Labs in EDAF05 Algorithms, data structures, and complexity

Jonas Skeppstedt

April 4, 2025

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.

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 i.e. macOS, Linux or the 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 Word ladders

3.1 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.

3.2 Goals of the labs

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

3.3 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.

3.4 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.

3.5 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.

3.6 Examination and Points of Discussion

To pass the lab make sure you have:

- Understanding of how to achieve a time complexity of $O(n+m)$ of BFS.
- Have successfully implemented the BFS algorithm with the correct time complexity (but no specific constraint on constructing the graph).
- 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?

3.7 Sample input and output

Input

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

Output

```
1
1
Impossible
```

4 Lab 2 Hash tables

The purpose of this lab is to learn how to implement three kinds of hash tables:

- separate chaining,
- open addressing with linear probing and at least one of:
  - *delete* key or
  - *move*.
- open addressing with quadratic probing and *delete* key.

Study the file `reference.py` which is a program that uses Python's standard hash table, i.e. the dictionary. The program uses words as keys and the number of times they appear as values. It reads one word from each input line and either counts it or tries to delete it from the dictionary.

You should be able to test it with a command similar to:

```
time ./check_solution.sh python3 reference.py
```

You can use any language and feel free to use a standard hash table to

- check that your main program works, and
- compare the efficiency of it compared to your solutions.

5 Requirements

- No memory leaks are permitted, i.e. in C and other high performance languages such as C++, you need to deallocate all allocated memory yourself. Check with `valgrind`.

6 Assignments

1. Start with translating `reference.py` to the language you will use, or if that is Python, modify it so it can use your hash tables instead.
2. Then create a function that can compute a hash value for a string. If your language has a standard function for that, you can use it. If you use C or C++, a popular hash function used in compilers was invented by Peter J. Weinberger at Bell Labs. See below. It is OK to use for example Java's `hashCode` or any open source hash function implementation for the language you use.

```
#include <stddef.h>
#include <stdint.h>

/* hashpjw from the Dragon book. */
size_t hashpjw(void* sp)
{
        char*           id = sp;
        char*           p;
        uint32_t        h;
        uint32_t        g;

        h = 0;

        for (p = id; *p; ++p) {
                h = (h << 4) + *p;
                if ((g = h & 0xf0000000)) {
                        h ^= g >> 24;
                        h ^= g;
                }
        }

        return h;
}
```

a faster alternative is the hash function from the Musl C library (slightly adapted here).

```
#include <stddef.h>
size_t keyhash(void *k)
{
        unsigned char *p = (void *)k;
        size_t h = 0;

        while (*p)
                h = 31*h + *p++;
        return h;
}
```

3. The easiest to implement is separate chaining so start with that. Start with an array with one element and adjust its size as needed, i.e. make the array bigger and smaller depending on the number of key-value pairs in your hash table.

4. What are suitable minimum and maximum numbers of key-value pairs (i.e.

list nodes) compared to the size of the array?

5. If your implementation does not voluntarily reduce the size of the array, force it to in order to test that it works!

6. Then implement linear probing. Measure the effects of different maximum α-values (see book or slides) on the running time.

7. Then implement quadratic probing. Again, measure the effects of different maximum α-values (see book or slides) on the running time. Does it still ”work” despite having $\alpha > 0.5$? What conclusions can you draw about the danger of ”just testing”?

8. Which implementation is most efficient? Summarize your findings in a table (include `reference.py`).

9. What do you think takes the most time in your implementations? (any answer is correct if you think that)

10. Two time consuming operations are division and remainder. In C/C++ integer remainder `a % b` is computed as:

    ```
    int     q = a / b;
    int     r = a - q * b;
    ```

    In hash tables which don’t need the number of array elements, `n`, to be a prime number, it is possible to select `n` so `i % n` can be computed much faster. Hint (1): what is the binary representation of $2^k - 1$? Hint (2): what does the operator << do? Hint (3): what does the operator & do? Hints (2) and (3) are valid also for Python. Investigate if this approach can affect the performance of your separate chaining and linear probing implementations. Possibly update your table.

11. **Optional:** Compare the performance of both linear probing alternatives. Can you notice any performance difference?

7 Presentation

Show that your code works and explain it and what you found.

8 Sample input and output

Input 1

```
addis
addi
```

```
mflr
std
```

Output 1

```
addi 1
```

Input 2

```
addis
addi
mflr
std
stdu
std
mr
li
li
std
```

Output 2

```
std 3
```

9 Lab 3 Making Friends

9.1 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.

9.2 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.

9.3 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.

9.4 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.

9.5 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.

9.6 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?

9.7 Sample input and output

Input 1

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

Output 1

```
10
```

```
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
```

10 Lab 4 Closest pair

10.1 Introduction

It is now spring and what would be a better activity than to have a water gun fight? Thinks everyone else. Who would even like to be hit by water though?, you think. Instead you, godlike as you are, take a step up above the petty human beings and observe this crazy game (from a cloud high above, also known as a balcony with a great view). Shortly you get bored out of you mind and decide to create a betting game with your friend. After having observed the game for a while you think you have figured out the rules. Each minute all players have to stop somewhere in the field where they play and shoot exactly one shot at someone else. As the guns are not made for shooting long distance everyone tries to hit the very closest person. Before everyone shoot you and your friend will play a game. You win if you can name two people who both will shoot at each other, and otherwise you friend wins. You soon realize that this is an easy task, since of course the two people with the shortest distance (among all distances between pairs of players) between them obviously will shoot at each other. But as you also like to win you do not tell your friend, of course. So, what is the closest distance between any two players? If you can only answer this question, then the victory will be yours!

The goals of the lab are:

- Implementing a divide and conquer algorithm solving the Closest Pair problem.
- Debugging your code.
- Structuring your code in a logical fashion.
- Reason about correctness of your algorithm.
- Reason about upper bounds for time complexity.

10.2 Problem formulation

You are given the coordinates of the N people participating in the silly water pistol battle. You should calculate the minimal distance between two people on the battle ground. Input The first line of input consists of a single integer, $2 \leq N \leq 106$, the number of players. Then follow N lines, the i-th line containing two integers, the x- and y-coordinates of the i-th player. All coordinates have absolute value less than 109.

10.3 Output

The output should contain a single floating point number containing the shortest distance between two players on the battle field. The output should be a floating point number rounded to exactly 6 decimal digits.

10.4 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 descriptive 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:

- What is the time complexity, and more importantly why?
- Why is it sufficient to check a few points along the mid line?
- Draw a picture and show/describe when each distance is checked in your solution!
- When do you break the recursion and use brute force?

10.5 Sample Input 1

```
4
0 1
0 0
1 0
1 1
```

Correct output: `1.000000`

10.6 Sample Input 2

```
4
0 -1
-1 0
0 1
1 0
```

Correct output: `1.414214`

11 Lab 5 Gorilla

11.1 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.

11.2 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.

11.3 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 ”*”.

11.4 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.

11.5 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.

11.6 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?

11.7 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
```

12 Lab 6: Railway planning

12.1 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.

12.2 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.

12.3 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.

12.4 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.

12.5 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.

12.6 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?

12.7 Sample input and output

Input

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

Output

```
2 10
```