Notes on Combinatorial Algorithms

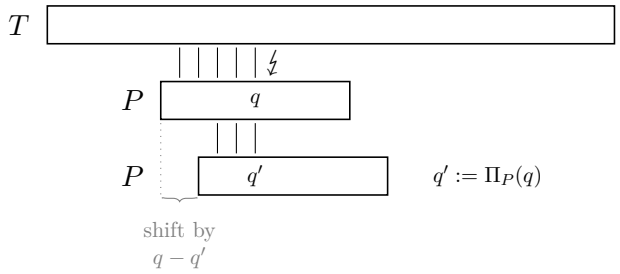
Raphael Reitzig

Winter term 2014/15

2015-03-18

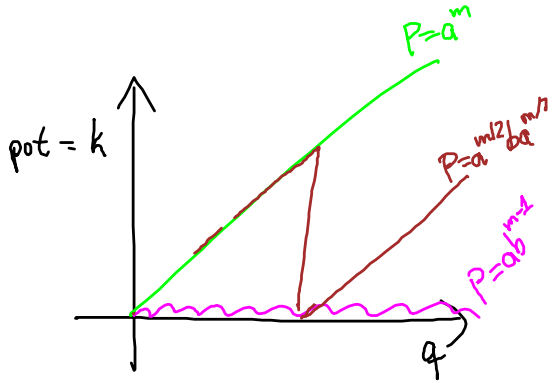
String Matching

We discussed the idea behind prefix-function Π_P and why we can use it to determine shifts in string matching. We came up with the following illustration (basically a generic version of figures 6.3 – 6.5 in Nebel [6, p 253f]):



For a similar argument, see the proof of Lemma 6.8 in Nebel [6, p 256f] or an adaption on `cs.stackexchange.com`[1].

We also discussed the idea behind the runtime analysis for the computation of Π_P [6, p 253f]. Investigating the development of the value of variable `k` for some example strings P proved helpful:



[1] `http://cs.stackexchange.com/questions/1669/`

Recall the core argument: the sum of all decreasings of `k` is an upper bound on the number of iterations of the `WHILE`-loop. Since we can decrease `k` only as often/much as we increase it, and can at most increase it by one in every of the $\approx m = |P|$ iterations of the surrounding `FOR`-loop – note in particular that $\Pi_P(\mathtt{k}) \leq \mathtt{k}$ – we get a total upper bound of m `WHILE`-iterations.

Knuth-Morris-Pratt

For reference, a "Random-Algorithmus" (cf. Definition 6.6) is essentially an *absorbing Markov chain*[2] with exactly one absorbing state.

For the proof of Satz 6.10, note that our goal is

$$\mathbb{E}\, C = \sum_{j=1}^{n} R_{1,j} \ .$$

For the KMP part of the comparative analysis (against the naive algorithm), note that we essentially use *linearity of expectation*.

- We first determine (a bound on) the expected jump target assuming a *random pattern*; for every fixed pattern, the transitions are of course fixed and have mismatch probability, i. e. the same as in the chain for the naive algorithm. We then *fix* these "expected transitions", that is an "average pattern" (that may not even exist!).
- Now we have got the graph structure fixed and assign probabilities corresponding to the random text, just as we did for the naive algorithm,
- The rest is crunching the numbers.

We get the correct result (or at least a bound on it) only because the expecation is linear, so we *can* indeed compute and fix the "inner value" (which is a graph resp. transition matrix here). We can not compute higher moments in this fashion!

Boyer-Moore

Nebel [6, p267] writes that BM usually attains sub-linear runtime in practice. This is not to be read as $o(n)$, however!

- If P is fixed and hence m constant, we clearly get an $\Omega(n)$ lower bound since we never shift by more than m.
- If m and n are unrelated, we do not get better bounds either.
- If $m \in \omega(1) \cap \mathcal{O}(n)$, we may actually see $o(n)$ behaviour.
- The intended meaning, however, is that usually $< n$ symbol comparisons are necessary, and in fact $\leq cn$ for some $c \in (0, 1)$. That is, the algorithm does "usually" not even read the whole input!

[2] `https://en.wikipedia.org/wiki/Absorbing_Markov_chain`

Compression

Sedgewick and Wayne [8, Sec. 5.5] give nice visual clues as to what "randomness" of a text means: comparing image dumps of uncompressed files and their compressions, the loss of "structure" is apparent.

We already know the *Huffman* code [6, p 295ff]. We have shown that it is an *optimal* code [6, p 297f]. In the light of what we have learned in this part of the course – in particular the no-free-lunch theorems – the following questions present themselves.

- Why use the ideas from Lempel and Ziv and not Huffman codes?
- Which texts does Huffman – even as provably optimal code! – make *longer*? Can you give an intuitive reason why, and why this is unavoidable?

For following the presentation of Ziv and Lempel [10, 11] and Welch [9] keep in mind that the main idea of the algorithms is the one outlined in the 1976 article [5]: represent as big parts of the text as possible by references to occurences of the same substring we have already found (in the prefix of the text). The algorithms differ in which substrings they store, i. e. the dictionaries they maintain, and how they encode the references.

The work by Lempel and Ziv [5] prepares the ground for showing the quality of these algorithms *for this idea*. It is instructional to think about the reasons for Lempel and Ziv restricting themselves to this particular model. Keep the time of publication in mind: computer science was just starting to become a thing and the programmers of that time faced severe resource restrictions!

Network Flows

Augmenting Paths

Our main source [4] does not directly give examples for FF failing with real capacities. You should look these up an form an intuition for *why* FF fails (and EK does not). Where does the correctness proof of FF break down for real capactities, and how does the proof for EK circumvent the issue?

