Labs in EDAF05 Algorithms, data structures, and complexity

Jonas Skeppstedt

March 29, 2023

1 Background

These labs have been developed over the years for the EDAF05 course at Lund University by many individuals, and we are grateful to Erik Amirell-Eklöf, Thore Husfeldt, Björn Magnusson, Jonatan Nilsson, Hans Wennborg, Lars Åström, and others. Lab 4 four is new 2023.

2 How to do the labs

- To make lab presentation more efficient, you must have a Discord username that shows your real name (but to ask questions, you can use any username).
- You can use any implementation language you wish.
- First download the Tresorit directory you got an email about.
- You should do the labs on a Unix machine (or Ubuntu app on Windows)
- If you are unfamiliar to using the terminal then you can check out appendix A in the book (and tresorit).

3 Lab 1 Stable Matching

3.1 Goals of the labs

- Implementing an algorithm.
- Debugging your code.
- Structuring your code in a logical fashion.
- Parsing messy input.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

3.2 Problem formulation

You are given each student's and company's preference lists. Given these lists find a stable matching, meaning that there does not exist two student-company-pairs $(s_1, c_1), (s_2, c_2)$ such that c_2 is higher ranked on s_1's preference list than c_1 and simultaneously s_1 is higher ranked on c_2's preference list than s_2 (thus that both s_1 and c_2 would prefer to leave their current matching and create a new pair).

3.3 Input

Input is given through standard in (not from a file). To run your program with some specific input, write for instance

- `python3 solution_file < input_file`
- `./a.out < input_file`
- `java solution_file < input_file`

When you think your program works, you should use the `check_solution.sh` such as one of the following commands:

```
sh check_solution.sh java solution_file
sh check_solution.sh python3 solution_file.py
sh check_solution.sh ./a.out
```

The first line of the input contains a single integer N $(1 \leq N \leq 3000)$, the number of students and companies. Then follows the preference lists. In total $(N+1) \cdot 2N$ integers are given on one or more lines. These numbers are structured in $2N$ blocks of $N+1$ integers each. Each block contains the description of one student or company preference list – first an integer $1 \leq i \leq N$, the index of the person, then N distinct integers, the preference list of student or company i. These N integers are all different and between 1 and N, inclusive.

Note that each index (of a student or company) $i \in \{1, 2, ..., N\}$ appears twice – the first appearence is for company i, the second for student i.

3.4 Output

- `stdout` in C
- `cout` in C++.
- `print()` in Python
- `System.out.println()` in Java

The output should consist of exactly N rows. Row i should contain exactly one integer, the index of the student matched with company i.

3.5 Examination and Points of Discussion

To pass the lab make sure you:

- Have successfully implemented the algorithm with time complexity $O(n^2)$.
- Have neat and understandable code.
- Have sensible variable names. (if they are short, make a comments about their meaning)
- Have filled in the blanks in the report.
- Have run the `check_solution.sh` script, to validate your solution.

During the oral presentation you will discuss the following questions with the lab assistant:

- Why does your algorithm obtain a stable solution?
- Could there be other stable solutions? Do you have an example/proof of uniqueness?
- What is the time complexity and why?
- Are there any applications of the algorithm (as it is or in a slightly shifted version)?

3.6 Sample input and output

Input 1

```
2
1 1 2
2 2 1
1 1 2
2 2 1
```

Output 1

```
1
2
```

Input 2

```
2
1 1 2
1 2 1
2 2 1
2 1 2
```

Output 2

```
2
1
```

4 Lab 2 Word ladders

4.1 Introduction

Finding the shortest path between two places is a quite common task. It might be the solution to finding a GPS-route or a tool to see how related different subjects are1. Therefore efficient algorithms for this task is of course quite crucial.

