SPEZIELLE THEMEN DER ALGORITHMIK

Patrizio Angelini

Universität Tübingen

angelini@informatik.uni-tuebingen.de

INFORMATION

- Lectures: Friday, 10:00 – 13:00 – Room C215

- Assignments: 2
  - At least 1 to participate to the exam
  - Possible bonus points for the exam

- Exam: to be decided

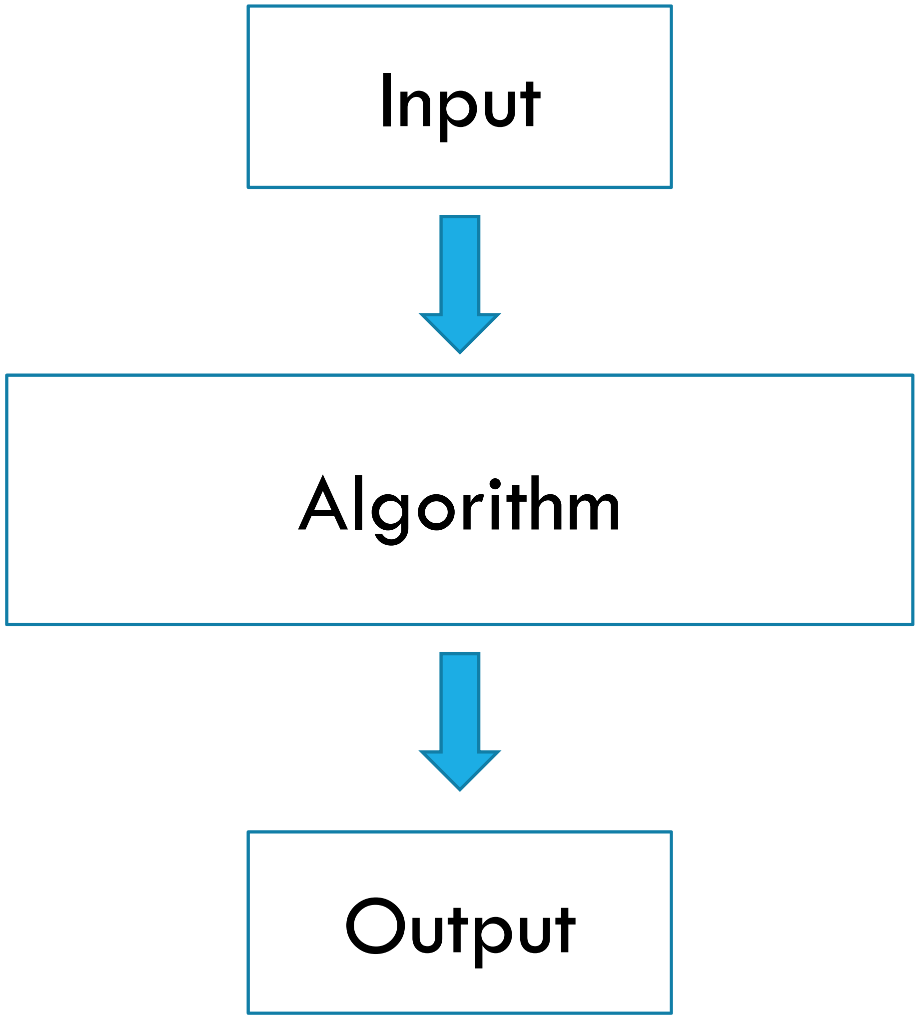
Please register asap to the Moodle (password = stda)

TOPICS

- **Part 1**: General algorithmic techniques for settings in which exact and efficient algorithms are not possible
  - Online algorithms
  - Approximation algorithms
  - Randomization
  - Game Theory
- **Part 2**: More specific algorithmic topics, mainly about graphs
  - Data structures for planar graph embeddings
  - Schnyder realizers for straight-line drawings
  - Euler's formula and Crossing Lemma

