Combinatorial optimization for undergraduate students

Lecture note 2. Algorithm and complexity

Lecturer : O-joung Kwon
Spring, 2018

In Chapter 1, we observed the characterization of Eulerian graphs: a graph is Eulerian if and only if every vertex has even degree. This condition can be easily checked. But can we output a specific Eulerian trail?

In fact, the proof in the characterization theorem gives a hint for how to efficiently get a specific Eulerian trail.

We say that this kind of work is to turn a mathematical theorem into an 'algorithm'.

We will see an example of a graph algorithm and learn the concept of NP-hardness. (Have you heard about one of Millennium problems called 'P vs NP' problem?)

We want to test whether a given graph is connected. There are two ways.

1) Observe that a graph G is connected if and only if for every vertex partition (A, B) of G with $A \neq \emptyset$ and $B \neq \emptyset$, there is an edge between A and B. Based on it, we do :

- We consider all partitions (A, B) of the vertex set of G, and test whether there is an edge between A and B.

It takes $2^{|V(G)|}$ many procedures (number of possible A subsets is $2^{|V(G)|}$).

A tree is a connected graph having no cycles.

2) We construct a tree subgraph of G as follows.

(1) Choose an edge vw and set H as the subgraph on $\{v, w\}$.
(2) Choose a vertex z of degree 1 in H, and add all neighbors of z outside H, to H with edges connecting to z. We do this until extensions are no longer possible.
(3) If H contains all vertices of G, then we say that G is connected; otherwise, we say that G is not connected.

We observe that in all procedures, a vertex can be used as z in Step (2) only once. And each time, we need to look at all neighbors of z that are not in H. Thus, we can run this procedures in $|V(G)| \times |V(G)|$ many procedures.

Algorithm 2) is much more efficient than Algorithm 1), even though Algorithm 1) is simpler.

The complexity of algorithms (Chapter 2.5).

Time complexity is the function f, where $f(n)$ is the maximal number of steps need to solve a problem instance having input data with size n. For graphs, we will usually use $n = |V(G)|$ or $m = |E(G)|$.

Space complexity is defined analogously for the memory space.

In this lecture, we mostly focus on the time complexity. And the complexity is always measured for the worst possible cases for a given length of the input data.

For two functions $f, g : \mathbb{N} \to \mathbb{R}^+$, we write

- $f(n) = \mathcal{O}(g(n))$ if there is a constant $c > 0$ such that $f(n) \leq cg(n)$ for all sufficiently large n;
- $f(n) = \Omega(g(n))$ if there is a constant $c > 0$ such that $f(n) \geq cg(n)$ for all sufficiently large n;
- $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

Theorem 1. *Given an Eulerian graph, one can in polynomial time output an Eulerian trail.*