We have neither been able to find direct real-world interpretations of minimum cuts (i. e. which practical problem is solved by finding a minimum cut), nor have we had ideas for algorithms that solve Max-Flow via MFMC (i. e. by finding a minimum cut "directly"). You should do some independent research on these question. Are there any such scenarios resp. algorithms? If not, what do we need the MFMC for, i. e. for which proofs is it crucial?

We talked about the limitations of the augmenting-path approach. In addition to the push-relabel approach you will get to know next, you may want to read about the algorithm by Dinic [4, Section 9.6].

Symbolic Method

When applying the symbolic method, be mindful of the fact that the transfer theorems and analysis methods we know count *the number of ways of constructing* objects of a certain size. Only if the specification is unambiguous does the method yield the number of objects (of that size).

The power of the symbolic method lies mainly in its *transfer theorems* that allow us – given a suitable specification – to get quite mechanically from specification to generating function and (with other theorems you have yet to see) to coefficients. In fact, computer algebra systems can automate the process for a rich class of specifications. It's not a silver bullet for all counting problems, though.

- Not *all* specifications can be analysed algorithmically[3]. The problem is in fact not even computable (cf. undecidability of universality $L \stackrel{?}{=} \emptyset$ for context-free languages; all context-free grammars are specifications in the sense of the symbolic method). Solving the equation system for the generating function and/or determining the coefficients from it can be arbitrarily hard.
- Some combinations of objects and size notions do not yield a combinatorial class in the sense of Sedgewick and Flajolet [7, p 221] – even though we can write down a completely valid specification, syntactically!

  As an example, consider *binary tries* (with at least two entries) which can be specified like this:

  $$\mathcal{T} = \square\ \square \;\Big|\; \{\square, \blacksquare\}\ \mathcal{T} \;\Big|\; \mathcal{T}\ \{\square, \blacksquare\} \;\Big|\; \mathcal{T}\ \mathcal{T}$$

  where $\square$ is a leaf with an entry and $\blacksquare$ is a leaf with a `nil`-pointer (which are necessary). If we add as size notion the number of stored entries, i. e. the number of $\square$ nodes, then we get *infinitely many* trees of each size.

Why do we need EGF? Recall that, intuitively, the power series of a (useful) generating function has to converge in an area around 0. Now, for $z \to 0$ the growth of z^n can only compensate for exponential growth in a_n; hence, series that grow super-exponentially can not be represented by (useful) OGF.

Remember that the distinction exists only in our mind when we consider the function; for instance, $(1-z)^{-1}$ is the OGF of $(1)_{n\in\mathbb{N}}$ but the EGF of $(n!)_{n\in\mathbb{N}}$.

Notational note: $\Theta\mathcal{A}$ by Flajolet, Zimmerman, and Van Cutsem [3] is the same as $\mathcal{A}^\bullet$ by Duchon et al. [1].

Why are non-plane trees specified by

$$\mathcal{T} = \mathcal{Z} + \mathcal{Z} \times \mathrm{MSET}(\mathcal{T})$$

in the unlabelled, but

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

[3] At least not exactly.

in the labelled case? Recall the definitions [2]:

$$\begin{aligned}
\mathrm{SET}(\mathcal{T}) &:= \bigcup_{k\geq 0} \mathrm{SET}_k(\mathcal{T}) \,, \\
\mathrm{SET}_k(\mathcal{T}) &:= \mathrm{SEQ}_k(\mathcal{T})/R \quad \text{and} \\
\mathrm{SEQ}_k(\mathcal{T}) &:= \begin{cases} \varepsilon, & k = 0, \\ \underbrace{\mathcal{T} \star \ldots \star \mathcal{T}}_{k\times}, & k > 0, \end{cases}
\end{aligned}$$

where R is arelation under which two sequences are equivalent if and only if they are permutations of each other. Note that relabelling happens while constructing the sequence (whose order is later discarded) so the same tree can be chosen for each element.

"True" multisets do not exist in the labelled world, and do not make much sense either (an inconsistent labelling would result, after all). We notice that we do not have any means to define "structural" sets in the labelled world, either.

Singularity Analysis

Duchon et al. [1] use the notions of "Δ-singular" and "Δ-analytic" functions. They are synonymous.

Note that many (if not all the) transfer theorems rest on the following results which hold for suitable f (Google the details):

- By *Cauchy's Integral Formula*,

  $$[z^n]f(z) = \frac{1}{2\pi i} \oint_\gamma \frac{f(z)}{z^{n+1}} dz$$

  where countour γ encircles the origin once in a counterclockwise manner and contains the circle of convergence of f (around the origin).

- By the *Residue Theorem*, if f has poles $Z \subset \mathbb{C}$ within γ, then

  $$\oint_\gamma f(z)dz = 2\pi i \cdot \sum_{z_0 \in Z} \mathrm{Res}(f, z_0) \,.$$

Given a function f with known characteristics (you have seen some), it is sometimes possible to obtain residues or evaluate the integral (of Cauchy's formula). The integral also offers the possibility of *approximation*. Singularities that are not poles, e. g. from roots or logarithms, present additional challenges.

For a more detailed and rigorous treatment on singularity analysis see Flajolet and Sedgewick [2].

References

[1] 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.

[2] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. ISBN: 9780511801655. DOI: 10.1017/CBO9780511801655. URL: http://ebooks.cambridge.org/ref/id/CBO9780511801655.

[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] 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.

[5] 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.

[6] 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.

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

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

[9] T.A. Welch. "A Technique for High-Performance Data Compression." In: *Computer* 17.6 (1984), pp. 8–19. ISSN: 0018-9162. DOI: 10.1109/MC.1984.1659158.

[10] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression." In: *Information Theory, IEEE Transactions on* 23.3 (1977), pp. 337–343. DOI: 10.1109/TIT.1977.1055714.

[11] 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.