CS 4820 Analysis of Algorithms

Instructor

Prof. Michael Kim

Plan for today

1. Motivation for 4820
2. Brief Administrivia
3. Case Study: Tiling Problems

Motivation

Algorithms is the study of how to solve computational problems

Computational Problem Examples

* Given a road map, find the shortest path from A to B
* Given a list of integers, return them in sorted order.
* Given a social network, find a large clique
  (mutually-connected accounts)

Can we solve these problems? Can we solve them efficiently?

An algorithm is a well-defined sequence of steps to solve a computational problem

A computational "recipe"

* precise inputs / outputs
* language agnostic

Why 4820?

Why 4820?

* It's a requirement.
  → Why? Algorithms show up in every area of CS.

Why 4820?

* It's a requirement.
  → Why? Algorithms show up in every area of CS.

* The "algorithmic lens" on other fields
  → Algorithms give new perspectives in
  Econ, Bio, Physics, Sociology, Linguistics

Why 4820?

* It's a requirement.
  → Why? Algorithms show up in every area of CS.

* The "algorithmic lens" on other fields
  → Algorithms give new perspectives in Econ, Bio, Physics, Sociology, Linguistics

* Understanding the Laws of Nature
  → Algorithmic analysis reveals nature of computation. (What is / is not possible?)

Why 4820?

* It's a requirement.
  → Why? Algorithms show up in every area of CS.

* The "algorithmic lens" on other fields
  → Algorithms give new perspectives in Econ, Bio, Physics, Sociology, Linguistics

* Understanding the Laws of Nature
  → Algorithmic analysis reveals nature of computation. (What is / is not possible?)

* Algorithmic Thinking is problem solving
  → Designing algorithms requires creativity
  → Analyzing algorithms requires clarity of thought
  ↓
  Precise definitions & mathematical proofs.

Guiding Philosophy

For many computational problems there is a simple, but inefficient algorithm.

Guiding Philosophy

For many computational problems there is a simple, but inefficient algorithm.

eg. Shortest Path from A to B:

- Iterate through every path in the road map.
- If the path connects A to B,
  - ↳ record the path and its distance
- Return the A→B path of minimum distance

Guiding Philosophy

For many computational problems there is a simple, but inefficient algorithm.

e.g. Shortest Path from A to B:

- Iterate through every path in the road map.
- If the path connects A to B,
  - ↳ record the path and its distance
- Return the A→B path of minimum distance

Concern. There are exponentially many paths in a map.

As the map grows bigger,

Brute force is prohibitively expensive!

Guiding Philosophy

For many computational problems there is a simple, but inefficient algorithm.

e.g. Shortest Path from A to B:

- Iterate through every path in the road map.
- If the path connects A to B,
  - ↳ record the path and its distance
- Return the A→B path of minimum distance

Key Question: Can we do better?

Intermission: Administrivia

All information @ course website

cs.cornell.edu/courses/cs4820/2025sp

↳ Ed Discussions & Gradescope

↳ Course Staff Office Hours

* Exams
  - Prelim #1, 13 Feb 7:30p
  - Prelim #2, 27 Mar 7:30p
  - Final Exam, Finals Week TBD.

* Weekly Homework.
  - Released on Wed after lecture
  - Due Following Tues.
    ↳ "Grace Period" til Wed AM.

* No slip days
* No grace period on the grace period

Recommended (but optional)

HW 0 Released Today

* Participating in Class

- Ask questions!
  - ↳ In lecture
  - ↳ On Ed Discussions
  - ↳ At Office Hours

But also, Make space for your peers to ask/answer questions

* Office Hours

- Best place to get help learning 4820 material
- HW help (not answers) from peers & professor.
  - ↳ 39 TAs!

4820 is an Inclusive & Safe space for learning.

