Questions on Combinatorial Algorithms

Course Participants

Winter term 2014/15

2015-02-04

This is a collection of “exam” questions on the material discussed in the course “Kombinatorische Algorithmen”. We have ranked them roughly by the so-called level of competence they require:

$\star$ – **reproduction** You should be able to find the answers to these questions directly in the provided material.

$\star\star$ – **assessment & application** These are questions that ask you to do something with or discuss the material. The answers will not be in the text(s) explicitly, but it should be clear what to do once you have read and understood the sources.

$\star\star\star$ – **transfer** These questions go beyond the material and ask you to build on its foundations with creative, independent thought.

Note that this “scale” does not necessarily correlate with increasing hardness; in particular, there are easy transfer questions. Nevertheless, you should be able to answer most of the $\star$ and $\star\star$ questions and some of the $\star\star\star$ questions before you take your exam. Happy thinking!

Warning: These questions are (for the most part) taken as-is from student submissions (modulo translation). Some of them may be unclear or even ill-posed – such issues are for you to detect and overcome!

String Matching

- What are the prefix- and suffix-function? $\star$
- Give the definition of the prefix-function. Explain how to compute it recursively. $\star$
- What is a random algorithm? $\star$
- Let $P = abaybabaxa$. Determine Π_P: $\star\star$

q	1	2	3	4	5	6	7	8	9	10
$\Pi_P(q)$										

- Let $P = \square\square\square\square abcababa\square\square\square$ with some unknown letters ($\square$). Use the given values to calculate $\Pi_P(11)$ and $\Pi_P(12)$. $\star\star$

q	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\Pi_P(q)$						4			4	5	?	?			

  Bonus: Prove that such a string does not exist.

- In our algorithm for calculating prefix-function Π_P we have the statement $\star\star$

  ```
  k := Pi[k]
  ```

  which may be executed many times (since it is in a loop). However, the array `Pi` is only filled with reasonable values up to a specific point (larger indices may contain arbitrary values). How do we assure that we only read valid values?

- Use the suffix function σ_P to calculate the value of $\Pi_P(q)$. $\star\star$
- Develop a linear-time algorithm that finds all occurrences of multiple spaces, tabs and line breaks. $\star\star\star$
- Develop an algorithm that finds all palindromes[1] in a given text $T \in \Sigma^*$ and is as efficient as possible. $\star\star\star$
- Let $P \in \Sigma^m$ and Π_P the corresponding prefix-function. Prove or disprove: $\star\star\star$

  $$\forall q \leq m.\ |\Pi_P^*(q)| \leq m\ .$$

- What is a best-case input for the algorithm that computes Π_P [9, p253], and what is its runtime on this input? $\star\star\star$
- Develop an algorithm that solves the *Context-Free Substring Matching* problem (defined similarly to Regular Substring Matching from Problem 3 for L context-free). $\star\star\star$

String Matching Automaton

- Give the definition of the SMA. $\star$
- Given Π_P, can we build the SMA for P more efficiently? $\star$
- Construct the SMA for the given “needle”. $\star\star$
- Construct the SMA for finding $P = ananas$ in German text. You may ignore capitalisation. $\star\star$
- Construct the SMA for $P = abbaababb$. Explain how you came up with the transition function. $\star\star$
- Construct the SMA for $P = huhuhuu$ and $\Sigma = \{h, u\}$. Which states does it traverse on text $T = huuhuhuhuhuuuhu$? $\star\star$
- Which property of the states in an SMA ensures that it detects overlapping matches? $\star\star$

[1] A word $w \in \Sigma^*$ is a palindrome if $w = w^R$.

Knuth-Morris-Pratt

- Which information does KMP use that the naive algorithm ignores? ⋆
- Which information does KMP use that the SMA does not, and how? ⋆
- How do we determine the automaton used in the KMP algorithm? ⋆
- What is the relationship between the SMA and the KMP algorithm? ⋆⋆
- Explain the essential difference between compute the SMA and the KMP algorithm. How does it influence the runtimes of the two procedures? ⋆⋆⋆
- If we wanted to exclude overlapping matches, how would we have to adapt KMP? ⋆⋆⋆

  Does the new algorithm find the *maximum* number of non-overlapping matches?
- Discuss whether and how the KMP algorithm can be parallellised. ⋆⋆⋆
- Discuss whether and how the KMP algorithm can be used for approximate string search (only $k < |P|$ symbols must match P). ⋆⋆⋆

  Does the answer change if P can have gaps?
- Is the implementation of KMP by Sedgewick and Wayne [11] optimal with respect to memory usage? ⋆⋆⋆
- Are the naive resp. the KMP algorithm suitable for string matching with *infinite* alphabets (e. g. $\Sigma = \mathbb{N}$)? ⋆⋆⋆

  If yes, does the comparative analysis of Nebel [9, p263] still work out? Does in particular $\mathsf{KMP}/\mathsf{NAIVE} \to 1$ for $c \to \infty$?

