CSE2310 Algorithm Design

Lecture 1: Course introduction & Greedy Algorithms

Stefan Hugtenburg, Emir Demirović, and Mathijs de Weerdt

Algorithmics group — EEMCS — TU Delft

2023–2024 Q2

TU Delft

What is this thing called "Algorithm"?

> *An algorithm is the representation of an insight into solving a problem in such a way that with this algorithm solutions can be found without having any (further) insights.*
>
> *(Prof. Dr. Cees Witteveen, inaugural speech, 17 January 2007)*

In this course we will *design* and *analyze* such algorithms.

TU Delft

What is this thing called "Algorithm"?

An algorithm is the representation of an insight into solving a problem in such a way that with this algorithm solutions can be found without having any (further) insights.

(Prof. Dr. Cees Witteveen, inaugural speech, 17 January 2007)

In this course we will *design* and *analyze* such algorithms.

Way back when

Muḥammad ibn Mūsā al-Khwārizmī (780 to 850 AD), gave the first *systematic* solution of linear and quadratic equations.

Image from: *Wikipedia*

So what do we use this for...?

Answer: Many things!

Within Computer Science:

- Caching: what is a good caching policy?
- Scheduling: what jobs should we do when?
- Networking: what is an optimal routing protocol?
- Computer Graphics: how do we efficiently remove hidden lines?

TU Delft

So what do we use this for...?

Answer: Many things!

Within Computer Science:

- Caching: what is a good caching policy?
- Scheduling: what jobs should we do when?
- Networking: what is an optimal routing protocol?
- Computer Graphics: how do we efficiently remove hidden lines?

But also beyond that!

- Operations Research: how do we optimally schedule classes?
- Computational Biology: how do we efficiently compute 3D properties of RNA?
- Artificial Intelligence: how do we produce a plan to achieve your goal?

TU Delft

In this course

Question: How does this relate to Algorithms & Datastructures?

- Now that you know about all these data structures, like lists, trees, and graphs...

TU Delft

In this course

Question: How does this relate to Algorithms & Datastructures?

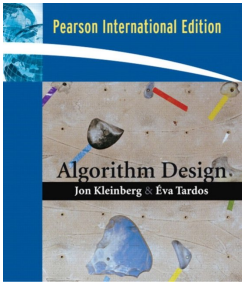
- Now that you know about all these data structures, like lists, trees, and graphs...
- We will look at algorithms for *new* problems that these data structures allow us to solve.
- There are three criteria we pay significant attention to:
  - Efficiency: Is this the best algorithm in terms of *time and/or space complexity*?
  - Correctness: Can we guarantee the answer returned is *correct*?
  - Optimality: Can we guarantee the answer returned is *the best answer*?

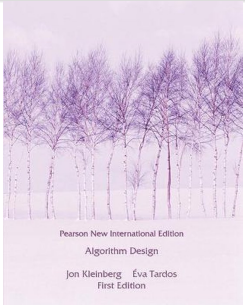
TU Delft

Our place in the curriculum

Dependencies

- We build on the data structures you have used in Algorithms & Data structures.
- We will use the proof techniques you have mastered in Reasoning & Logic.
- We rely on your programming skills such as from Object-Oriented Programming.





Book?

Algorithm Design by Kleinberg and Tardos, which shows many relations with realistic problems.

TU Delft

Why is this important?

Your futures!

Algorithms are taking on more and more roles in society.
It is up to *you* to ensure this happens responsibly!

Solliciteren bij een algoritme

Kunstmatige intelligentie Unilever gebruikt slimme software voor het aannemen van trainees. Het doel: het personeelsbestand diverser maken.

Applying for a job with an Algorithm (*NRC, 11 May 2018*)

Artificial intelligence Unilever uses smart software for the hiring of their trainees. The goal: diversifying their personnel.

TU Delft

What we will be teaching you?

Learning objectives

Four algorithmic paradigms, four modules

1. Greedy (Week 1 and 2) – by Mathijs de Weerdt
2. Divide & Conquer (Week 3) – by Emir Demirović
3. Dynamic Programming (Week 4–6) – by Emir Demirović
4. Network Flow (Week 7–9) – by Stefan Hugtenburg

TU Delft

What we will be teaching you?

Learning objectives

Four algorithmic paradigms, four modules

1. Greedy (Week 1 and 2) – by Mathijs de Weerdt
2. Divide & Conquer (Week 3) – by Emir Demirović
3. Dynamic Programming (Week 4–6) – by Emir Demirović
4. Network Flow (Week 7–9) – by Stefan Hugtenburg

But of course we will teach you more than just that!

- critical thinking and problem-solving,
- experimentally and theoretically analyzing runtime, correctness, and optimality, and
- communicating your findings and reflecting on those by you and others.

TU Delft

The exams

One computer exam

Some programming assignments in WebLab, which focus mostly on:

- Designing a new algorithm.
- Implementing a new algorithm.

Two written exams

- Definitely open questions, and potentially Multiple Choice questions, focusing on:
  - Remembering properties of algorithms.
  - Applying algorithms.
  - Proofs for correctness.
  - Determining runtime or space.
  - Designing/modifying algorithms.

The final grade

Note: Different from last year!

The final grade (from the studyguide)

- Written exam part 1 in week 5 (W_1)
- Written exam part 2 in week 10 (W_2)
- Implementation exam in week 10 (P)

The final grade for the course is computed as follows:

- $W = 0.5W_1 + 0.5W_2$
- If $P < 5$ or $W < 5$, the final grade is $\min(P, W)$.
- Else the final grade is $0.5W + 0.5P$

TU Delft

So how does this all work then?

Just sitting here is not enough

Just listening to or reading about the material is not enough to pass this course.

TU Delft

So how does this all work then?

Just sitting here is not enough

Just listening to or reading about the material is not enough to pass this course.

You need to get your hands dirty

- You need to play with this, fail with this, and try again!

TU Delft

So how does this all work then?

Just sitting here is not enough

Just listening to or reading about the material is not enough to pass this course.

You need to get your hands dirty

- You need to play with this, fail with this, and try again!
- It is up to you to make sure the failing and trying again happens *before* the exam, and not during it!

TU Delft

We offer a lot of material to help

A varied menu

To help you get through, we offer *a lot of* practice material.