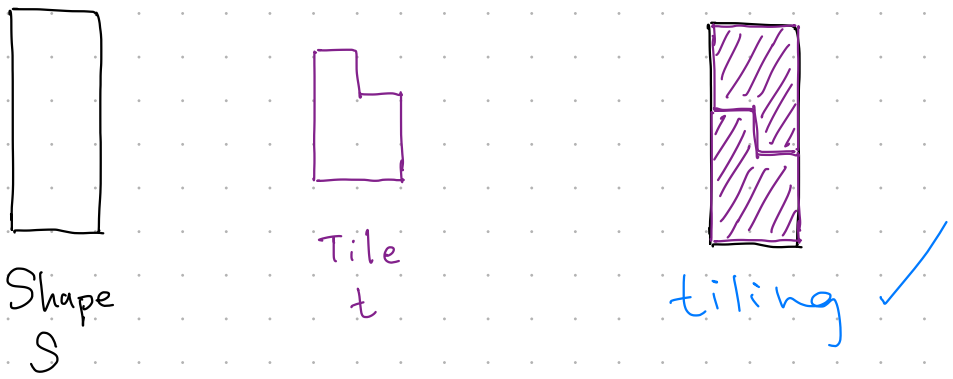
---

* You belong here
* You deserve respect from your peers & the course staff
* We expect you to respect your peers & the course staff.

For personal concerns
↳ contact cs4820sp25@gmail.com
↳ Set up a meeting w/ Prof. Kim (on course site)

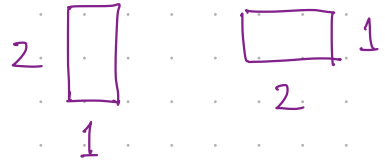
Case Study in Computational Problems: Tiling Problems

Case Study in Computational Problems: Tiling Problems



Definition. A tiling of a shape S with tile t is an arrangement of non-overlapping copies of t that completely covers S.

Example #1. Domino-Tiling

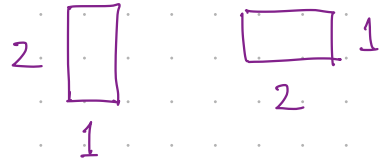
* Dominos are 2x1 tiles 

The Domino-Tiling Problem.

Given Shape S, can S be tiled with Dominos?

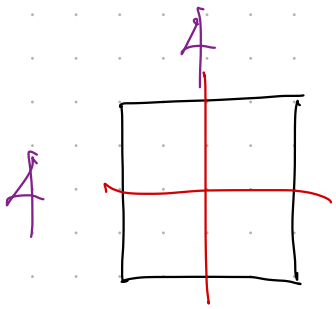
Example #1. Domino-Tiling

* Dominos are 2x1 tiles



The Domino-Tiling Problem.

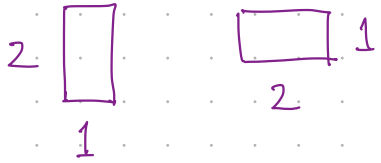
Given Shape S, can S be tiled with Dominos?



S: 4 x 4 square

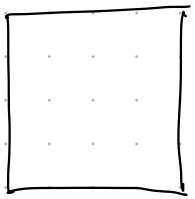
Example #1. Domino-Tiling

* Dominos are 2×1 tiles



The Domino-Tiling Problem.

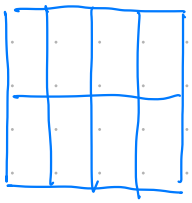
Given Shape S, can S be tiled with Dominos?



S: 4×4 square

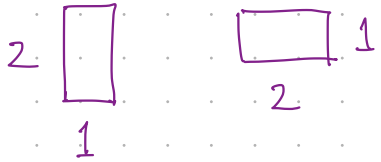
Claim. S can be Domino-tiled.

Pf. By construction.



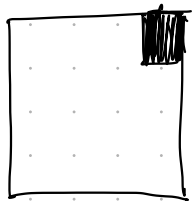
Example #1. Domino-Tiling

* Dominos are 2x1 tiles



The Domino-Tiling Problem.