Boyer-Moore(-Horspool)

- Which heuristics does the BM algorithm use? ⋆
- Explain the bad-character and good-suffix heuristic, respectively. ⋆
- Why does the b-c-heuristic not miss any admissable shifts? ⋆
- How much memory does the BM algorithm use? ⋆⋆
- Compare the algorithms by BM and KMP. ⋆⋆
- Compute the b-c-heuristic function λ for *abrakadabra* and alphabet $\{a, \dots, z\}$. ⋆⋆
- Find a family of best-case inputs for the BM algorithm and determine the corresponding runtime. Which role do each of the two heuristics play in this case? ⋆⋆⋆
- What is the difference between the b-c-heuristic of the BM algorithm and the (sole) heuristic used by the BMH algorithm? Discuss possible implications regarding the runtimes of the two algorithms. ⋆⋆
- Implement the BMH algorithm! ⋆⋆⋆
- Is it possible to combine the advantages of the algorithms by BM and KMP? ⋆⋆⋆
- Suppose we wanted to allow wildcards in the search pattern. How would we have to change the BM algorithm? ⋆⋆⋆

Karp-Rabin

- How many false positives do we expect during a run of the KR algorithm? ⋆

Aho-Corasick

- Construct the search-word tree[2] for *abccbcabaa*. ⋆⋆
- Is it possible to combine the ideas from BM and AC? ⋆⋆⋆

Suffix Trees

- Give the definition of simple suffix trees. ⋆
- Sketch the simple suffix tree for $T = abcddc$. ⋆⋆

String Compression

Complexity of Words

- What are vocabulary and eigenvocabulary? ⋆
- What is the definition of $B \Rightarrow S$? ⋆
- What is the difference between producibility $\Rightarrow$ and reproducibility $\rightarrow$? ⋆
- What characterizes an information lossless encoder? ⋆
- Determine the exhaustive history $E(\_)$ and LZ-complexity $c(\_)$ of ⋆⋆
  - 0110111100,
  - *ababbabc*,
  - 1010001001011000 and
  - *mississippi*,

  respectively.
- What is the *ergodicity* [cf. 8, p 78] of a sequence source? ⋆⋆
- Give a de Brujin sequence for $\Sigma = \{0, 1\}$ and $k = 2$ and show that it is indeed one! ⋆⋆
- Determine which relations $\circ \in \{\rightarrow, \nrightarrow, \Rightarrow, \nRightarrow\}$ hold between the three pairs of strings ⋆⋆
  - $abc \circ abcabcd$,
  - $aba \circ ababa$ and
  - $aab \circ aabcd$,

[2] Also *trie*.

respectively. Argue why one of the four possible combinations is impossible.

- Prove that ε is a base for every symbol in the alphabet Σ but for no word from Σ^+. ⋆⋆⋆
- Prove that any given word has a unique exhaustive history. ⋆⋆⋆
- Suppose $\Sigma = \{0, 1\}$. Define a measure of complexity $M : \Sigma^\star \to [0, 1]$ that is a normalised version of the LZ-complexity c, i. e. in particular $M(w) = 0$ for w with minimal $c(w)$ and $M(w) = 1$ for w with maximal $c(w)$ (among all strings of same length, respectively). ⋆⋆⋆

  Determine words w_1, w_2 and w_3 so that $M(w_1) = 0$, $M(w_2) = 0.5$, $M(w_3) = 1$.
- Develop an algorithm which calculates eigenfunction $g_s(i)$ [cf. 8, p 80] efficiently. ⋆⋆⋆

Lempel-Ziv '77

- Give an input which LZ77 makes longer (cf. no-free-lunch theorem). Does it depend on the choice of parameters? ⋆⋆

Lempel-Ziv '78

- Give a worst-case input for LZ78. ⋆⋆
- Give an input which LZ78 makes longer (cf. no-free-lunch theorem). Does it depend on the choice of parameters? ⋆⋆
- Describe how the LZ78 compression algorithm [12] can be implemented as *online* algorithm, i. e. the algorithm may read every input symbol only once. ⋆⋆⋆
- Which data structure is suitable for implementing the dictionary used during LZ78 decompression [12]? Explain how its properties regarding extendability, search method and search cost fit the usage profile at hand. ⋆⋆⋆

Lempel-Ziv-Welch

- What is LZW compression? ⋆
- Explain the "tricky situation" Sedgewick and Wayne mention [11, p 843] both with an example and in general terms. How do we get around it and why can we do this? ⋆
- Why use tries for LZW encoding? ⋆
- What is the essential difference between LZ78 and LZW? ⋆⋆
- Are the data structures we used for storing dictionaries in LZ (de)compression also a good choice for LZW? ⋆⋆
- Compute $LZW(\_)$ for ⋆⋆
  - $w_1 = \texttt{GIBOPIKEINOPIUMDENNOPIUMBRINGTOPIUM}$,

  - $w_2 =$ `Mississippi`,
  - $w_3 =$ `abaacaccabbaaaac`,
  - $w_4 =$ `isnuknsuik`,
  - $w_5 =$ `ABBBBABB` (assume 7-bit ASCII input and 8-bit codes).

