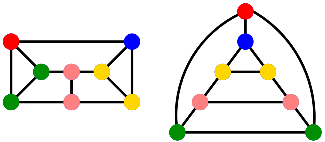
Module 7 project:
Graph Isomorphism

Part II: Branching Algorithms



Ruben Hoeksma, slides partly by Paul Bonsma

Summary of the Previous Video

Recall the *Graph Isomorphism (GI) Problem*, and the *color refinement* algorithm to *efficiently* compute a *coarsest stable coloring* of a given graph.

Definition

A coloring α of G is stable *if for all $u, v \in V(G)$ it holds that:*
if $\alpha(u) = \alpha(v)$, then u and v have identically colored neighborhoods.

Key fact:

Theorem

For every graph G, there is a unique coarsest stable partition of $V(G)$, which is the partition given by color refinement.

Summary of the Previous Video

Recall the *Graph Isomorphism (GI) Problem*, and the *color refinement* algorithm to *efficiently* compute a *coarsest stable coloring* of a given graph.

Definition

A coloring α of G is stable *if for all $u, v \in V(G)$ it holds that:*
if $\alpha(u) = \alpha(v)$, then u and v have identically colored neighborhoods.

Key fact:

Theorem

For every graph G and partition π_0 of $V(G)$*, there is a unique coarsest stable partition of $V(G)$* that refines π_0*, which is the partition given by color refinement*, when starting with π_0.

Color Refinement Invariant

A useful invariant that follows from last lecture:

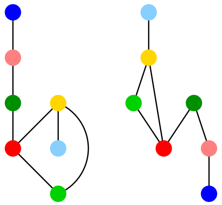
Proposition (Invariant)

Let α be a coloring of a graph $G \uplus H$, and let β be the stable coloring of $G \uplus H$ given by color refinement, when starting with α.
Then any isomorphism $f : V(G) \to V(H)$ that is color preserving for α is also color preserving for β.

Possible choices for the "start coloring" α are *uniform coloring* or coloring by *vertex degree*, but today we will also see alternatives.

Color Refinement Outcome: Example

Let α be the stable coloring of $G \uplus H$ that results from applying color refinement, starting with the *uniform coloring*.

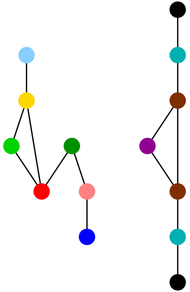


Observation

If α defines a bijection $f : V(G) \to V(H)$, then f is the unique *isomorphism from G to H.*

Color Refinement Outcome: Example

Let $\color{red}{\alpha}$ be the stable coloring of $G \uplus H$ that results from applying color refinement, starting with the *uniform coloring*.

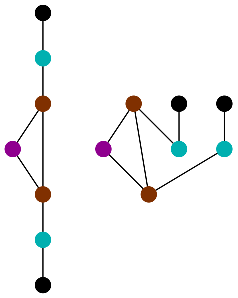


Observation

If α is unbalanced, then there is no isomorphism from G to H.

Color Refinement Outcome: Example

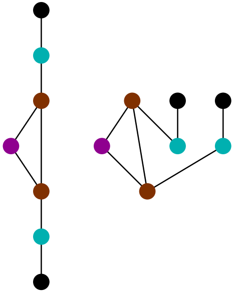
Let α be the stable coloring of $G \uplus H$ that results from applying color refinement, starting with the *uniform coloring*.



Q: But what if the coarsest stable coloring α is balanced, but does not define a bijection?

Color Refinement Outcome: Example

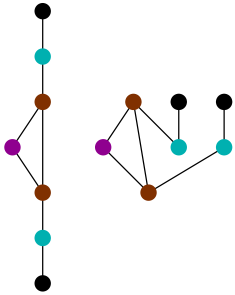
Let α be the stable coloring of $G \uplus H$ that results from applying color refinement, starting with the *uniform coloring*.



Q: But what if the coarsest stable coloring α is balanced, but does not define a bijection?

Idea: Try all color preserving bijections f. Here: $2 \cdot 2 \cdot 2 = 8$ possibilities.

Better than Enumeration: Recursive Color Refinement



Better idea: Choose a vertex x of G for which $f(x)$ is not yet clear.
"Fix" which vertex y of H it should be mapped to. Give x and y a new color, and apply color refinement, again.
(Every guess corresponds to a branch of a *recursive algorithm*.)

In the example: only two guesses necessary.

Individualization Refinement - Overview

- α: balanced stable coloring of $G \uplus H$, does not define a bijection.

Individualization Refinement - Overview

- α: balanced stable coloring of $G \uplus H$, does not define a bijection.
- Exists color class C with $2k$ vertices, $k \geq 2$ (k vertices of G and k vertices of H). Denote $C_G = C \cap V(G)$ and $C_H = C \cap V(H)$.

Individualization Refinement - Overview

- α: balanced stable coloring of $G \uplus H$, does not define a bijection.
- Exists color class C with $2k$ vertices, $k \geq 2$ (k vertices of G and k vertices of H). Denote $C_G = C \cap V(G)$ and $C_H = C \cap V(H)$.
- Choose $x \in C_G$. Every color preserving isomorphism f has $f(x) = y$ for some $y \in C_H$.