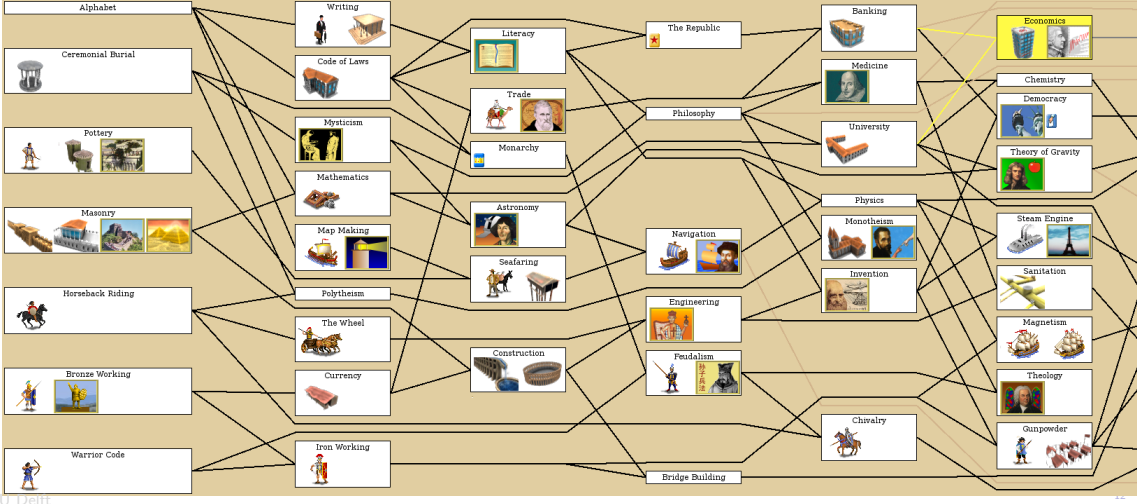
- Quizzes with old exam MC-questions
- Peer-reviewed and TA-reviewed open question assignments
- Programming assignments in WebLab

The menu chart

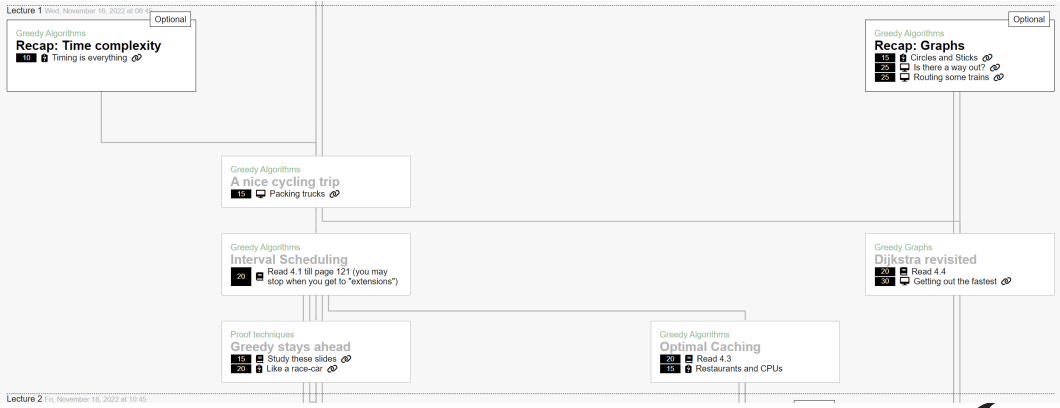
All the materials is structured in *Skill circuits*.

TU Delft

Skill circuits are like technology trees



Skill circuits are like technology trees



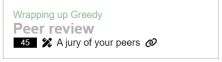
Skill circuits: https://skills.ewi.tudelft.nl/

A skill contains

Tasks like:

- reading a part of the book
- implementing some algorithm
- answering some MC-questions

Skill circuits: https://skills.ewi.tudelft.nl/



A skill contains

Tasks like:

- reading a part of the book
- implementing some algorithm
- answering some MC-questions

Peer review assignment

Unlock peer reviews to practice (with feedback during lab):

- open questions requiring algorithms or proofs.
- experience importance of correct formulations.

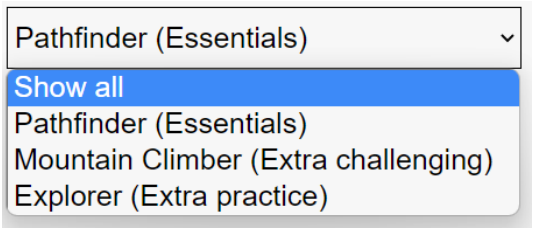
TA-check assignments

When you have prepared sufficiently, you can practice with:

- open questions requiring algorithms or proofs.
- experimental runtime analysis.

t

Not just one path...

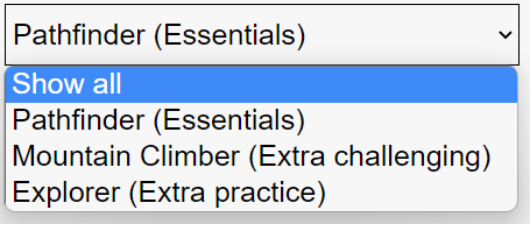


Choose your own path

- The pathfinder: most time-efficient path, including the essentials
- The mountain climber: some extra challenging tasks
- The explorer: more doable tasks

New: you're now completely free to edit your own path (click on skill and delete/add)!

Not just one path...



Working together

Travel together (e.g., in project group)! Two caveats:

- Do the peer review assignments **alone** to get fair assessment of your progress.
- Do not submit similar answers for the TA-check, but **one per team**.

How to reach us

Have a question?

1. Ask your colleagues. Explaining things to each other is very effective to learn!
2. Post it on **Answers EWI**: `https://answers.ewi.tudelft.nl/`.
3. Ask the TAs during the labs.
4. Ask us during lecture breaks/after the lecture.
5. For *administrative/personal issues only*: ad-cs-ewi@tudelft.nl

Note: e-mails to personal e-mail addresses of lecturers will be ignored.

TU Delft

Course strategy

Do it properly, or not at all!