  Decompress the results afterwards! Be sure to make clear what the content of the dictionary is in every step.

- Expand the LZW-compressed file $\star\star$

  $$\text{LZW}(w) = 4E\ 45\ 56\ 45\ 52\ 47\ 4F\ 4E\ 4E\ 41\ 47\ 49\ 83\ 55\ 55\ 50\ 80\ .$$

  Assuming that an ASCII character takes 7 bits and a codeword 8 bits, what is the compression ratio?

- Give an input which LZW makes longer (cf. no-free-lunch theorem). Does it depend on the choice of parameters? $\star\star$

- Let A be the number of bits for an input character and B the number of bits for a codeword. What is the worst (resp. best) compression rate LZW can produce for nonempty words? Give an example of such an input and calculate the rate. $\star\star$

- Does LZW require the input alphabet to be fixed and/or known? Is it different from LZ7(7/8) in this regard? $\star\star$

- What is a worst case w. r. t. compression rate if we don't clear the dictionary? $\star\star\star$

- Does LZW always compress better than LZ77? Prove your claim. $\star\star\star$

- Give words x and y so that $\star\star\star$
  - $|\,\text{LZ77}(x)| > |\,\text{LZW}(x)|$ and
  - $|\,\text{LZ77}(y)| > |\,\text{LZW}(y)|$.

- Why does LZ only store the codeword (cf. LZ77 and LZ78)? $\star\star\star$

- Which needs fewer code words on a fixed text, LZ or LZW? Has the other advantages that compensate? $\star\star\star$

- Suppose we have a very large file compressed with the LZW algorithm. Is it possible to expand just a part of this file without having to expand everything before that? What if we used compressed with LZ78 (or LZ77) compression? $\star\star\star$

- Assume we wanted to parallelise LZW; which problems would arise and (how) can we circumvent them? Do these changes hurt the compression rate and if, by how much? $\star\star\star$

- Assume we only allowed dictionaries of (finitely) bounded size. Which options to we have for dealing with a full dictionary? Which information does the algorithm need about the as yet unread part of the input? $\star\star\star$

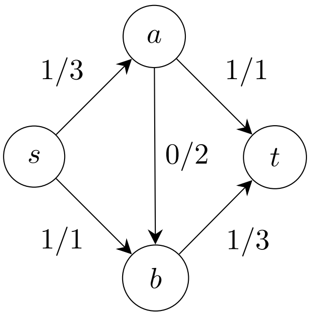
- In practice, how would you compress and bundle a *set* of files? Discuss different strategies and compare with experience you have had with packaging/compression tools. $\star\star\star$

Network Flows

- Define the basic notions network, s-t-network, flow, feasible flow, cut, flow conservation and capacity constraints. ⋆
- Define the Max-Flow problem. ⋆
- Name some algorithms which solve Max-Flow. ⋆
- What is a residual network? How do you construct it? ⋆
- State, explain and prove the Max-Flow-Min-Cut theorem (MFMC). ⋆
- Given network G, is there a flow f so that $G_f = G$ (up to isomorphism)? ⋆
- Give an arbitrary network with a (non-maximal) flow. Explain how increasing that flow works. ⋆⋆
- Explain how the MFMC leads to an algorithm(ic idea) for solving Max-Flow. ⋆⋆
- Explain why a path (from s to t) with free capacity is not a necessary condition for a sub-optimal flow. ⋆⋆
- Give the minimal cut and the residual network of the following networks with flows. ⋆⋆
  - Let $G = (V, E, c)$ and f be defined by adjacency matrix

$$A = \begin{pmatrix} - & (2,3) & (3,4) & - & - & - \\ - & - & (1,4) & - & (1,4) & - \\ - & - & - & (2,3) & (1,1) & - \\ - & - & - & - & (1,1) & (1,2) \\ - & - & - & - & - & (3,4) \\ - & - & - & - & - & - \end{pmatrix}$$

    where the nodes are named s, 1, 2, 3, 4 and t from left/top to right/bottom.
  - Let $G = (V, E, c)$ and f be defined by the following diagram:



- Formulate sufficient conditions for a graph whose maximum flow f^* has value $\mathrm{val}(f^*) = 0$. ⋆⋆
- Which problems can occur when solving Max-Flow on a network with non-integer capacities? ⋆⋆

- Discuss the advantages of using residual networks instead of “normal” networks. ⋆⋆
- Is it ever useful to send flow along a cycle? Does it ever hurt? Which of the algorithms you know can/will do so? ⋆⋆
- Name (at least) two real-world applications of the Max-Flow problem. ⋆⋆⋆
- Prove the following claim: ⋆⋆⋆

  > There is a network $G = (V, E, c)$ with a feasible s-t-flow f in G so that there is a node which has higher in-flow than the source s has out-flow, i. e.
  >
  > $$\exists\, v \in V.\; f\big(\delta^+(s)\big) < f\big(\delta^-(v)\big)\ .$$

- We know that ⋆⋆⋆

  1. Max-Flow is $\mathcal{P}$-complete (w. r. t. log-space reductions) [4], i. e. it is one of the “hardest” problems in $\mathcal{P}$,
  2. Max-Flow can be solved in time $\mathcal{O}(n^3)$ and
  3. there are arbitrarily hard problems in $\mathcal{P}$, i. e. for every c there are problems that require time $\Omega(n^c)$ to solve [5, Theorem 9].

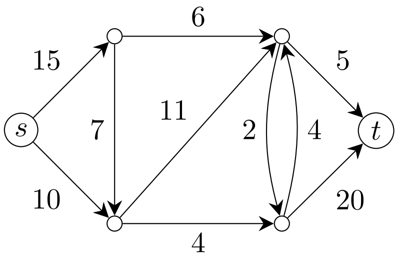
  How do these facts fit together?

Augmenting Paths

- How does the algorithm by Ford and Fulkerson (FF) work? ⋆
- Explain why FF works. In particular, why do we use the residual network (and not the original one)? ⋆
- What is the shortest-augmenting-path method? ⋆
- How does the algorithm by Edmonds and Karp (EK) work? ⋆
- What is the difference between FF and EK? Give an illustrating example. ⋆⋆
- Perform FF on the network in Figure 9.4 (upper left), Krumke and Noltemeier [7, p 202]. ⋆⋆
- Under which conditions is it possible for the generic algorithm (FF) to not output a maximum flow and why? Does the answer change if we always choose a shortest path (cf. EK)? ⋆⋆⋆
- Give a network on which FF does not terminate. ⋆⋆⋆
- Is FF *partially correct* for arbitrary capacities, i. e. is its output a maximal flow *if* it terminates? ⋆⋆⋆
- Explain why “oscillation graphs” [cf. 7, p 208] do not fool EK (but FF). ⋆⋆⋆

Push-Relabel

- What is a distance labeling? How does PR makes use of one? ⋆
- What is the Push-Relabel (PR) procedure? What is an intuition for the generic PR algorithm? ⋆
- Which (nontrivial) bounds on the number of relabels and pushes in PR have we obtained? Sketch the proofs. ⋆
- Explain why PR is correct, i. e. yields a maximum flow. ⋆
- What is the difference between generic PR and its specialised variants? ⋆
- Which advantages do the modifications highest-label and FIFO have, respectively? How so? ⋆
- Explain the proof of Korollar 9.30 in Krumke and Noltemeier [7, p 219] in detail. ⋆
- Does a distance labeling as defined by Krumke and Noltemeier [7, Def 9.28] fulfill the triangle inequality? ⋆⋆
- PR algorithms start by as much flow into the network as the edges incident to the source allow. If $(\{s, V \setminus \{s\})$ is not a minimal cut, this is too much; how do they ensure that no superfluous flow remains in the end? ⋆⋆ I tried.
- Explain why the analogy of a pipe system is helpful for understanding PR algorithms. Use it to explain how these algorithms work. ⋆⋆
- Apply the different versions of PR to the example from Krumke and Noltemeier [7, p 222]. What do you observe? ⋆⋆
- Apply the highest-label PR to the following network: ⋆⋆



- Sketch a residual network that has minimal sum of all distance labels (among all networks with the same number of nodes). ⋆⋆
- Analyse the data structure tailor-made for PR w. r. t. space usage. How close is it to the (asymptotic) optimum? ⋆⋆
- Analyse the space usage of PR. ⋆⋆
- Are there networks for which some other algorithm(ic idea) outperforms PR? ⋆⋆⋆
- Is it possible to adapt PR so that it solves Max-Flow on graphs with multiple sources and sinks? ⋆⋆⋆
- What is a worst-case instance for the generic PR? ⋆⋆⋆