4.2 Goals of the labs

- Implementing BFS.
- Debugging your code.
- Structuring your code in a logical fashion.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

4.3 Problem formulation

We construct a graph where each node represents a five-letter word (the words are not necessarily in the English dictionary, but consist only of lowercase English letters, a–z). Furthermore we draw an (directed) arc from u to v if all of the last four letters in u are present in v (if there is more than one of a specific letter among the last four letters in u, then at least the same number has to be present in v in order for us to draw the edge). For example there is an edge from `"hello"` to `"lolem"` but not the other way around. There is both an edge from `"there"` to `"where"` and the other way around. There is not an edge from `"there"` to `"retch"` since `"e"` is only present once in `"retch"`. You will be asked to answer a series of queries. For each query you will be given two words, the ”starting”-word and the ”ending”-word. The task is to find the length of the shortest path from the ”starting” to the ”ending” word for each query.

4.4 Input

The first row of the input consists two integers N, Q with $1 \leq N \leq 5 \cdot 10^3$ and $1 \leq Q \leq 5 \cdot 10^3$, the number of words we consider and the number of queries. Then follows N lines containing one five-letter word each. After this Q lines follow containing two space-separated five-letter words each. For each of these lines answer the query.

4.5 Output

For each query output a single line with the answer. If there exists a path from the ”starting” to the ”ending” word, print the length of the shortest path. Otherwise print `"Impossible"` Note that you have to write exactly `"Impossible"` to get the verdict *Correct* of the checker.

4.6 Examination and Points of Discussion

To pass the lab make sure you have:

- Have successfully implemented the algorithm with the correct time complexity.
- Have neat and understandable code.
- Have sensible variable names and comments explaining them.
- Have filled in the blanks in the report.
- Have run the `check_solution.sh` script to validate your solution.

During the oral presentation you will discuss the follwoing questions with the lab assistant:

- How do you represent the problem as a graph, and how is the graph built?
- If one were to perform backtracking (find the path along which we went), how would that be done?
- What is the time complexity, and more importantly why?
- Is it possible to solve the problem with DFS: why or why not?
- Can you think of any applications of this? Of BFS/DFS in general?

4.7 Sample input and output

Input

```
5 3
there
where
input
putin
hello
there where
putin input
hello putin
```

Output

```
1
1
Impossible
```

5 Lab 3 Making Friends

5.1 Introduction

Your are a consultant of a recently started making-friend-agency. Soon you are planning to host an event where the goal is to help everyone befriend everyone else at the event. It is widely known that the friend of a friend automatically becomes your friend. Therefore it is only of importance to connect all people, once there is a friendly path (a path of friendships) between two people they will soon become friends. Since you, as all consultants and more specifically all programmers, are lazy you want to minimize the effort you have to put in. Therefore all needed information about the participants have been collected and from this you can extract the compatability of two people. From this you have predicted how many minutes you would have to spend on introducing some pairs of people. Due to your laziness you have not predicted the number of minutes to introduce all pairs of people, only a subset of the pairs, however it is guaranteed that there is a path of predicted pairs between all pairs of people (i.e. it is possible to construct a path between each pair of people). Now the goal is to minimize your amount of work in order to connect all people.

5.2 Goals of the labs

- Implementing an algorithm to solve Minimal Spanning Tree.
- Debugging your code.
- Structuring your code in a logical fashion.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

5.3 Problem formulation

You are given a number of people and a number of undirected weighted edges between them (the weight being the number of minutes you would have to spend in order for the pair to become friends). The task is to print the number of minutes you will have to spend (in total) in order to connect all people.

5.4 Input

The first line of the input consists of two integers, N, M, the number of people at the event and the number of pairs for which you have predicted the time it would take you to make them friends. It is guaranteed that $2 \leq N \leq 10^5$ and
$1 \leq M \leq min(N \cdot (N-1)/2, 3 \cdot 10^6)$.

Then follows M lines containing the description of each edge. Each of these lines contain three integers u, v, w, where $1 \leq u, v \leq N$ are the indices of the persons of the edge and $0 \leq w \leq 10^6$ is the weight of the edge, i.e. the number of minutes it would take you to make u and v friends. It is guaranteed that each pair (u, v) occurs at most once in the input.