If you decide to take this course this year, commit yourself:

- prepare *before and after* the lectures (at your own speed):
  - read pages from book
  - do the written assignments/proofs
  - do the implementation assignments

  the skill circuits indicate what is expected before and after each lecture
- try out the evidence-based study techniques during the *Acing AD* sessions (register via the Queue: `https://queue.tudelft.nl/lab/7543`)
- actively participate in lectures:
  - make notes of essential parts (helps you to remember better)
  - try to answer (multiple choice) questions and explain them to each other
  - think along with introduction of new topics

TU Delft

Questions?

Question: Ready, set, . . .

So after all that, are you all set to go?
Or are there some questions already?

TU Delft

We appreciate your feedback!

Thanks to student feedback, we are continually improving!

- Since 2019:
  - A redesign in how we present the material to you
- Since 2020:
  - Improved skill circuits
  - More smaller assignments, also for proofs
- Since 2021:
  - Sessions on how to study for AD
- Since 2022:
  - Clean up of some issues with implementation assignments
  - The Quadruple Quest
- Since this year:
  - Change in the peer review assignments
  - Change in exam structure so that greedy proofs are done earlier

Your feedback

Your feedback can help us improve even more!

The Quadruple Quest

Making its way to the year 2 courses now!

- Some of you have already joined the quest starting in R&L and ADS last year.
- The Quadruple Quest (QQ) continues in AD this quarter and you can join in even if you have not done the parts of R&L and ADS!
- Weekly puzzles based on exam-level questions (usually with a twist) wrapped up in a story.
- We hope you will join in (ideally as a team!) to solve them.
- For those of you that do, there may be a sticker at the end.
- And of course the story will conclude in ACC next quarter!

TU Delft

You are here

The course so far

- Course organisation

Today's content

- Checking your ADS/R&L knowledge – algorithms with proofs
- An introduction to Greedy algorithms – also with proofs

The future

- More greedy algorithms
- Divide & Conquer algorithms
- Dynamic programming
- Network Flow

Recap

Putting you to work

Prerequisite knowledge

What should you be able to use/do?

- Proving techniques: proof by induction and by contradiction
- Complexity analysis (time and space)
- Data structures: Stacks, Queues, Priority Queues, Trees, and Graphs
- Graph algorithms:
  - BFS, DFS, Topological orderings
  - Shortest path algorithm: Dijkstra
  - Minimum spanning trees: Kruskal & Prim-Jarnik

TU Delft

Prerequisite knowledge

What should you be able to use/do?

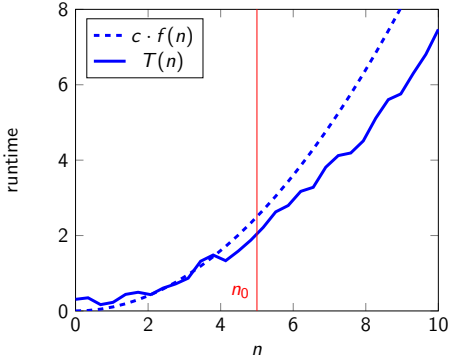
- Proving techniques: proof by induction and by contradiction
- Complexity analysis (time and space)
- Data structures: Stacks, Queues, Priority Queues, Trees, and Graphs
- Graph algorithms:
  - BFS, DFS, Topological orderings
  - Shortest path algorithm: Dijkstra
  - Minimum spanning trees: Kruskal & Prim-Jarnik

Need a refresh?

- *Delftse Foundations of Computation* from CSE1300.
- *Data Structures and Algorithms in Java* from CSE1305.
- Kleinberg & Tardos: Chapters 1–3, Sections 4.4–4.6, 6.8–6.9.

Asymptotic order of growth

Were there others besides the big-Oh?



Let n denote the length of the input and $T(n)$ describe the runtime of an algorithm.

Definition (Upper bound on the runtime)

$T(n)$ is $O(f(n))$ iff $\exists c > 0 \in \mathbb{R}, n_0 > 0 \in \mathbb{N}$ such that $\forall n \geq n_0 : T(n) \leq c \cdot f(n)$

In English: The runtime $T(n)$ is smaller than $c \cdot f(n)$ after some point n_0.

Asymptotic order of growth

Were there others besides the big-Oh?

Definition (Upper bound on the runtime)

$T(n)$ is $O(f(n))$ iff $\exists c > 0 \in \mathbb{R}, n_0 > 0 \in \mathbb{N}$ such that $\forall n \geq n_0 : T(n) \leq c \cdot f(n)$

In English: The function $T(n)$ is smaller than $c \cdot f(n)$ after some point n_0.

Definition (Lower bound on the runtime)

$T(n)$ is $\Omega(f(n))$ iff $\exists c > 0 \in \mathbb{R}, n_0 > 0 \in \mathbb{N}$ such that $\forall n \geq n_0 : T(n) \geq c \cdot f(n)$

In English: The function $T(n)$ is larger than $c \cdot f(n)$ after some point n_0.

Definition (Tight bound of the runtime)

$T(n)$ is $\Theta(f(n))$ iff $T(n)$ is $O(f(n))$ and $T(n)$ is $\Omega(f(n))$.

TU Delft

Why are you talking about the queen?

Please take your browser to `vevox.com` and use Session ID `191-417-320`

Question: Finding maxima

What is the tightest worst-case upper bound on the runtime to find the maximum value in an array of length n?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$
E. $O(n^2)$
F. I don't know

TU Delft

Why are you talking about the queen?

Please take your browser to `vevox.com` and use Session ID `191-417-320`

Question: Finding maxima

What is the tightest worst-case upper bound on the runtime to find the maximum value in an array of length n?

Answer: Linear time!

At most a constant factor times the size of the input.

function ComputeMax($a_1, \ldots, a_n$)
    max $\leftarrow a_1$
    for $i = 2 \rightarrow n$ **do**
        if $a_i >$ max **then**
            max $\leftarrow a_i$
    return max

About notation

Question: Deciphering the hieroglyphs

What is a correct *formulation* of an upper bound on the runtime of an algorithm?

A. $T(n) = O(n)$.
B. $O(n)$ is n.
C. The runtime is $O(n)$.
D. The runtime is n.
E. I don't know.

TU Delft

About notation

Question: Deciphering the hieroglyphs

What is a correct *formulation* of an upper bound on the runtime of an algorithm?

A. $T(n) = O(n)$. We understand you, but we prefer $T(n) \in O(n)$.
B. $O(n)$ is n.
C. The runtime is $O(n)$.
D. The runtime is n.
E. I don't know.