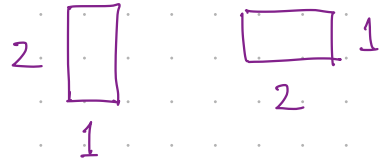
Given Shape S, can S be tiled with Dominos?



S: 4x4 square w/ corner removed

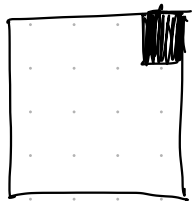
Example #1. Domino-Tiling

* Dominos are 2x1 tiles



The Domino-Tiling Problem.

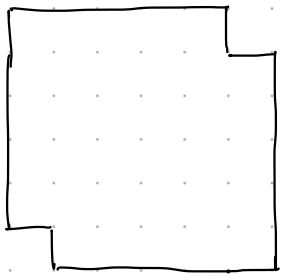
Given Shape S, can S be tiled with Dominos?



S': 4x4 square w/ corner removed

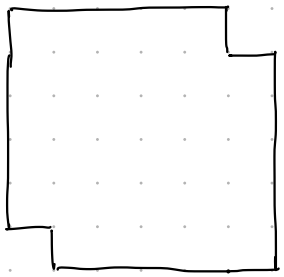
Claim. S' CANNOT be Domino-Tiled

Pf. By Parity Argument. (odd vs. even)



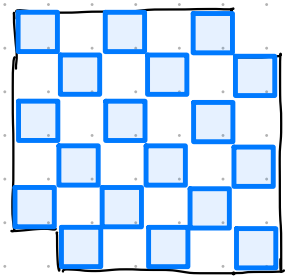
S" 6x6 square w/ opposing corners removed.

Can S" be Domino-Tiled?



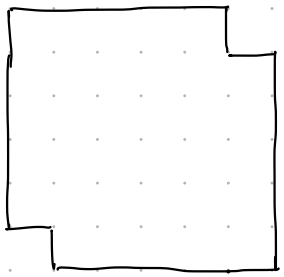
S" 6x6 square w/ opposing corners removed.

Claim: S" CANNOT be Domino-Tiled.



Pf. By coloring argument.

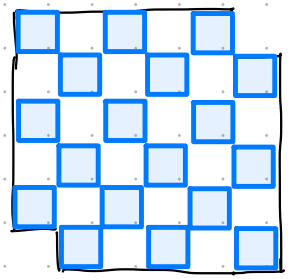
- Every Domino covers exactly 1 white & 1 Blue square



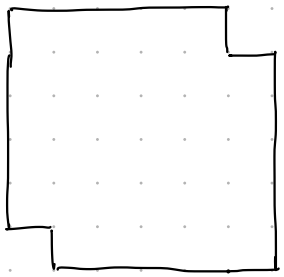
"6x6 square w/ opposing corners removed.

Claim: "CANNOT be Domino-Tiled.

Pf. By coloring argument.

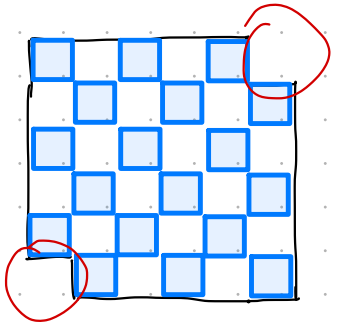


- Every Domino covers exactly 1 white & 1 Blue square
- Thus, every shape that can be Domino-Tiled has an equal number of White & Blue squares.



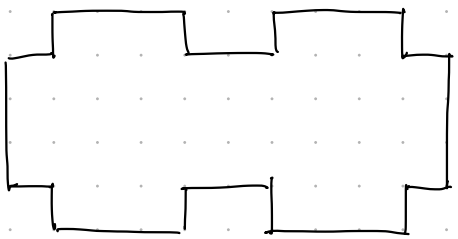
S" 6x6 square w/ opposing corners removed.

