GENOME-SCALE ALGORITHM DESIGN
by Veli Mäkinen, Djamal Belazzougui, Fabio Cunial and Alexandru I. Tomescu
Cambridge University Press, 2015
www.genome-scale.info

Exercises for Chapter 2. Algorithm design

2.1 Consider the $\mathtt{fail}(\cdot)$ function of the Morris–Pratt (MP) algorithm. We should devise a linear-time algorithm to compute it on the pattern to conclude the linear-time exact pattern matching algorithm. Show that one can modify the same MP algorithm so that on inputs $P = p_1 p_2 \cdots p_m$ and $T = p_2 p_3 \cdots p_m \#^m$, where $\#^m$ denotes a string of m concatenated endmarkers $\#$, the values $\mathtt{fail}(2)$, $\mathtt{fail}(3)$, $\ldots$, $\mathtt{fail}(m)$ can be stored on the fly before they need to be accessed.

2.2 The Knuth–Morris–Pratt (KMP) algorithm is a variant of the MP algorithm with optimized $\mathtt{fail}(\cdot)$ function: $\mathtt{fail}(i) = i'$, where i' is largest such that $p_1 p_2 \cdots p_{i'} = p_{i-i'+1} p_{i-i'+2} \cdots p_i$, $i' < i$, and $p_{i'+1} \neq p_{i+1}$. This last condition makes the difference from the original definition. Assume you have the $\mathtt{fail}(\cdot)$ function values computed with the original definition. Show how to update these values in linear time to satisfy the KMP optimization.

2.3 Generalize KMP for solving the *multiple pattern matching* problem, where one is given a set of patterns rather than only one as in the exact string matching problem. The goal is to scan T in linear time so as to find exact occurrences of any pattern in the given set. *Hint.* Store the patterns in a tree structure, so that common prefixes of patterns share the same subpath. Extend $\mathtt{fail}(\cdot)$ to the positions of the paths in the tree. Observe that unlike in KMP, the running time of the approach depends on the alphabet size σ. Can you obtain scanning time $O(n \log \sigma)$? Can you build the required tree data structure in $O(M \log \sigma)$ time, where M is the total length of the patterns? On top of the $O(n \log \sigma)$ time for scanning T, can you output all the occurrences of all patterns in linear time in the output size?

2.4 Show that a certificate for the Hamiltonian path problem can be checked in time $O(n)$ (where n is the number of vertices) assuming an adjacency representation of the graph that uses $O(n^2)$ bits. *Hint.* Use a table of n integers that counts the number of occurrences of the vertices in the given certificate.

2.5 Suppose that we can afford to use no more than $O(m)$ space to represent the adjacency list. Show that a certificate for the Hamiltonian path can now be checked in time $O(n \log n)$.

2.6 Find out how bit-manipulation routines are implemented in your favorite programming language. We visualize below binary representations of integers with the most-significant bit first. You might find useful the following examples of these operations:

- *left-shift*: $\mathsf{0000000000101001} << \mathsf{2} = \mathsf{0000000010100100}$,
- *right-shift*: $\mathsf{0000000010100100} >> \mathsf{5} = \mathsf{0000000000000101}$,
- *logical or*: $\mathsf{0000000000101001} \mid \mathsf{1000001000001001} = \mathsf{1000001000101001}$,

- *logical and*: $\texttt{0000000000101001} \,\&\, \texttt{1000001000001001} = \texttt{0000000000001001}$,
- *exclusive or*: $\texttt{0000000000101001} \oplus \texttt{1000001000001001} = \texttt{1000001000100000}$,
- *complement*: $\sim\texttt{0000000000101001} = \texttt{1111111111010110}$,
- *addition*: $\texttt{0000000000101001} + \texttt{0000000000100001} = \texttt{0000000001001010}$, and
- *subtraction*: $\texttt{0000000000101001} - \texttt{0000000000100001} = \texttt{000000000001000}$, $\texttt{000000000001000} - \texttt{000000000000001} = \texttt{000000000000111}$.

These examples use 16-bit variables (note the overflow). Show two different ways to implement a function $\texttt{mask}(B, d)$ that converts the d most significant bits of a variable to zero. For example, $\texttt{mask}(\texttt{1000001000001001}, 7) = \texttt{0000000000001001}$.

2.7 Implement with your favorite programming language a fixed-length bit-field array. For example, using C++ you can allocate with

```
A=new unsigned[(n*k)/w+1]
```

an array occupying roughly $n \cdot k$ bits, where w is the size of the computer word (unsigned variable) in bits and $k < w$. You should provide operations `setField(A,i,x)` and `x=getField(A,i)` to store and retrieve integer x from A, for x whose binary representation occupies at most k bits.