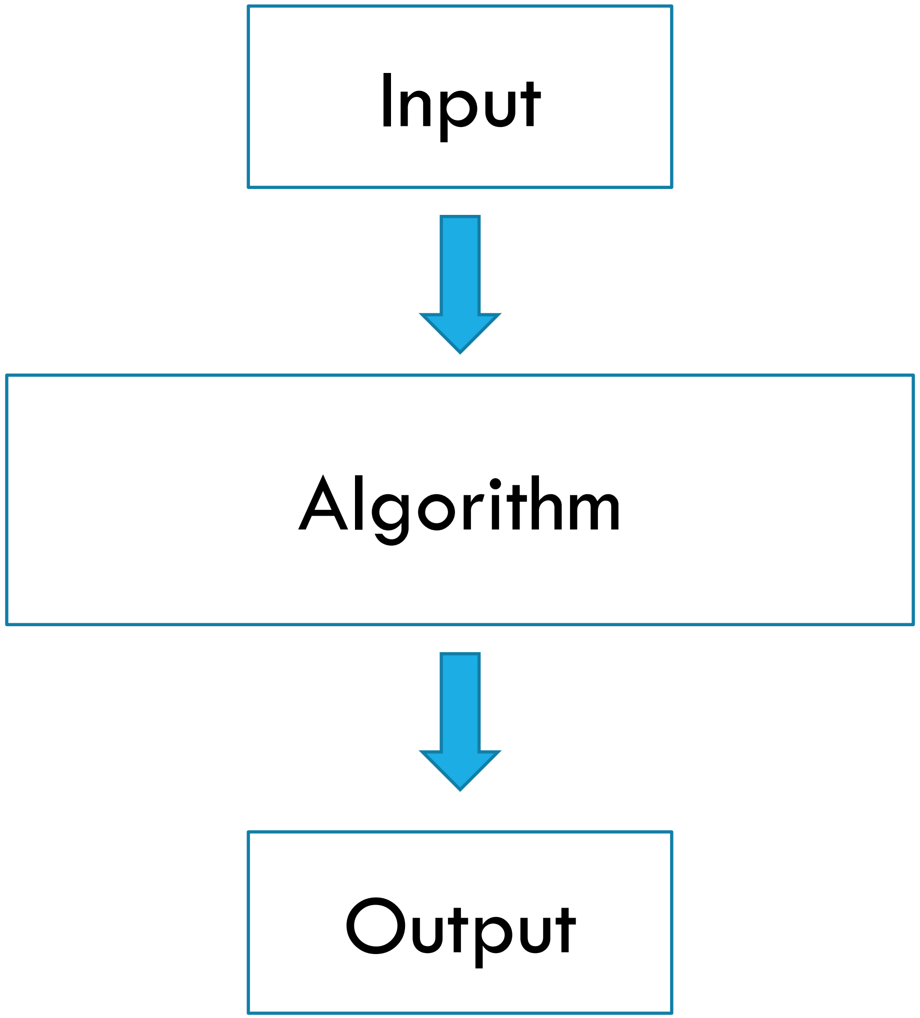
PART 1

- Given a problem, defining an input and a desired output, we usually aim at designing exact and efficient algorithms
- Exact: correct/optimal solution
- Efficient: polynomial running time



PART 1

- Unfortunately, this is not always possible
- The problem may be difficult
- The input may be difficult to read
- The centralized execution of the algorithm may be not possible



ONLINE ALGORITHMS

- The input is provided one portion at a time
  - The algorithm has to make choices without knowing the following portions

- The quality is measured by comparing it with an optimum offline algorithm for the same problem and the same input
  - The offline algorithm knows the whole input in advance

- Not to be confused with real-time and streaming
  - Time plays no role here
  - Input has finite size

COMPETITIVE ANALYSIS

- The measure is performed by considering an adversary providing at every step the next portion of the input, which is the worst possible for the current state
  - Corresponds to the worst-case analysis in the offline setting

- Competitive ratio: An algorithm is *c*-competitive if it is at most *c* times worse than the optimum

COMPETITIVE ANALYSIS

- Let A be an online algorithm for a problem
- Let OPT be the optimal offline algorithm for the same problem

Algorithm A is *c*-competitive if there exists a constant β such that, for every input sequence σ

$$A(\sigma) \leq c\ OPT(\sigma) + \beta$$

c is called competitive-ratio

APPLICATIONS

- Scheduling
  - Industrial tasks
  - Computer processes
- Load balancing
- Paging
- Routing
- Data structures
- …
- Ski rental ☺

SKI RENTAL PROBLEM

- A person *P* goes to the Alps and wants to ski
- *P* does not have any ski, and should decide whether buying or renting them
- Renting costs *1* EUR / day
- Buying costs *B* EUR
- The weather is unstable, so *P* does not know how many days he can stay
- What should he do?

SKI RENTAL PROBLEM: OFFLINE OPTIMUM

- The input consists of the sequence σ of good-weather days
- The offline algorithm knows how long is this sequence
  - It knows how many days *P* can ski
  - It can decide whether *P* should rent or buy
  - $If\ |\sigma| < B, then\ RENT, otherwise\ BUY$
  - $OPT(\sigma) = \min(\sigma, B)$

SKI RENTAL PROBLEM: ONLINE

- What is the worst-case scenario?
  - How does a bad adversary behave?

- If *P* buys ski at day *d*, then the adversary makes it rain at day *d+1*

- If *P* does not buy at day *d*, then the adversary lets day *d+1* be sunny

SKI RENTAL PROBLEM: ONLINE

- Algorithm 1: *BUY* at day *d=1*
  - What is the competitive ratio?
  - B/1 = B
- Algorithm 2: always *RENT* (say, *BUY* at day *d=∞*)
  - What is the competitive ratio?
  - ∞/B = ∞
- Algorithm 3: *BUY* at day *d = B*
  - What is the competitive ratio?
  - (B-1+B)/B = 2 -1/B

SKI RENTAL PROBLEM: ONLINE

- **Theorem**: $2 - 1/B$ is the best competitive ratio
- **Proof**: Suppose that *P* buys at day $d = d^*$
- The opponent will make it rain on day d^*+1