- With what trick can we determine an $\mathcal{O}(n^2 \cdot \sqrt{m})$ bound on the runtime of highest-label PR? ⋆⋆⋆
- Assume the source can have incoming edges. Does PR still always find a maximum flow? ⋆⋆⋆
- The generic PR does prescribe neither which active node nor which of its edges to choose. We have already seen that clever node choices can improve the algorithm. Can we expect similar effects from rules for picking edges? Propose and discuss possible strategies! ⋆⋆⋆

Matchings

- What is the marriage problem and how do we solve it? ⋆
- Prove the Marriage theorem [7, Satz 9.59]. ⋆⋆
- What is a (perfect) matching? ⋆
- Prove König's theorem [7, Satz 9.60]. ⋆
- Prove Menger's theorem [7, Satz 9.61]. ⋆
- How does the Shrink-and-Update algorithm work? ⋆
- What are the definitions of alternating and M-augmenting paths, respectively? ⋆
- Why can there be no super nodes on an odd level? ⋆
- Why do we use capacities $M = n+1$ in the proof of König's theorem [7, Satz 9.60]? In particular, why not $M = 1$ as in the proof of the Marriage theorem [7, Satz 9.59]? ⋆⋆
- Given two matchings of different size, we can construct an M-augmenting path for the smaller one. Why, and how? ⋆⋆
- What is a sufficient criterion for a graph not having a perfect match? ⋆⋆
- In which way did the MFMC theorem aid us in our dealings with matchings? ⋆⋆
- How can we consolidate the following factoids? ⋆⋆
  - Finding a maximum matching is (essentially) the same as finding a minimum vertex cover.
  - We can find a maximum matching in time $\Theta(nm)$, i. e. polynomial time.
  - Vertex Cover is $\mathcal{NP}$-hart.
  - We do not know that $\mathcal{P} = \mathcal{NP}$.
- Explain the theorem of König [7, Satz 9.60] in term of duality. ⋆⋆⋆
- Give best- and worst-case examples of graphs w. r. t. the size of (minimal) vertex- and edge-covers, respectively. ⋆⋆⋆
- Are maximal matchings unique? If not, how can you determine the *number* of such in a given graph G? ⋆⋆⋆

- The algorithm for computing matchings of maximum cardinality aggregates multiple partial matchings into one larger one. Why is this correct, i. e. why is this union always a matching? ⋆⋆⋆
- How can we find weight-maximal matchings in bipartite graphs? ⋆⋆⋆
- We use alternating trees for finding perfect matchings. Can we use red-black trees instead? How, resp. why not? ⋆⋆⋆
- Can we use the M-augmenting path algorithm [7, p 279] for general graphs? Why (not)? ⋆⋆⋆
- Rewrite the algorithms for deciding whether graphs have perfect matchings by Krumke and Noltemeier [7, Algs. 10.2 and 10.4] to not use `goto` [1]. (ò_ó)

Symbolic Method

Basics

- What are generating functions useful for? What are their advantages? ⋆
- What is a *combinatorial class*? What do we require of its size function? ⋆
- What is the connection between a set of cycles and a set of permutations? ⋆
- Which type of generating function do we use for unlabeled and labeled objects, respectively? Give general examples for both. ⋆
- What is the difference between cross and star product? ⋆
- Determine the number of decimal numbers with n digits which have (at least) three subsequent digits 0 (at least once) – using generating functions, of course! ⋆⋆
- Model the Fibonacci sequence with the symbolic method. ⋆⋆
- Let $\mathcal{A}$ a class with counting sequence $a_n = \binom{c}{n}$ for some $c \in \mathbb{N}$. Give the closed form of the OGF of $\mathcal{A}$. ⋆⋆
- Give a combinatoric construction for ternary (search) trees and its OGF, with the number of leaves being the tree size. ⋆⋆
- Give a combinatoric construction for extended binary trees and its OGF, with the number of nodes being the tree size. ⋆⋆
- Where does the factor $\binom{n}{k}$ come from in the product rule for labeled classes [10, proof of Theorem 5.2] and why is it correct? ⋆⋆
- Explain what distinguishes a "consistent" relabeling. ⋆⋆
- Explain the analogy of the pointing operator Θ and differential calculus (cf. e. g. in the proof of Lemma 6.4 [2]). ⋆⋆
- Why are we not able to determine the average height of trees using the symbolic method? ⋆⋆⋆

- How many combinatoric constructions with n operations are there, assuming we allow only the operation from Theorem 5.1 [10]? ⋆⋆⋆
- Sedgewick and Flajolet [10] investigate unlabeled binary trees. Compare them two binary *search* trees, which are clearly labeled? What changes? ⋆⋆⋆
- How many labeled trees, i. e. simple cycle-free graphs, of size n are there? ⋆⋆⋆