2.8 Implement using the above fixed-length bit-field array an $O(n \log n)$-bit representation of a node-labeled static tree, supporting navigation from the root to the children.

2.9 Recall how stack, queue, and *deque* work. Implement them using your favorite programming language using doubly-linked lists.

2.10 Given an undirected graph G, a subset $S \subseteq V(G)$ is called an *independent set* if no edge exists between the vertices of S. In the independent-set problem we are given an undirected graph G and an integer k and are asked whether G contains an independent set of size k. Show that the Independent set problem is NP-complete.

2.11 A Boolean formula $f(x_1, \ldots, x_n)$ is in *3-CNF form* if it can be written as

$$f(x_1, \ldots, x_n) = c_1 \wedge \cdots \wedge c_m,$$

where each c_i is $y_{i,1} \vee y_{i,2} \vee y_{i,3}$, and each $y_{i,j}$ equals x_k or $\neg x_k$, for some $k \in \{1, \ldots, n\}$ (with $y_{i,1}$, $y_{i,2}$, $y_{i,3}$ all distinct). The subformulas c_i are called *clauses*, and the subformulas $y_{i,j}$ are called *literals*. The following problem, called 3-SAT, is known to be NP-complete. Given a Boolean formula $f(x_1, \ldots, x_n)$ in 3-CNF form, decide whether there exist $\alpha_1, \ldots, \alpha_n \in \{0, 1\}$ such that $f(\alpha_1, \ldots, \alpha_n)$ is true (such values α_i are called a *satisfying truth assignment*).

Consider as "new" problem the clique problem from Example 2.3. Show that clique is NP-complete by constructing a reduction from 3-SAT. *Hint.* Given a 3-CNF Boolean formula

$$f(x_1, \ldots, x_n) = (y_{1,1} \vee y_{1,2} \vee y_{1,3}) \wedge \cdots \wedge (y_{m,1} \vee y_{m,2} \vee y_{m,3}),$$

construct the graph G_f as follows (see Figure 1 for an example):

- for every $y_{i,j}$, $i \in \{1, \ldots, m\}$, $j \in \{1, 2, 3\}$, add a vertex $y_{i,j}$ to G_f;
- for every $y_{i_1,j}$ and $y_{i_2,k}$ with $i_1 \neq i_2$ and $y_{i_1,j} \neq \neg y_{i_1,k}$, add the edge $(y_{i_1,j}, y_{i_2,k})$.

Show that f has a satisfying assignment if and only if G_f has a clique of size m.

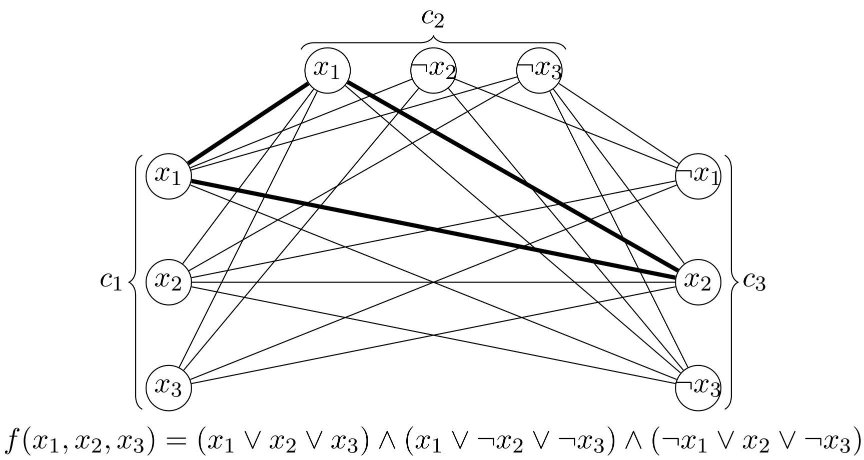

Figure 1: A reduction of the 3-SAT problem to the clique problem. A clique in G_f is highlighted; this induces either one of the truth assignments $(x_1, x_2, x_3) = (1, 1, 0)$ or $(x_1, x_2, x_3) = (1, 1, 1)$.

Additional exercises not in the book

2.12 In Exercise 2.11 above we have reduced the "old" problem 3-SAT to the "new" problem clique. Devise an opposite reduction, that is, show that the clique problem can be reduced in polynomial time to the 3-SAT problem. *Hint.* Suppose we are given a graph G on n vertices $v_1, \ldots, v_n$, and we are asked whether G contains a clique with k vertices. For every vertex v_i and for every $j \in \{1, \ldots, k\}$, introduce a Boolean variable $x_{i,j}$ with the meaning "vertex v_i is the jth vertex of the clique of size k". What is the corresponding Boolean formula on the variables $x_{i,j}$?