5.5 Output

The output should contain a single integer on a single line, the total sum of the weights of the minimal spanning tree in the graph – i.e. the miminal total number of minutes you would have to spend on connecting all people.

5.6 Examination and Points of Discussion

To pass the lab make sure you have:

- Have successfully implemented the algorithm with the correct time complexity.
- Have neat and understandable code.
- Have reasonable variable names.
- Have filled in the blanks in the report.
- Have run the `check_solution.sh` script to validate your solution.

During the oral presentation you will discuss the follwoing questions with the lab assistant:

- Why does the algorithm you have implemented produce a minimal spanning tree?
- What is the time complexity, and more importantly why?
- What happens if one of the edges you have chosen to include collapses? Might there be any problems with that in real applications?
- Can you think of any real applications of MST? What would the requirements of a problem need to be in order for us to want MST as a solution?

5.7 Sample input and output

Input 1

```
3 2
1 2 3
2 3 7
```

Output 1

```
10
```

Input 2

```
4 4
1 2 1
2 3 4
3 4 5
1 3 7
```

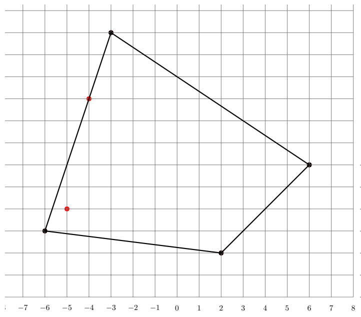
Output 2

```
10
```

6 Lab 4: Convex hull

6.1 Introduction

This lab is about implementing two convex hull algorithms, Graham scan and Preparata-Hong. In addition to input and correct output data, for the smaller inputs there are also pdf-files which show the correct solution, with black points as the convex hull and interior points red:



6.2 Goals of the labs

- Implementing the Graham scan algorithm
- Implementing a divide and conquer algorithm: Preparata-Hong
- Debugging your code for instance by creating a pdf file or an Octave plot to visualize your output.
- Structuring your code in a logical fashion.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

6.3 Problem formulation

Given a set of points, find the convex hull.

6.4 Input

The first line contains a dimension and the number of points. The dimension in the lab is always 2, and the number of points varies between 3 and $2,048,000$. The coordinates are mostly integers but some inputs have non-integer coordinates. We know that floating point arithmetic can cause rounding errors but what might be new is that this is also true for such ”trivial” tasks as just reading and writing floating point numbers in text form. Therefore, some programming languages, including C, support hexadecimal floating point numbers. They are identical to normal floating point numbers in memory and the CPU and use the same types, such `float` or `double`. Their purpose is only to achieve more accurate input and output.

Each line contains one point in the following format: x-coordinate as a hexadecimal floating point number, y-coordinate as a hexadecimal floating point number, a #, x-coordinate as a ”normal” floating point number, the y-coordinate as a ”normal” floating point number.

It is sufficient to ignore the hexadecimal floating point numbers, and just read the coordinates after the #.

6.5 Output

The output should be one line with the number of points, h, in the convex hull, and then h lines each with the x and y coordinates of one point. The sequence of points should be in the order produced by the Preparata-Hong algorithm as specified in the lectures (and book), such as:

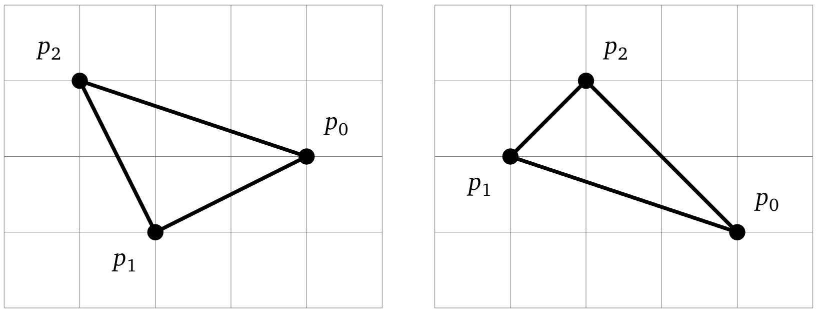


Figure 1: Example output sequences.