Singularity Analysis

- How do we get from the radius of convergence of a generating function to an asymptotic of its sequence of coefficients? ⋆
- What is a Taylor series expansion and why do we use it (and not another kind of series expansion)? ⋆
- Given a generating function, how can we find its singular exponent $-\alpha$ (cf. Duchon et al. [2])? ⋆
- What does H-admissability [2] mean? ⋆
- Taylor series work as presented for OGF; what about EGF? ⋆⋆
- Derive the $\sim$-asymptotic of ⋆⋆

$$[z^n]\,\frac{1-z}{1-z-z^2}\;.$$

- Approximate the number of binary strings without 10, i. e. find a $\sim$-asymptotic for L_n, where ⋆⋆

$$L = \{w \in \{0,1\}^\star \mid 10 \notin \mathrm{substrings}(w)\}\;.$$

- Why are $\alpha \in -\mathbb{N}_0$ excluded from Theorem 5.5 [10] and its Corollary? ⋆⋆
- State possible reasons for being satisfied with an asymptotic of the coefficients of a generating function. ⋆⋆⋆
- Explain why the radius of convergence alone can not give you more than upper bounds on the coefficients [10, p 248]. ⋆⋆⋆

Random Generation

Top-Down

All questions in the top-down section relate to the article of Flajolet, Zimmerman, and Van Cutsem [3].

- What is the Θ operator? ⋆
- Summarize the basic steps towards the random generation of combinatoric structures as per Sections 1-3. ⋆

- State and explain the runtime of the simple sampling algorithm. ⋆
- The given random generation algorithm runs in time $\mathcal{O}(n^2)$. Which component dominates runtime? Can we improve it? ⋆
- Is the product $\cdot$ used here $\times$ (unlabelled product) or $\star$ (labelled product)? ⋆⋆
- Explain how the semantics of $\star$ extend to pointing, i. e. what $(\Theta\mathcal{A}) \star (\Theta\mathcal{B})$ is. ⋆⋆
- Explain the notion of *standard specification* and the standardization algorithm in your own words. ⋆⋆
- Explain why the standardisation algorithm is feasible, e. g. how $\mathcal{B} = \mathrm{SET}(\mathcal{A})$ is “equivalent” to $\Theta\mathcal{B} = B \cdot \Theta(\mathcal{A})$. ⋆⋆
- What is the advantage of *standard* specifications (over regular ones)? ⋆⋆
- Why have the conversions for the simple algorithm been chosen in the way they have been? Are there alternatives? ⋆⋆
- Explain step by step what happens when generating $\Theta\mathcal{Z}$ at $n = 1$. ⋆⋆
- What is the standard form of surjective mappings? ⋆⋆
- Show with an example that the product operator $\star$ on structures is not commutative. ⋆⋆
- Generate a random (labelled) non-plane tree, i. e. a member of ⋆⋆

  $$\mathcal{T} = \mathcal{Z} + \mathcal{Z} \star \mathrm{SET}(\mathcal{T})$$

  with $n = 6$ nodes using the algorithms from Theorem 2.2 and 3.1. Assume $(1/2, 3/7, 5/6, 2/9, 1/10, 9/19)^+$ as random number sequence.
- Give pseudo code (templates) for the postprocessing, i. e. recovery of a structure compatible with the original (non-standard) specification (cf. p 14). ⋆⋆
- Explain how boustrophedonic search saves time compared to the basic approach (cf. Theorem 4.1). ⋆⋆
- Explain how to compute the (missing) initial conditions (cf. p 13). ⋆⋆
- Theorem 3.1 mentions only *upper* runtime bounds. What can you say about *lower* bounds? ⋆⋆⋆
- Is the (non-standard) specification derived from the binary parse tree of the algorithm Random Generation (cf. proof of Theorem 3.1) unique? ⋆⋆⋆
- How does Section 1 integrate with the material covered in previous weeks? ⋆⋆⋆
- How would we have to change the sampling algorithms if there were two (or more) atoms with different probabilities? ⋆⋆⋆
- The presented procedure explicitly generates only *shapes* of strucures. The authors not that a labelling can be inserted by drawing a random permutation later. ⋆⋆⋆

  Does this not create problems with the (uniform) target distribution, in particular for structures such as cycles in which there are many labellings for the same object?
- Explain how to implement the algorithm(s) with *finite* word length resp. precision. ⋆⋆⋆
- What is $\Theta\,\Theta\mathcal{A}$? ⋆⋆⋆