TU Delft

About notation

Question: Deciphering the hieroglyphs

What is a correct *formulation* of an upper bound on the runtime of an algorithm?

A. $T(n) = O(n)$. We understand you, but we prefer $T(n) \in O(n)$.
B. $O(n)$ is n. This is incorrect, it does not mention the runtime!
C. The runtime is $O(n)$.
D. The runtime is n.
E. I don't know.

TU Delft

About notation

Question: Deciphering the hieroglyphs

What is a correct *formulation* of an upper bound on the runtime of an algorithm?

A. $T(n) = O(n)$. We understand you, but we prefer $T(n) \in O(n)$.

B. $O(n)$ is n. This is incorrect, it does not mention the runtime!

C. The runtime is $O(n)$. Yes! $T(n)$ is $O(n)$ is also fine!

D. The runtime is n.

E. I don't know.

TU Delft

About notation

Question: Deciphering the hieroglyphs

What is a correct *formulation* of an upper bound on the runtime of an algorithm?

A. $T(n) = O(n)$. We understand you, but we prefer $T(n) \in O(n)$.

B. $O(n)$ is n. This is incorrect, it does not mention the runtime!

C. The runtime is $O(n)$. Yes! $T(n)$ is $O(n)$ is also fine!

D. The runtime is n. This is incorrect, there is some constant times n work!

E. I don't know.

TU Delft

About notation

Question: Deciphering the hieroglyphs

What is a correct *formulation* of an upper bound on the runtime of an algorithm?

A. $T(n) = O(n)$. We understand you, but we prefer $T(n) \in O(n)$.

B. $O(n)$ is n. This is incorrect, it does not mention the runtime!

C. The runtime is $O(n)$. Yes! $T(n)$ is $O(n)$ is also fine!

D. The runtime is n. This is incorrect, there is some constant times n work!

E. I don't know.

Or, as in the definition: $T(n)$ is $O(n)$.

Playing with Os and Omegas

Question: So many options

Take $T(n) = 32n^2 + 17n + 32$.
Which, if any, of these are true? $T(n)$ is ...

A. $O(n^2)$
B. $\Omega(n^3)$
C. $O(n^3)$
D. $\Theta(n)$
E. $O(n)$
F. $\Omega(n^2)$
G. $\Omega(n)$
H. $\Theta(n^2)$
I. $\Omega(n \log n)$
J. $O(n \log n)$

TU Delft

Playing with Os and Omegas

Question: So many options

Take $T(n) = 32n^2 + 17n + 32$.
Which, if any, of these are true? $T(n)$ is ...

A. $O(n^2)$
B. $\Omega(n^3)$
C. $O(n^3)$
D. $\Theta(n)$
E. $O(n)$
F. $\Omega(n^2)$
G. $\Omega(n)$
H. $\Theta(n^2)$
I. $\Omega(n \log n)$
J. $O(n \log n)$

TU Delft

About such polynomials

Question: Polynomials

Take a polynomial $f(n) = a_0 + a_1 n + \cdots + a_d n^d$, such that $f(n)$ is $\Theta(g(n))$ if $a_d > 0$. What is the simplest $g(n)$ that makes this statement correct?

TU Delft

About such polynomials

Question: Polynomials

Take a polynomial $f(n) = a_0 + a_1 n + \cdots + a_d n^d$, such that $f(n)$ is $\Theta(g(n))$ if $a_d > 0$. What is the simplest $g(n)$ that makes this statement correct?

Answer: Simplest terms

$g(n) = n^d$

TU Delft

Some more worst-case analysis

Question: Sorting cards

What is the tightest worst-case upper bound on the runtime of **insertion sort** of all elements of an array of length n?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$
E. $O(n^2)$
F. I don't know

TU Delft

Some more worst-case analysis

Question: Sorting cards

What is the tightest worst-case upper bound on the runtime of **insertion sort** of all elements of an array of length n?

Answer: Fair and Square

Quadratic time. When inserting an element, possibly all elements in the array need to be shifted. Worst-case: n insertions, each taking $O(n)$ time so $O(n^2)$ time in total.

TU Delft

Wait, couldn't we do better?

Question: Sorting cards

What is the tightest worst-case upper bound on the runtime of **merge sort** of all elements of an array of length n?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$
E. $O(n^2)$
F. I don't know

TU Delft

Wait, couldn't we do better?

Question: Sorting cards

What is the tightest worst-case upper bound on the runtime of **merge sort** of all elements of an array of length n?

Answer: More complex, but a smaller time complexity

We can do this in $O(n \log n)$ time! We will revisit merge sort two weeks from now when we talk about Divide & Conquer algorithms.

TU Delft

About these logarithms

Question: Logarithms

Is it true that: $\forall d > 0$, $\log n$ is $O(n^d)$?

A. Yes

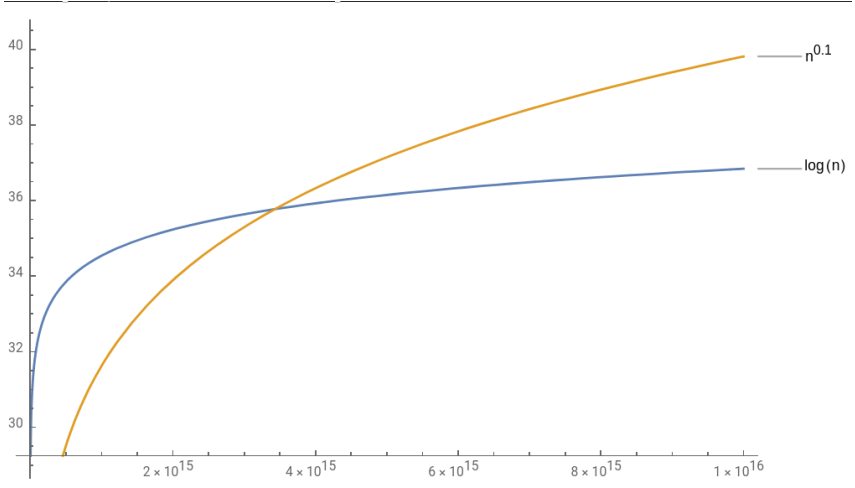
B. No

C. It depends.

D. I don't know.

TU Delft

Logarithms



Logs are slow!

Answer: So yes!