- If $d^* < B$, then c.r. $= (d^*+B-1)/\min\{d^*,B\} = (d^*+B-1)/d^* >= 2$

- If $d^* > B$, then c.r. $= (d^*+B-1)/\min\{d^*,B\} = (d^*+B-1)/B >= 2$

ANOTHER PROBLEM: K-PAGING

- A two-level memory system, accessed by a CPU, consisting of a small fast memory and a large slow memory
- Memory is partitioned into pages of equal size
- Input: a sequence of requests, each specifying a page in the memory
- A request can be served immediately if the page is in the fast memory
- Otherwise, a page fault occurs

K-PAGING

- In case of a page fault, the missing page is loaded from the slow into the fast memory so that the request can be served
- To make room, a page has to be removed from the fast memory
- Which one?
  - A decision must be made without knowing future requests
- The goal is to minimize the number of page faults

K-PAGING: MOST USED ALGORITHMS

- LRU: Least Recently Used
  - The page to be removed is the one that has been requested least recently
- FIFO: First In First Out
  - The page to be removed is the one that has been added to the fast memory least recently
- How competitive are these two algorithms?

K-PAGING: MODEL

- Fast memory has k pages
- Slow memory has $m >> k$ pages
- Input: sequence σ of requests
- Cost $A(\sigma)$ of an algorithm A: number of page faults

LRU : LEAST RECENTLY USED

- **Theorem**: LRU is k-competitive
- **Proof idea**:
  - Divide the sequence σ into phases such that in each phase φ
    - there are at most k page faults in LRU, that is, $LRU(\varphi) <= k$
    - in any (offline) algorithm, even the optimum one OPT, there is at least 1 page fault, that is OPT(φ)>=1
- This implies that $LRU(\sigma)/OPT(\sigma) <= k$, for any σ

LRU : LEAST RECENTLY USED

- Divide σ into phases $\sigma = \varphi_0, \varphi_1, \ldots, \varphi_i, \ldots, \varphi_m$ in such a way that:
  - φ_0 has at most k faults (and at least one)
  - φ_i has exactly k faults, for each $i=1, \ldots, m$

- This can be done by following the sequence σ in reverse order, starting from the end, and counting k faults each time

LRU : LEAST RECENTLY USED

- Assume that, before applying the two algorithms *LRU* and *OPT*, the fast memory contains the same pages (or is empty)

- In φ_0, the first page fault for *LRU* is also a page fault for *OPT*
  - Hence, OPT has at least 1 page fault

LRU : LEAST RECENTLY USED

- Consider any φ_i, with $i > 0$

- Let p be the last page that has been requested in phase φ_{i-1}
  - Page p is in the fast memory at the beginning of φ_i

- We prove that the set of pages that are requested in phase φ_i contains (at least) k pages different from p
  - This implies there is at least a page fault

LRU : LEAST RECENTLY USED

- We prove that the set of pages that are requested in phase φ_i contains (at least) k pages different from p
- Let $r_1, \ldots, r_k$ be the k requests on which *LRU* had a page fault in φ_i
- Let r_0 *be the last request in* φ_{i-1}, *which requested p*

- Two cases, based on whether there exist two requests in $r_0, \ldots, r_k$ of the same page, or not
- If not, p(r_1), …, p(r_k) are the pages we are looking for

LRU : LEAST RECENTLY USED

- We prove that the set of pages that are requested in phase φ_i contains (at least) k pages different from p
- *If yes, let* $r_h = r_j$ be two requests of the same page such that $r_h, \ldots, r_{j-1}$ request different pages
- Since $p(r_h)$ was the least recently requested page after request r_h, in order to have a new page fault of the same page $p(r_h) = p(r_j)$ at r_j, there are at least k different pages requested in $r_h, \ldots, r_{j-1}$
  - If $h = 0$, then these k pages are different from p
  - If $h > 0$, then $k\text{-}1$ of them are different from p, and $p(r_h)$ is different from them and from p

LRU : LEAST RECENTLY USED

- Note that the proof is independent of the optimum algorithm *OPT*
  - It only uses upper and lower bounds

- An analogous argument can be used to prove that algorithm FIFO is *k*-competitive

- Can we do better?

LRU : LEAST RECENTLY USED

- **Theorem**: There exists no c-competitive deterministic online algorithm for the k-paging problem such that $c < k$
- **Proof***:*
- Consider a set of $k+1$ pages.
- For any deterministic algorithm A, there is an input sequence such that A has a page fault at every step
  - The opponent asks the (unique) page not in fast memory

LRU : LEAST RECENTLY USED

- **Theorem**: There exists no *c*-competitive deterministic online algorithm for the *k*-paging problem such that $c < k$
- **Proof (continued)**:
- For each sequence of *k* requests, there is one page that is not present
- When *OPT* has a page fault, it can remove from the memory a page that will not be requested in the next $k-1$ steps
  - In every *k* consecutive steps, *OPT* pays at most once

LITERATURE

- S. Albers. *Online algorithms: a survey*. Math. Program. 97(1-2): 3-26 (2003)
- A. Borodin, R. El-Yaniv: *Online computation and competitive analysis*. Cambridge University Press, ISBN 978-0-521-56392-5, pp. I-XVIII, 1-414