Basic Boltzmann

- What is a Boltzmann sampler? ⋆
- Explain the unterlying probabilistic model [2, p 581]. In particular, why do the equations define a probability distribution over all objects? ⋆
- How does parameter x influence the variance of the (random) length n of the generated object? ⋆
- Explain the differences between the recursive and geometric sampler for sequences [2, p 587]. ⋆⋆
- Some generators have a recursive and a non-recursive variant. Why do the recursive ones terminate almost surely, i. e. with probability 1? And where do the non-recursive ones "hide" the recursion? ⋆⋆
- What are the main differences between Boltzmann samplers and the method by Flajolet, Zimmerman, and Van Cutsem [3]? What is similar? ⋆⋆
- Why does Boltzmann sampling not (need to) use standard specification? Did the simple sampling [3] have to? ⋆⋆
- Why, specifically, is Boltzmann faster than the simple top-down approach? ⋆⋆
- In which setting would you prefer simple top-down, and in which Boltzmann sampling? ⋆⋆
- Give a Boltzmann sampler for binary words that do not contain 101, i. e. ⋆⋆

  $$L = \{w \in \{0,1\}^\star \mid 101 \notin \text{substrings}(w)\} .$$

- What can we do if we have hard lower and/or upper bounds on the size of the generated object, say in an application? ⋆⋆
- Explain why we have to pick x from inside the circle of convergence of the generating function of the class we generate from. What happens if we pick a value from outside the circle? ⋆⋆
- We pick x w. r. t. the circle of convergence of the "main" class we sample from. We then use it as parameter for sampler of *other* classes. Why is this correct, i. e. why is x an admissable parameter for all these classes? ⋆⋆
- How to determine suitable parameter x for your target object size n? ⋆⋆⋆
- How come we can sample the components of a Cartesian product *independently* here? ⋆⋆

  Remember that for simple top-down sampling [3], we had to pick $0 \leq k \leq n$ as size for one component and use $n - k$ for the other, that is the components were dependent.
- What effect do we observe when we generate Motzkin words with a Boltzmann sampler with tolerance ≥ 1 and want to count the number of $*$? ⋆⋆⋆
- Where did the pointing operator Θ go? More specifically, why do we not need it here and how can we sample if it is part of a specification? ⋆⋆⋆
- Explain why/how "bigger" x results in "bigger" objects. ⋆⋆⋆

Advanced Boltzmann

All references in this list refer to the paper by Duchon et al. [2] unless otherwise indicated.

- How can we use Boltzmann samplers for generating objects of sizes within a fixed interval? ⋆
- Exact-size sampling by rejection seems to work nicely. How well does it perform in which cases? ⋆
- Explain how do get a decent sampler (both in terms of "target size accuracy" and runtime) for the three "types" of size distributions Duchon et al. identify – bumpy, flat and peaked – and how to show that it is. ⋆⋆
- Construct a Boltzmann sampler for Motzkin words out of a fixed size interval. ⋆⋆
- Assume you built a Boltzmann sampler and realize its distribution is peaked. What can you do to improve its quality? ⋆⋆
- Why does using a sampler for $\Theta\mathcal{A}$ perform better for exact-size sampling than the one for $\mathcal{A}$? Can it ever perform worse? ⋆⋆
- Why does the exact-size Boltzmann samples using the rejection method always terminate? ⋆⋆
- What exactly does the pointing operator do with the size distribution of the generated objects? ⋆⋆
- The Mean Value Condition (6.1) technically admits $\nu_1(x) = +\infty$ for x near ρ. But that would defeat its purpose; where do the authors exclude this case? ⋆⋆
- How do we deal with generating functions that are not H-admissable? ⋆⋆
- Compare theorems 6.1 and 6.2. In which ways is 6.2 stronger resp. weaker? ⋆⋆
- Explain the origin (and "meaning") of the integral in Theorem 6.3. ⋆⋆
- Does Theorem 6.3 make a statement about all rejection Boltzmann samplers (for suitable classes)? If not, what are the restrictions? ⋆⋆
- Compare theorems 6.3 and 6.5. In which ways is 6.5 stronger resp. weaker? ⋆⋆
- Prove Theorem 6.5. ⋆⋆
- Given a specification, how can you tell which Boltzmann variant to use? ⋆⋆⋆
- Why does the standard deviation grow asymptotically slower than the mean when we have H-admissability? ⋆⋆⋆
- You are tasked to develop a random sampling algorithm for objects of fixed sizes. Which construction do you use, and why? ⋆⋆⋆
- Sketch a proof for Theorem 6.1. ⋆⋆⋆
- How can we relax the conditions of Theorem 6.1 so that we still get *constant* success probability $p \in (0, 1]$ as $n \to \infty$? ⋆⋆⋆

  Note how this would still leave us with an average sampling runtime of $\mathcal{O}(n)$.