$\log(n)$ grows slower than every polynomial!

TU Delft

Logs are slow!

Answer: So yes!

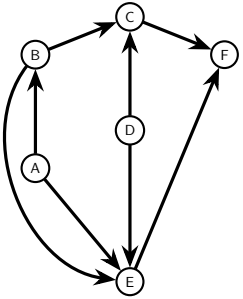
$\log(n)$ grows slower than every polynomial!

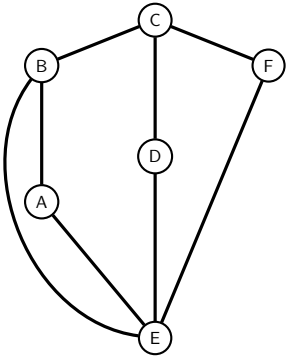
All your bases are the same

$O(\log_a(n)) = O(\log_b(n))$ for any constants $a, b > 0$
Because $\log_a(n) = \log_b(n)/\log_b(a)$, aka a constant ratio in difference!

TU Delft

Graphs





Undirected graphs

Question: Components and edges

Given an undirected graph $G = (V, E)$ with $|V| = n$. G contains no cycles and G consists of c (disjoint) connected components. What is $|E|$?

A. $|E| = n - 1$

B. $|E| = n - c - 1$

C. $|E| = n - c$

D. We cannot determine $|E|$ based solely on these properties.

TU Delft

Undirected graphs

Question: Components and edges

Given an undirected graph $G = (V, E)$ with $|V| = n$. G contains no cycles and G consists of c (disjoint) connected components. What is $|E|$?

A. $|E| = n - 1$
B. $|E| = n - c - 1$
C. $|E| = n - c$
D. We cannot determine $|E|$ based solely on these properties.

Answer: Sure you can

No cycles, means $|E| = n - 1$. Every separate component loses another edges. So $|E| = n - 1 - (c - 1) = n - c$.

DAGs and topological orderings

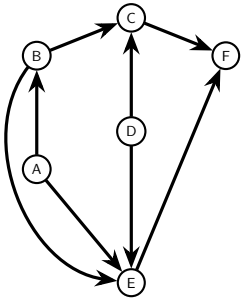
No geographical knowledge required

Definition (DAG)

A *DAG* is a directed graph that contains no directed cycles.

Precedence constraints

Let an edge (v_i, v_j) represent that v_i must happen before v_j.
Often used in *planning* algorithms.



TU Delft

DAGs and topological orderings

No geographical knowledge required

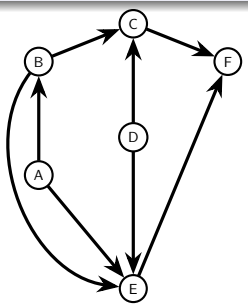
Definition (Topological ordering)

A *topological ordering* of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have $i < j$.

Every DAG

Every DAG has a topological ordering, as we will prove soon

In this case: A, B, D, E, C, F is one option.



TU Delft

Precedence constraints

Applications

- Skill circuits: we recommend you train skill x before skill y.
- Compilation: module v_i must be compiled before v_j.
- Pipeline of computing jobs: output of job v_i is the input for job v_j.

Question: But how do we find it?

How do we *efficiently* and *correctly* find a topological order?

A. Sort the vertices on decreasing outdegree.

B. Sort the vertices on increasing indegree.

C. Repeatedly remove a node with no outgoing edges from the graph and place this *after* the remaining vertices of the graph.

D. Repeatedly remove a node with no incoming edges from the graph and place this *before* the remaining vertices of the graph.

An algorithm for topological ordering

function TopologicalOrdering(G)
  if $|V| = 0$ **then**
    return empty list
  find a node v with no incoming edges
  remainder $\leftarrow$ TopologicalOrdering($G - \{v\}$)
  return v followed by remainder

Claim about runtime

This algorithm runs in $O(n + m)$ time, where $|V| = n$ and $|E| = m$.

TU Delft

An algorithm for topological ordering

function TopologicalOrdering(G)
  if $|V| = 0$ **then**
    return empty list
  find a node v with no incoming edges
  remainder $\leftarrow$ TopologicalOrdering($G - \{v\}$)
  return v followed by remainder

Claim about runtime

This algorithm runs in $O(n + m)$ time, where $|V| = n$ and $|E| = m$.

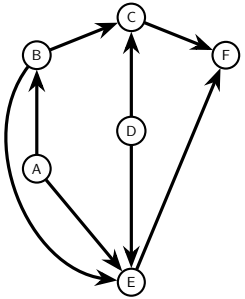
Claim about correctness

- There is always a node with no incoming edges.
- Taking this repeatedly gives us a topological order.

First things first

Lemma 3.19

If G is a DAG, then G has a node with no incoming edges.



TU Delft

First things first

Lemma 3.19

If G is a DAG, then G has a node with no incoming edges.

Proof by contradiction.

- Let G be a directed a-cyclic graph (DAG).
- Suppose every node has at least one incoming edge.

First things first

Lemma 3.19

If G is a DAG, then G has a node with no incoming edges.

Proof by contradiction.

- Let G be a directed a-cyclic graph (DAG).
- Suppose every node has at least one incoming edge.
- Pick any node v and follow edges backward from v.

First things first

Lemma 3.19

If G is a DAG, then G has a node with no incoming edges.

Proof by contradiction.

- Let G be a directed a-cyclic graph (DAG).
- Suppose every node has at least one incoming edge.
- Pick any node v and follow edges backward from v.
- Repeat until we have gone back $|V|$ times.

First things first

Lemma 3.19

If G is a DAG, then G has a node with no incoming edges.

Proof by contradiction.

- Let G be a directed a-cyclic graph (DAG).
- Suppose every node has at least one incoming edge.
- Pick any node v and follow edges backward from v.
- Repeat until we have gone back $|V|$ times.
- We must now have visited a node twice.
- Thus there is a cycle.

First things first

Lemma 3.19

If G is a DAG, then G has a node with no incoming edges.

Proof by contradiction.

- Let G be a directed a-cyclic graph (DAG).
- Suppose every node has at least one incoming edge.
- Pick any node v and follow edges backward from v.
- Repeat until we have gone back $|V|$ times.
- We must now have visited a node twice.
- Thus there is a cycle.
- Contradiction with G being a-cyclic.
- So there must be a node with no incoming edges.

