-Karp algorithm modifies Ford-Fulkerson algorithm by finding the augmenting path using a BFS instead of any other search method (such as DFS) and this turns out to drastically improve the time complexity for finding the maximum flow. Read p.728-730 in CLRS to find out how it achieves this.

Project 3 [Push-relabel] (++) Read section 26.4 in CLRS for how the *push-relabel* algorithm, finds the maximum flow in $\mathcal{O}(VE^2)$ time.

Project 4 [Relabel-to-front] (+++) Read section 26.5 in CLRS for how the *relabel-to-front* algorithm finds the maximum flow in $\mathcal{O}(V^3)$ time.