Random Numbers

- What does “random” mean (here)? ⋆
- Name five uses of random numbers. ⋆
- Give the general form of a linear congruence sequence. ⋆
- What are the three conditions Knuth [6] gives for a linear congruence sequence to have period m? ⋆
- We would like to drop the addition in the linear congruence method for reasons of efficiency. When is doing so not harmful? ⋆
- When do we say a PRNG is “good”? ⋆⋆
- Explain the linear congruence method. ⋆⋆
- What are good resp. bad choices for parameters m, a and c? Why, and how can we tell? ⋆⋆
- What is the period of the PRN sequence given by ⋆⋆

  $$X_{n+1} = (61 \cdot X_n + 7) \mod 180\ ?$$

- Determine $a > 1$, c and X_0 for a linear congruence sequence with period $m = 40$. ⋆⋆
- Can we make a good PRNG from a bad one plus LZ compression? ⋆⋆⋆
- Assume we have k (good) PRNGs with sequence $\left(G_i(n)\right)_{n\in\mathbb{N}}$ each. What can we say about the PRN sequence given by $\left(G_{n \bmod k}(n \operatorname{div} k)\right)_{n\in\mathbb{N}}$? ⋆⋆⋆
- Show [6, (6) on p 11] that ⋆⋆⋆

  $$X_{n+k} = (a^k \cdot X_n + (a^k - 1) \cdot {}^{c}\!/_{b}) \mod m$$

  for linear congruence sequence $(X_n)n \in \mathbb{N}$ with $a \geq 2$, $b \geq 1$ and $n, k \geq 0$.

- Assume you need normal-distributed random numbers; which method do you use? ⋆⋆⋆
- We define the *domain coverage (ratio)* of a linear congruence PRNG by the number of different values from $\{0, \dots, m\}$ (divided by $m + 1$) it emits. ⋆⋆⋆

  Use Lemma Q and Theorem A [6] to increase the domain without impairing the domain coverage ratio.

- Can the methods we use for attaining uniformity in hash tables (collision resolution) help generating random numbers? ⋆⋆⋆

  Conversely, are linear congruence methods suitably for use as hash function?

References

[1] Edsger W. Dijkstra. “Letters to the Editor: Go to Statement Considered Harmful.” In: *Communications to the ACM* 11.3 (Mar. 1968), pp. 147–148. ISSN: 0001-0782. DOI: 10.1145/362929.362947.

[2] Philippe Duchon et al. “Boltzmann Samplers for the Random Generation of Combinatorial Structures.” English. In: *Combinatorics, Probability and Computing* 13.4-5 (July 2004), pp. 577–625. ISSN: 1469-2163. DOI: 10.1017/S0963548304006315.

[3] Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. “A calculus for the random generation of labelled combinatorial structures.” In: *Theoretical Computer Science* 132.1-2 (Sept. 1994), pp. 1–35. ISSN: 03043975. DOI: 10.1016/0304-3975(94)90226-7.

[4] Leslie M. Goldschlager, Ralph A. Shaw, and John Staples. “The maximum flow problem is log space complete for P.” In: *Theoretical Computer Science* 21.1 (1982), pp. 105–111. ISSN: 0304-3975. DOI: 10.1016/0304-3975(82)90092-5.

[5] J. Hartmanis and R. E. Stearns. “On the computational complexity of algorithms.” In: *Transactions of the American Mathematical Society* 117 (1965), pp. 285–306. ISSN: 1088-6850. DOI: 10.1090/S0002-9947-1965-0170805-7.

[6] Donald E. Knuth. *Seminumerical Algorithms.* 3rd. Vol. 2. The Art Of Computer Programming. Addison-Wesley Longman Publishing, 2001. 762 pp. ISBN: 0201896842.

[7] Sven Oliver Krumke and Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen.* Wiesbaden: Vieweg+Teubner Verlag, 2012. ISBN: 978-3-8348-1849-2. DOI: 10.1007/978-3-8348-2264-2.

[8] Abraham Lempel and Jacob Ziv. “On the Complexity of Finite Sequences.” In: *Information Theory, IEEE Transactions on* 22.1 (1976), pp. 75–81. DOI: 10.1109/TIT.1976.1055501.

[9] Markus E. Nebel. *Entwurf und Analyse von Algorithmen.* Wiesbaden: Vieweg+Teubner Verlag, 2012. ISBN: 978-3-8348-1949-9. DOI: 10.1007/978-3-8348-2339-7.

[10] Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms.* 2nd. Addison-Wesley Professional, 2013, p. 592. ISBN: 032190575X.

[11] Robert Sedgewick and Kevin Wayne. *Algorithms.* Addison-Wesley, Mar. 2011. ISBN: 978-0-321-57351-3.

[12] Jacob Ziv and Abraham Lempel. “Compression of individual sequences via variable-rate coding.” In: *Information Theory, IEEE Transactions on* 24.5 (1978), pp. 530–536. DOI: 10.1109/TIT.1978.1055934.