DAG $\rightarrow$ topological ordering

Lemma 3.20

If G is a DAG, then G has a topological ordering.

Proof.

Idea for the proof: assume G is a DAG, then give a topological ordering by iteratively choosing and removing *a node without incoming edges*. □

Question: What to do, what to do, what to do?

What proof technique can reflect this iterative procedure?

A. by induction
B. by contradiction
C. by division into cases
D. by contraposition
E. by elimination of the implication
F. I don't know

DAG → topological ordering

See also page 102 of the book

Lemma 3.20

If G is a DAG, then G has a topological ordering.

function TopologicalOrdering(G)
  if $|V| = 0$ **then**
    return empty list
  find a node v with no incoming edges
  remainder $\leftarrow$ TopologicalOrdering($G - \{v\}$)
  return v followed by remainder

Proof by induction on $|V| = n$.

DAG $\rightarrow$ topological ordering

See also page 102 of the book

Lemma 3.20

If G is a DAG, then G has a topological ordering.

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.

TU Delft

DAG $\rightarrow$ topological ordering

See also page 102 of the book

Lemma 3.20

If G is a DAG, then G has a topological ordering.

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.
- Induction Hypothesis: If G is a DAG of size $\leq k$, then G has a topological ordering.
- Induction step:

□

Question: Now what?

What do we need to prove here?

TU Delft

DAG $\rightarrow$ topological ordering

See also page 102 of the book

Lemma 3.20

If G is a DAG, then G has a topological ordering.

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.
- Induction Hypothesis: If G is a DAG of size $\leq k$, then G has a topological ordering.
- Induction step:
  - Given a DAG G with $k + 1$ nodes.
  - Something with the IH.
  - Create topological ordering for G
- Thus by induction we have proven the lemma.

□

DAG $\rightarrow$ topological ordering

See also page 102 of the book

Lemma 3.20

If G is a DAG, then G has a topological ordering.

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.
- Induction Hypothesis: If G is a DAG of size $\leq k$, then G has a topological ordering.
- Induction step:
  - Given a DAG G with $k + 1$ nodes.
  - Find a node v with no incoming edges (which exists: Lemma 3.19).
  - $G - \{v\}$ must be a DAG, as removing v cannot create cycles.
  - By the IH, $G - \{v\}$ has a topological ordering T.
  - So v followed by T is a valid topological ordering for G as all edges of v go to are outgoing and thus to nodes later in the ordering.
- Thus by induction we have proven the lemma.

Hang on a second!

Just like Popeye

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.
- Induction Hypothesis: If G is a DAG of size $\leq k$, then G has a topological ordering.
- Induction step:
  - Given a DAG G with $k + 1$ nodes.
  - Find a node v with no incoming vertices.
  - $G - \{v\}$ must be a DAG, as removing v cannot create cycles.
  - By the IH, $G - \{v\}$ has a topological ordering T.
  - So v followed by T is a valid topological ordering for G as v has no incoming edges.
- Thus by induction we have proven the lemma.

□

Question: Hold it!

Is there anything odd about this proof?

Hang on a second!

Just like Popeye

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.
- Induction Hypothesis: If G is a DAG of size $\leq k$, then G has a topological ordering.
- Induction step:
  - Given a DAG G with $k + 1$ nodes.
  - Find a node v with no incoming vertices.
  - $G - \{v\}$ must be a DAG, as removing v cannot create cycles.
  - By the IH, $G - \{v\}$ has a topological ordering T.
  - So v followed by T is a valid topological ordering for G as v has no incoming edges.
- Thus by induction we have proven the lemma.

□

Answer: Strong induction

We use all values smaller than or equal to k. This is *strong* induction.

Hang on a second!

Just like Popeye

Proof by induction on $|V| = n$.

- Base case: holds as for $n = 1$, because G is already a topological ordering.
- Induction Hypothesis: If G is a DAG of size $= k$, then G has a topological ordering.
- Induction step:
  - Given a DAG G with $k + 1$ nodes.
  - Find a node v with no incoming vertices.
  - $G - \{v\}$ must be a DAG, as removing v cannot create cycles.
  - By the IH, $G - \{v\}$ has a topological ordering T.
  - So v followed by T is a valid topological ordering for G as v has no incoming edges.
- Thus by induction we have proven the lemma.

□

All out of Spinach

With a *weak* induction hypothesis, you only use the previous step.

Greedy Algorithms

XKCD Moria: https://xkcd.com/760/

Happy new Quarter!

Problem: Buying Champagne

- You'd like to buy C bottles of a specific champagne to celebrate the start of Q2.
- There are n shops in the neighbourhood that sell this.
- For each shop i, you know the price p_i and the amount of bottles available c_i.

Question: Put on our thinking caps

Where do we buy the wine to minimize buying costs?

A. Visit the shops in order of increasing p_i.
B. Visit the shops in order of decreasing p_i.
C. Visit the shops in order of increasing c_i.
D. Visit the shops in order of decreasing c_i.
E. I don't know.

Just be greedy!

sort the shops by price, so that $p_1 \leq p_2 \leq \cdots \leq p_n$

$A \leftarrow \emptyset$

for $k = 1 \rightarrow n$ **do**

  $A \leftarrow A \cup \{k\}$

  if $c_k \geq C$ **then**

    $\text{buy}_k \leftarrow C$

    return A

  else

    $\text{buy}_k \leftarrow c_k$

    $C \leftarrow C - c_k$

Question: Runtime?

What is the tightest worst-case upper bound on the runtime?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$
E. $O(n^2)$
F. I don't know.

TU Delft

Just be greedy!

sort the shops by price, so that $p_1 \leq p_2 \leq \cdots \leq p_n$

$A \leftarrow \emptyset$

for $k = 1 \rightarrow n$ **do**

  $A \leftarrow A \cup \{k\}$

  if $c_k \geq C$ **then**

    $\text{buy}_k \leftarrow C$

    return A

  else

    $\text{buy}_k \leftarrow c_k$

    $C \leftarrow C - c_k$

Question: Runtime?

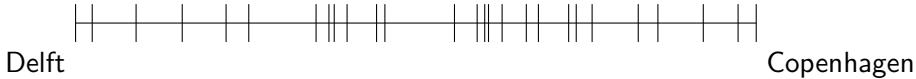
What is the tightest worst-case upper bound on the runtime?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$ Using Merge Sort
E. $O(n^2)$
F. I don't know.