If all coordinates are integers, then the output should be integers, and if any coordinate is non-integer, all output coordinates should be printed with three decimals, such as for `data/10/10.0.correct`:

```
5
7.010 -8.097
-5.966 -5.564
-8.546 7.338
-3.227 9.884
3.975 8.898
```

6.6 Examination and Points of Discussion

To pass the lab make sure you have:

- Have successfully implemented the two algorithms with the correct time complexity.
- Have neat and understandable code.
- Have reasonable variable names.
- Have filled in the blanks in the report.
- Have run the `check_solution.sh` script to validate your two solutions.

During the oral presentation you will discuss the follwoing questions with the lab assistant:

- What is the time complexity, and more importantly why?
- Why should the α and β comparisons with γ be done in a specific order (depending on which of the four cases you have)?
- What happens if an α or β is infinity? Is that a valid numerical value in the language you use?
- How is `%` computed in the language you use? Is it guaranteed to be non-negative? That is not the case for ISO C, at least (but the sign *is* specified identically for C and C++). The reason we have this question is that most programmers sooner or later will experience an unpleasant surprise if they have not thought about this.

  Note that the pseudo code should work for any language.
- Would it be easier to parallellize Graham scan or Preparata-Hong?

6.7 Sample input and output

Input

```
2 8
0x1p+1 0x1p+2 # 2.00000000000000000000 4.00000000000000000000
-0x1.cp+2 -0x1p+2 # -7.00000000000000000000 -4.00000000000000000000
0x1.cp+2 0x1.8p+2 # 7.00000000000000000000 6.00000000000000000000
-0x1.8p+1 0x1p+0 # -3.00000000000000000000 1.00000000000000000000
-0x1p+0 0x1p+3 # -1.00000000000000000000 8.00000000000000000000
-0x1p+1 -0x1.cp+2 # -2.00000000000000000000 -7.00000000000000000000
0x0p+0 -0x1p+1 # 0.00000000000000000000 -2.00000000000000000000
-0x1p+0 0x1.8p+1 # -1.00000000000000000000 3.00000000000000000000
```

Output

```
 4
 7  6
-2  -7
-7  -4
-1  8
```

7 Lab 5 Gorilla

7.1 Introduction

How closely related are humans and gorillas? It is well known that out DNA-sequencies are very similar. But how do we actually know this? How do we even measure the similarity between two strings of DNA. And speaking of strings, how do we know how similar two strings are? Is it possible to use almost the same algorithm to align all kinds of strings? Of course not, guitar strings are aligned in a completely different way.

7.2 Goals of the labs

- Implementing a DP-algorithm.
- Debugging your code.
- Structuring your code in a logical fashion.
- Parsing a bit messy input.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

7.3 Problem formulation

You are given two strings, which can be viewed as words or DNA-sequences. Furthermore you are given a matrix with the cost of aligning each pair of letters, i.e. the gain of aligning ”A” with ”B” and so on. Given these strings find an optimal alignment, such that the total gain of aligning the strings in maximized.

When aligning the two strings you are allowed to insert certain ”*”-characters in one or both of the strings at as many positions in the string as you like. Each such insertion can be done to a cost of -4. No letters in either of the strings can be changed, moved or removed – the only allowed modification is to insert ”*”.

7.4 Input

The first line contains a number of space-separated characters, $c_1, ..., c_k$ – the characters that will be used in the strings. Then follows k lines with k space-separated integers where the jth integer on the ith row is the cost of aligning c_i and c_j. Then follows one line with an integer $Q(1 \leq Q \leq 10)$, the number of queries that you will solve. Then follows Q lines, each describing one query. Each of these lines contain two space-separated strings, the strings that should be aligned with maximal gain. It is guaranteed that each string in all queries has size at most 3500.