Claim: S" CANNOT be Domino-Tiled.

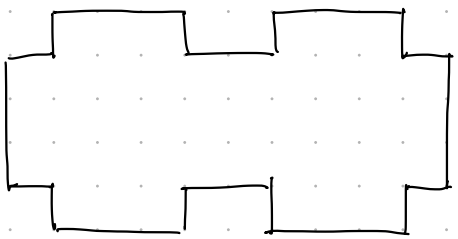


Pf. By coloring argument.

- Every Domino covers exactly 1 white & 1 Blue square
- Thus, every shape that can be Domino-Tiled has an equal number of White & Blue squares.
- But S" has 16 White squares & 18 Blue squares ▨



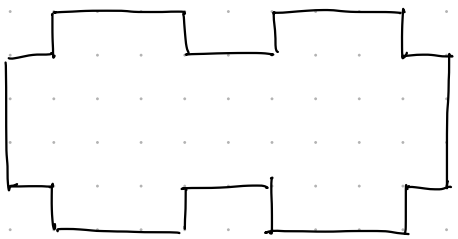
What about other shapes?



What about other shapes?

Theorem. There is an efficient algorithm DOMINO TILE that solves the Domino-Tiling Problem.

See HW4



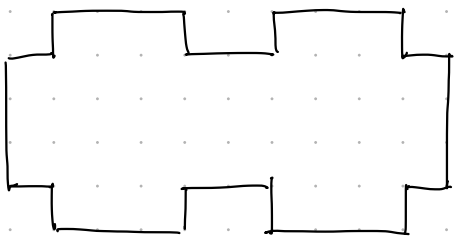
What about other shapes?

Theorem. There is an efficient algorithm DOMINOTILE that solves the Domino-Tiling Problem.

DOMINOTILE is CORRECT

For every shape S, DOMINOTILE outputs a valid tiling if and only if S can be domino-tiled.

(i.e. if DOMINOTILE says S cannot be domino-tiled, No valid tiling exists)



What about other shapes?

Theorem. There is an efficient algorithm DOMINO TILE that solves the Domino-Tiling Problem.

DOMINO TILE is EFFICIENT

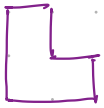
Let $t(n) \equiv$ Time steps DOMINO TILE executes on shapes S of area n.

Then, there exists $c \in \mathbb{N}$ s.t. $t(n) \leq O(n^c)$.

DOMINO TILE runs in polynomial time.

Example #2. Tromino - Tiling

Example #2. Tromino - Tiling

t = L-shaped trominos 

The Tromino - Tiling Problem.

Given Shape S, can S be tiled with L-shaped Trominos?

Example #2. Tromino-Tiling

t = L-shaped trominos 

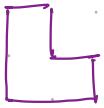
The Tromino-Tiling Problem.

Given Shape S, can S be tiled with L-shaped Trominos?

Q: Is there an efficient algorithm that solves Tromino-Tiling?

VOTE

Example #2. Tromino-Tiling

t = L-shaped trominos 

The Tromino-Tiling Problem.

Given Shape S, can S be tiled with L-shaped Trominos?

Q: Is there an efficient algorithm that solves Tromino-Tiling?

A: No one knows! MAJOR OPEN RESEARCH QUESTION

If you give a polynomial-time algorithm for Tromino-Tiling, then...

* You receive an A+ in 4820 (from Prof. Kim)

If you give a polynomial-time algorithm for Tromino-Tiling, then...

* You receive an A+ in 4820 (from Prof. Kim)

* You win $1M (Not from Prof. Kim)

If you give a polynomial-time algorithm for Tromino-Tiling, then...

* You receive an A+ in 4820 (from Prof. Kim)

* You win $1M (Not from Prof. Kim)

* You break all of modern cryptography

(No more blockchain OR secure Internet)

If you give a polynomial-time algorithm for Tromino-Tiling, then...

