



# Project MSF

## Updates

- Updates related to this project will be posted here.
- 2019-11-02: Added the following to the section "Common Problems": The number of nodes  $n$  is small enough to fit in a 32 bit integer. However, it is still large enough that if you multiply  $n$  with another large number, you might encounter integer overflows. Depending on the used programming language, no warning will be displayed when this happens, but your result will be wrong.
- 2019-10-14: We slightly changed how kattis grades your submission:
  - Wrong Answer: Your result is not accurate enough (it is not within  $(1 \pm \epsilon)$  ).
  - Time Limit Exceeded: If your program runs longer than 3 seconds.
  - Accept: Your result is accurate enough. You will get a score between 0-10, based on how many nodes were requested. 10 is the best score. 0 the worst.
- 2019-10-11: (Assuming you try to implement [CS10]) Try testing your algorithm offline and check if the subroutine for estimating the number of connected components gives a correct result.
- 2019-10-11: Try testing your algorithm offline for graphs that consist of many small connected components, e.g. many small cycles of length 3, 6, 12 etc. This can help detecting if you have some off by one error when counting nodes or if you have a bug in generating  $P[X \geq k] = 1/k$ .
- 2019-10-11: The first test group on kattis tests only connected graphs, i.e. an algorithm for spanning trees should pass it. If you decide to implement [CS10], then maybe it helps if you first try to get it to work on the first test group, before extending the algorithm to spanning forests.
- 2019-09-19: I updated the [example file](https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1)  [\\_ \(https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1\)](https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1) that demonstrates the communication for querying nodes. Previously it did not read the input parameters *epsilon* and *maximum weight*.

## Instructions

- **Questions:** Most questions should be asked on the discussion board. For sensitive questions that need private communication, please send an email to *Jan van den Brand* <janvdb@kth.se>, *Kilian Risse* <kilianr@kth.se> and *Danupon Nanongkai* <danupon@gmail.com>. Start the title/subject of your email with [DD2440-MSF].
- You should also register for the course on Kattis. Login with your Kattis account, go to <https://kth.kattis.com/courses/DD2440/avalg19>  [\(https://kth.kattis.com/courses/DD2440/avalg19\)](https://kth.kattis.com/courses/DD2440/avalg19) there should be a link "I am a student taking this course and I want to register for it on Kattis."

## Project name: [Sublinear Algorithms: Minimum Spanning Forest](https://kth.kattis.com/problems/kth.avalg.spanningforest)

(<https://kth.kattis.com/problems/kth.avalg.spanningforest>)

The main task of this project is to develop and implement a sublinear algorithm for the minimum spanning forest problem (click the link for detailed problem description).

You will be graded based on your result on Kattis and your report & interview.

## Grade on the Kattis Score

You should submit your code to Kattis for automatic grading and evaluation.

Your grade for this project depends on the score of your submission:

- E: score 11-19
- D: score 20-39
- C: score 40-49
- B: score 50-59
- A: score 60


**Please include the submission id in your report!**

## Report


In your report, you should document your decisions and why you made them. The reasoning is allowed to be experimental. You do not have to prove the bounds of your algorithm, but you should argue why you chose certain values for your parameters. For example some experimental comparison, that shows what happens for different parameters, is a valid way to explain why you decided to choose the parameters in a certain way.

From the report it should be clear that you understand your own algorithm. You should explain why the algorithm works. It doesn't have to be a formal proof but you should explain the idea of the algorithm and why it is able to return the weight of a minimum spanning forest.

You should not just copy some text/proof from other sources, but it is fine if you can reproduce their idea in your own words. If your algorithm is based on some papers, then you *must* reference them. The same is true for every other source that helped you with the project. Cite your sources!

**Please include the submission id in your report!** Also make sure you are registered for the course on Kattis, so the TAs can see your submission. Login with your Kattis account, go to <https://kth.kattis.com/courses/DD2440/avalg19>  (<https://kth.kattis.com/courses/DD2440/avalg19>) there should be a link "I am a student taking this course and I want to register for it on Kattis."


## Tips for MSF

- The task is to compute the weight of a minimum spanning forest in  $o(|E|)$  time. Such algorithms are typically called sublinear algorithms, because the complexity is less than linear in the input size. [This paper \[CS10\]](http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf)  (<http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf>) has a nice summary of

techniques used in sublinear algorithms. Specifically section 3.2 should be very helpful.

- The above mentioned paper contains a section about minimum spanning **trees**. Maybe you can adapt the presented algorithm to **forests**?
- Note that the analysis in [CS10] [↗](http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf) (<http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf>) is a *worst-case upper bound*. Do not be surprised, if you need to read much less of the input graph, than [CS10] [↗](http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf) (<http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf>) claims.
- The algorithm for spanning trees, presented in [CS10] [↗](http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf) (<http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf>), has complexity  $O(W^3/\epsilon^2 \log n)$ . The paper also cites a much more complicated algorithm [CRT05] [↗](https://www.cs.princeton.edu/~chazelle/pubs/mstapprox.pdf) (<https://www.cs.princeton.edu/~chazelle/pubs/mstapprox.pdf>) which manages a faster  $O(W / \epsilon^2)$  time bound. Passing this project does **not** require this complicated algorithm. So when starting this project, feel free to focus on the slower but simpler  $O(W^3/\epsilon^2 \log n)$  algorithm presented in [CS10] [↗](http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf) (<http://www.wisdom.weizmann.ac.il/~oded/PTW/sublin.pdf>).
- Programming language: You can write your programs in [any language that Kattis accepts](https://kth.kattis.com/help) [↗](https://kth.kattis.com/help) (<https://kth.kattis.com/help>). The main task of this project is to get an algorithm that does not read the entire graph. The actual time required by your algorithm does not matter, so feel free to use slower languages.

## Common problems:

- The start of this project can be a bit tricky, because your submission has to communicate with Kattis. To help you get started with this project I have [uploaded an example that shows how your program can query nodes from Kattis](https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1) [↗](https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1) (<https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1>)  (<https://kth.instructure.com/courses/12427/files/2181592/download?wrap=1>).
- The function `range(n)` in Python 2.7 will create a list of  $n$  integers. It is VERY slow for large  $n$ . Something like `random.sample(range(n), s)` will break your program when the graph has  $n > 10^9$  nodes.
- Python (probably also some other languages) does not immediately output strings via `print`. This can cause your program to fail with "time-limit-exceeded" because both Kattis and your program are waiting for each other to output something. You can fix this issue by calling `sys.stdout.flush()` after `print`. (You can see this for instance in the above linked example program.)
- The number of nodes  $n$  is small enough to fit in a 32 bit integer. However, it is still large enough that if you multiply  $n$  with another large number, you might encounter integer overflows. Depending on the used programming language, no warning will be displayed when this happens, but your result will be wrong.