7.5 Output

The output should contain exactly one line per query. On each of these lines two strings should be given, the strings from the input possibly with some ”*” inserted. The strings have to be given in the same order as in the input.

7.6 Examination and Points of Discussion

To pass the lab make sure you have:

- Have successfully implemented the algorithm with the correct time complexity.
- Have neat and understandable code.
- Have reasonable variable names.
- Have filled in the blanks in the report.
- Have run the `check_solution.sh` script to validate your solution.

During the oral presentation you will discuss the follwoing questions with the lab assistant:

- Is your solution recursive or iterative?
- What is the time complexity, and more importantly why?
- What would the time complexity of a recursive solution without cache be?
- Can you think of any applications of this type of string alignment?
- What could the costs represent in the applications?

7.7 Sample input and output

Input

```
  A  B  C
  2  0 -1
  0  3  1
-1  1  3
2
AABC ABC
ABA ACA
```

Output

```
AABC *ABC
ABA ACA
```

8 Lab 6: Railway planning

8.1 Introduction

The railway system was heavily oversized during the end of the Cold War and the party secretary from Minsk has been tasked with liquidate money from the budget by removing some routes. Still the minister of transport demands that it should be possible to transport C people a day from Minsk to Lund, since that is the number of students that want to take part in the incredible course in Algorithms. Luckily, as help, you have a map over all routes. Furthermore you have constructed a list over the routes that would be most profitable to remove.

8.2 Goals of the lab

- Implementing a Network-Flow-algorithm.
- Debugging your code.
- Structuring your code in a logical fashion.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

8.3 Problem formulation

You are given the structure of the railway system, in the form of nodes (cities) and edges (routes connecting the cities). For each edge you will be given the capacity (the number of students it can carry). Then you will be given a plan for which routes to remove in a specific order – in which you need to remove the routes if possible. If you cannot (which you cannot if the total maximal flow from Minsk to Lund is less than C) remove a route you stop your plan and instead you start implementing the plan. Your task is to answer how many of the routes on your list you can remove and what the maximal flow of students from Minsk to Lund will be after removing these routes.

8.4 Input

The first line consists of four integers N, M, C, P, the number of nodes, the number of edges, the number of students to transfer from Minsk (node 0) to Lund (node $N-1$) and the number of routes in your plan. It is guaranteed that $2 \leq N \leq 1000, 1 \leq M \leq N(N-1), 0 \leq P \leq M$ and that the capacity of the network before any routes are removed is at least C. Then follows M lines where the ith line (0 indexed) consists of three integers u_i, v_i, c_i, where $0 \leq u_i, v_i < N$ are the nodes and c_i is the capacity of the route, $1 \leq c_i \leq 100$. Note that students can travel both from u_i to v_i and from v_i to u_i (the graph is undirected) but in total at most c_i students. It is guaranteed that the each pair of nodes occurs at most once in the input and that $u_i \neq v_i$. Then follows P lines with one integer each, where the ith line contains the index of the ith route you want to remove. Note that you have to remove them in exactly this order as they are given and it is guaranteed that all routes you want to remove are unique in the input.

8.5 Output

The output should contain one line with two space-separated integers x and f, where x is the number of routes you can remove (while still having a maximal flow of at least C) and f is the maximal flow from node 0 to node $N-1$ after removing these routes.

8.6 Examination and Points of Discussion

To pass the lab make sure you have:

- Have successfully implemented the algorithm with the correct time complexity.
- Have neat and understandable code.
- Have reasonable variable names.
- Have filled in the blanks in the report.
- Have run the `check_solutions.sh` script to validate your solution.

During the oral presentation you will discuss the follwoing questions with the lab assistant:

- What is the time complexity, and more importantly why?
- Which other (well-known) algorithmic problems can be solved using Network-Flow?
- If the capacities of the edges are very large, how can one get a different (better) time complexity?

8.7 Sample input and output

Input

```
3 3 10 3
0 1 10
0 2 10
1 2 10
0
2
1
```

Output

```
2 10
```