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

Selecting Breakpoints

No not when debugging

Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

Question: Which campsites to chose?

A. Stay the night at every campsite on your route.

B. Select $\lceil L/C \rceil$ campsites at approximately equal distances.

C. Cycle until the last possible campsite each day.

D. I don't know.

Selecting Breakpoints

No not when debugging

Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

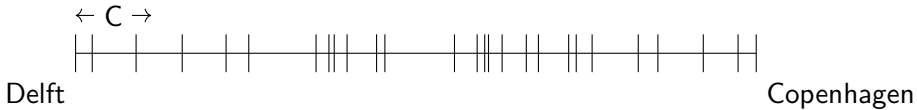
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

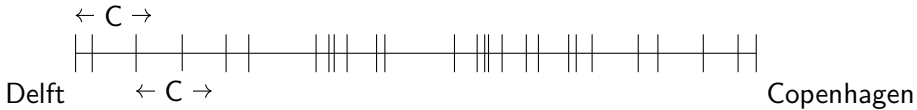
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

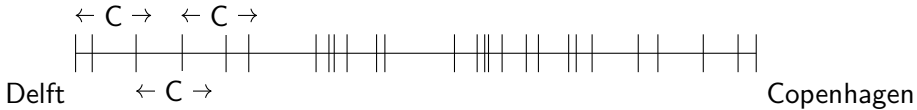
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

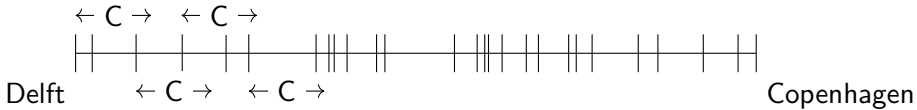
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

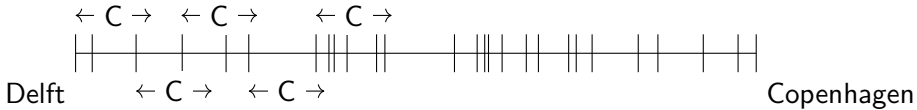
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

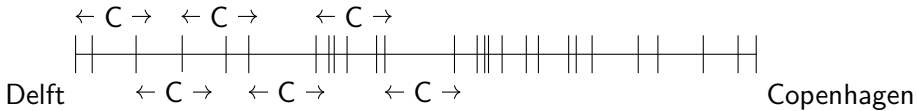
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

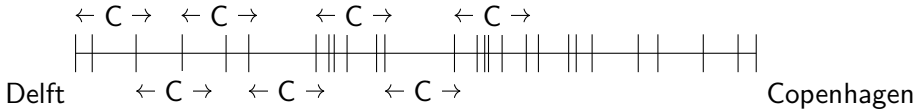
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

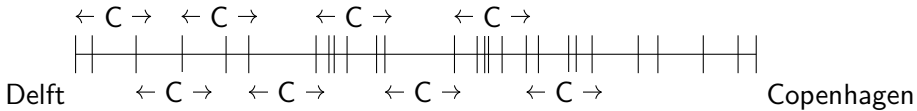
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

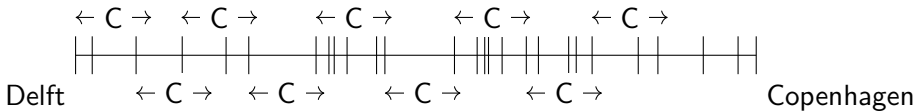
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

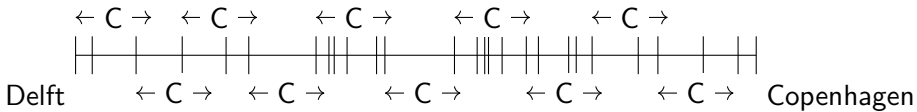
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

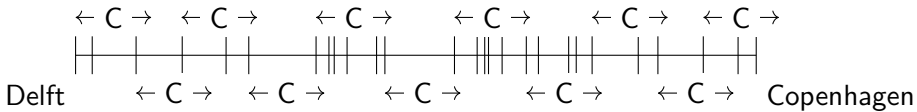
Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

Selecting Breakpoints

No not when debugging



Problem: Selecting breakpoints

- You're cycling from Delft to Copenhagen along a route of length L.
- You can camp at certain points with distances b_1 to b_n from Delft.
- You can cycle at most C kilometers per day.
- Goal: cycle there in as few days as possible.

Answer: Greedy Algorithm

Go as far as you can, every day!

TU Delft

The algorithm

sort the breakpoints, so that $0 = b_1 \leq b_2 \leq \cdots \leq b_n$
$b_n \leftarrow L$
$S \leftarrow \{0\}$ ▷ Selected campsites
$x \leftarrow 0$ ▷ Current location
while $x \neq b_n$ **do**
    let p be the largest integer $\leq n$ such that $b_p \leq x + C$
    if $b_p = x$ **then**
        return No solution
    $x \leftarrow b_p$
    $S \leftarrow S \cup \{p\}$
return S

Question: Runtime?

What is the tightest worst-case upper bound on the runtime?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$
E. $O(n^2)$
F. I don't know.

TU Delft

The algorithm

sort the breakpoints, so that $0 = b_1 \leq b_2 \leq \cdots \leq b_n$
$b_n \leftarrow L$
$S \leftarrow \{0\}$ ▷ Selected campsites
$x \leftarrow 0$ ▷ Current location
while $x \neq b_n$ **do**
    let p be the largest integer $\leq n$ such that $b_p \leq x + C$
    if $b_p = x$ **then**
        return No solution
    $x \leftarrow b_p$
    $S \leftarrow S \cup \{p\}$
return S

Question: Runtime?

What is the tightest worst-case upper bound on the runtime?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$ Using Merge Sort
E. $O(n^2)$
F. I don't know.

But is it optimal?

Proving techniques

1. Proof by induction: Greedy stays ahead
   - With equal number of stops, Greedy is at least as close to goal (Copenhagen) as optimal solution (Lemma)
   - Greedy thus is optimal (Theorem)
2. Or, proof by contradiction: reason about optimal solution most similar to greedy

TU Delft

But is it optimal?