* You receive an A+ in 4820 (from Prof. Kim)

* You win $1M (Not from Prof. Kim)

* You break all of modern cryptography

  (No more blockchain OR secure Internet)

* Your algorithm implies super-efficient optimization

  (e.g., for training AI models)

The Tromino - Tiling Problem.

Given Shape S, can S be tiled with L-shaped Trominos?

Theorem. Tromino-Tiling is NP-Hard (NP-Complete)

Solving Tromino-Tiling is the infamous

P vs. NP

problem, lurking in disguise.

See HW 6

Example #3. Tiling the Plane

Example #3. Tiling the Plane

Tiling-the-Plane Problem.

Given a finite collection of tiles $T = \{t_1, \ldots, t_k\}$ can the infinite 2D-grid be tiled using T?

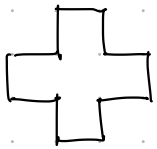
infinitely many copies of any $t \in T$.

Example #3. Tiling the Plane

Tiling-the-Plane Problem

Given a finite collection of tiles $T = \{t_1, \ldots, t_k\}$ can the infinite 2D-grid be tiled using T?

(using: infinitely many copies of any $t \in T$.)

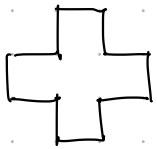
e.g. Can the plane be tiled w/  ?

Example #3. Tiling the Plane

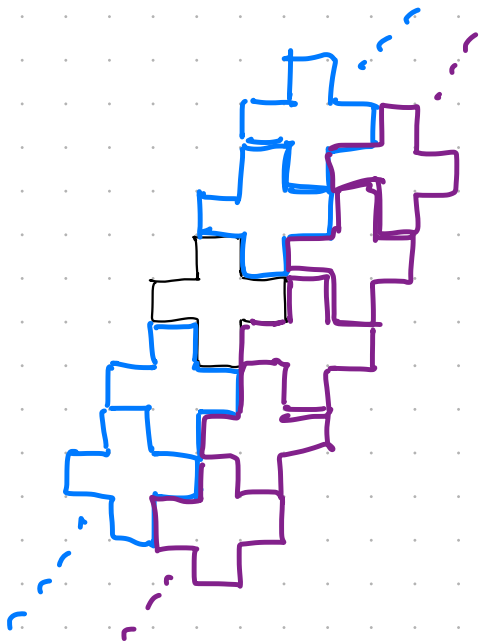
Tiling-the-Plane Problem

Given a finite collection of tiles $T = \{t_1, \ldots, t_k\}$ can the infinite 2D-grid be tiled using T?

(using: infinitely many copies of any $t \in T$.)

e.g. Can the plane be tiled w/  ?

A: Yes!



Tiling-the-Plane Problem

Given a finite collection of tiles $T = \{t_1, \ldots, t_k\}$ can the infinite 2D-grid be tiled using T?

Q: What is the best algorithm that solves Tiling-The-Plane?

Tiling-the-Plane Problem

Given a finite collection of tiles $T = \{t_1, \ldots, t_k\}$ can the infinite 2D-grid be tiled using T?

Q: What is the best algorithm that solves Tiling-The-Plane?

Theorem. There does NOT exist any algorithm that solves the Tiling-The-Plane Problem.

Tiling-The-Plane is **Undecidable**.

Why do we care about problems that can't be solved?

Why do we care about problems that can't be solved?

The Check-GPT Problem.

* Write an inefficient algorithm A for 4820 homework
* Ask GPT to return an efficient algorithm A^* that solves the same problem as A.
* Return True iff A and A^* solve the same problem.

Why do we care about problems that can't be solved?

The Check-GPT Problem.

* Write an inefficient algorithm A for 4820 homework
* Ask GPT to return an efficient algorithm A^* that solves the same problem as A.
* Return True iff A and A^* solve the same problem.

Theorem. Check-GPT is Undecidable!

No algorithm (current or future) can reliably check the output of AI for correctness.

See HW 9