Proving techniques

1. Proof by induction: Greedy stays ahead
   - With equal number of stops, Greedy is at least as close to goal (Copenhagen) as optimal solution (Lemma)
   - Greedy thus is optimal (Theorem)
2. Or, proof by contradiction: reason about optimal solution most similar to greedy

Notation

- Let $0 = g_1 < g_2 < \ldots g_p = L$ denote the campsites (distance) chosen by greedy.
- Let $0 = f_1 < f_2 < \ldots f_q = L$ denote the campsites in an optimal solution.

TU Delft

Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

□

Question: Proofs by induction

What are the basic elements of a proof by induction?

Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$):
- *Induction Hypothesis*:
- *Induction step*:

□

TU Delft

Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*:
- *Induction step*:

□

TU Delft

Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*: Suppose that for some k it holds that $f_k \leq g_k$.
- *Induction step*: *To prove*: for $k + 1$ it holds that $f_{k+1} \leq g_{k+1}$.

□

TU Delft

Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*: Suppose that for some k it holds that $f_k \leq g_k$.
- *Induction step*: *To prove*: for $k + 1$ it holds that $f_{k+1} \leq g_{k+1}$.

□

Question: Induction step

Now what?

TU Delft

Proof by induction: Greedy stays ahead

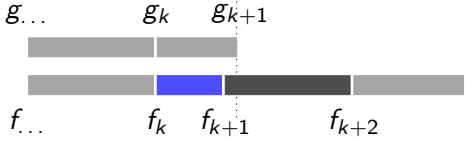
Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*: Suppose that for some k it holds that $f_k \leq g_k$.
- *Induction step*: *To prove*: for $k + 1$ it holds that $f_{k+1} \leq g_{k+1}$.
  - We know that $f_k \leq g_k$ by the IH.

□



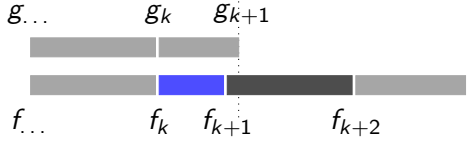
Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*: Suppose that for some k it holds that $f_k \leq g_k$.
- *Induction step*: *To prove*: for $k + 1$ it holds that $f_{k+1} \leq g_{k+1}$.
  - We know that $f_k \leq g_k$ by the IH.
  - Greedy selects g_{k+1} to be the campsite within reach as far from g_k as possible.
  - Campsite f_{k+1} cannot be further than the farthest reachable from f_k, and this is definitely within reach from g_k because of the IH. So $f_{k+1} \leq g_{k+1}$. □



TU Delft

Proof by induction: Greedy stays ahead

Lemma

$\forall r : f_r \leq g_r$, with g for Greedy campsites and f for some optimal solution.

Proof by induction.

- *Base case* ($r = 1$): When $r = 1$, g_1 is as far away as possible, i.e., $f_1 \leq g_1$.
- *Induction Hypothesis*: Suppose that for some k it holds that $f_k \leq g_k$.
- *Induction step*: *To prove*: for $k + 1$ it holds that $f_{k+1} \leq g_{k+1}$.
  - We know that $f_k \leq g_k$ by the IH.
  - Greedy selects g_{k+1} to be the campsite within reach as far from g_k as possible.
  - Campsite f_{k+1} cannot be further than the farthest reachable from f_k, and this is definitely within reach from g_k because of the IH. So $f_{k+1} \leq g_{k+1}$.

□

Question: So Greedy is optimal?

But why does this prove that Greedy selects *fewer* campsites?

But is it optimal?

Using the "greedy stays ahead" lemma

Theorem

The greedy algorithm is optimal.

Proof by contradiction.

- Let k be the number of campsites selected by greedy, and m the number of campsites in an optimal solution f.

□

[a] A Lemma is like a sub-routine.

But is it optimal?

Using the "greedy stays ahead" lemma

Theorem

The greedy algorithm is optimal.

Proof by contradiction.

- Let k be the number of campsites selected by greedy, and m the number of campsites in an optimal solution f.
- Suppose Greedy is not optimal. So $k > m$ and thus $g_m < f_m$.

□

[a] A Lemma is like a sub-routine.

But is it optimal?

Using the "greedy stays ahead" lemma

Theorem

The greedy algorithm is optimal.

Proof by contradiction.

- Let k be the number of campsites selected by greedy, and m the number of campsites in an optimal solution f.
- Suppose Greedy is not optimal. So $k > m$ and thus $g_m < f_m$.
- However, $f_m \leq g_m$ with the Lemma from the previous slide.[a]
- Contradiction with $g_m < f_m$. So Greedy must be optimal.

□

[a] A Lemma is like a sub-routine.

You are here

The course so far

- Course organisation

Today's content

- Checking your ADS/R&L knowledge – algorithms with proofs
- An introduction to Greedy algorithms – also with proofs

The future

- More greedy algorithms
- Divide & Conquer algorithms
- Dynamic programming
- Network Flow

What is still unclear?

Question: After every lecture...

Give us some homework and tell us:
What is still unclear after attending today's lecture?

TU Delft

Homework for this week

- Before next lecture:
  - Complete module "Welcome to Algorithm Design" (i.e. prerequisites)
  - Do all skills of module Greedy at least until "Lecture 2" (for your chosen path)

  For example, do this during the lab session on Thursday!
  Register for the Acing AD session via the Queue:
  `https://queue.tudelft.nl/lab/7543`.

TU Delft

Homework for this week

- Before next lecture:
  - Complete module "Welcome to Algorithm Design" (i.e. prerequisites)
  - Do all skills of module Greedy at least until "Lecture 2" (for your chosen path)

    For example, do this during the lab session on Thursday!
    Register for the Acing AD session via the Queue:
    `https://queue.tudelft.nl/lab/7543`.
- Next TA check:
  - *Experiments!* (Mountain Climber): November 23
  - *Greedy Triathlon*: November 24 (deadline Nov 25)
- Next peer review:
  - November 23 (prepare at home, review during lab)

TU Delft

CSE2310 Algorithm Design

Lecture 1: Course introduction & Greedy Algorithms

Stefan Hugtenburg, Emir Demirović, and Mathijs de Weerdt

Algorithmics group — EEMCS — TU Delft

2023–2024 Q2

TU Delft