Individualization Refinement - Overview

- α: balanced stable coloring of $G \uplus H$, does not define a bijection.
- Exists color class C with $2k$ vertices, $k \geq 2$ (k vertices of G and k vertices of H). Denote $C_G = C \cap V(G)$ and $C_H = C \cap V(H)$.
- Choose $x \in C_G$. Every color preserving isomorphism f has $f(x) = y$ for some $y \in C_H$.
- Try out all such possibilities for $f(x)$ (gives k *branches* of a recursive algorithm): *There exists a color preserving isomorphism if and only if a color preserving isomorphism will be found in at least one branch.*

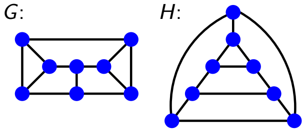
Individualization Refinement - Overview

- α: balanced stable coloring of $G \uplus H$, does not define a bijection.
- Exists color class C with $2k$ vertices, $k \geq 2$ (k vertices of G and k vertices of H). Denote $C_G = C \cap V(G)$ and $C_H = C \cap V(H)$.
- Choose $x \in C_G$. Every color preserving isomorphism f has $f(x) = y$ for some $y \in C_H$.
- Try out all such possibilities for $f(x)$ (gives k *branches* of a recursive algorithm): *There exists a color preserving isomorphism if and only if a color preserving isomorphism will be found in at least one branch.*
- In each branch, the *choice* $f(x) = y$ is encoded by giving both vertices a *new, unique color*.

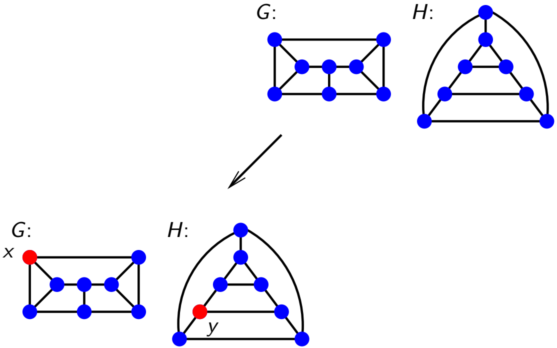
Individualization Refinement - Overview

- α: balanced stable coloring of $G \uplus H$, does not define a bijection.
- Exists color class C with $2k$ vertices, $k \geq 2$ (k vertices of G and k vertices of H). Denote $C_G = C \cap V(G)$ and $C_H = C \cap V(H)$.
- Choose $x \in C_G$. Every color preserving isomorphism f has $f(x) = y$ for some $y \in C_H$.
- Try out all such possibilities for $f(x)$ (gives k *branches* of a recursive algorithm): *There exists a color preserving isomorphism if and only if a color preserving isomorphism will be found in at least one branch.*
- In each branch, the *choice* $f(x) = y$ is encoded by giving both vertices a *new, unique color*.
- Given this start coloring, we can apply color refinement again, and continue recursively until either an isomorphism is found, or it is concluded that no isomorphism with $f(x) = y$ exists.

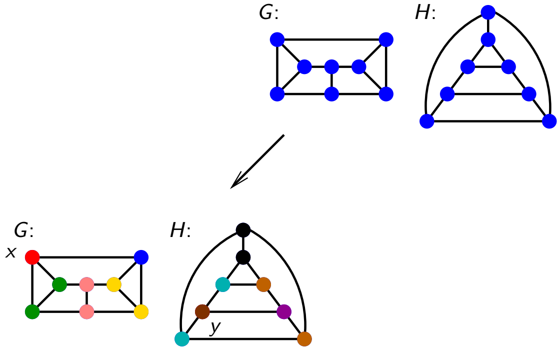
Example



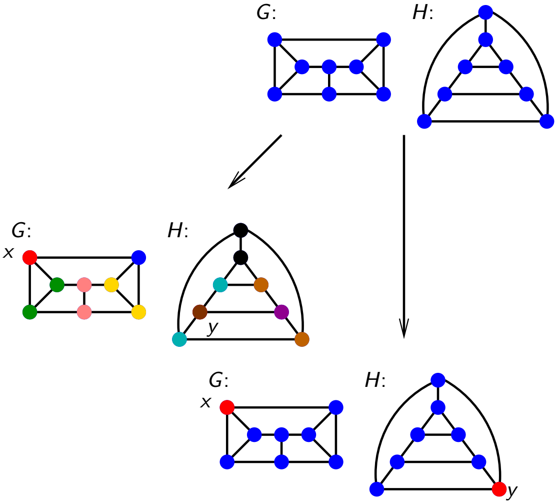
Example



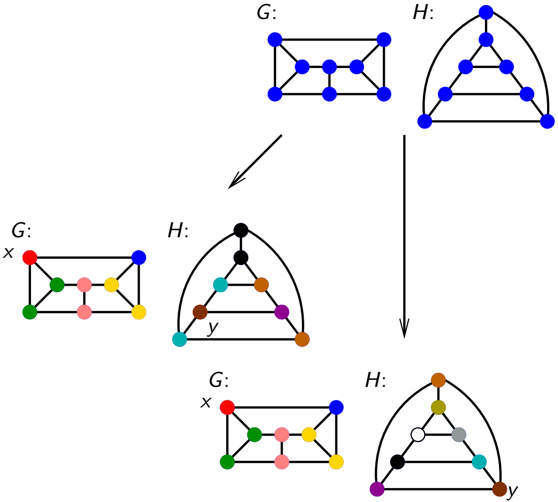
Example



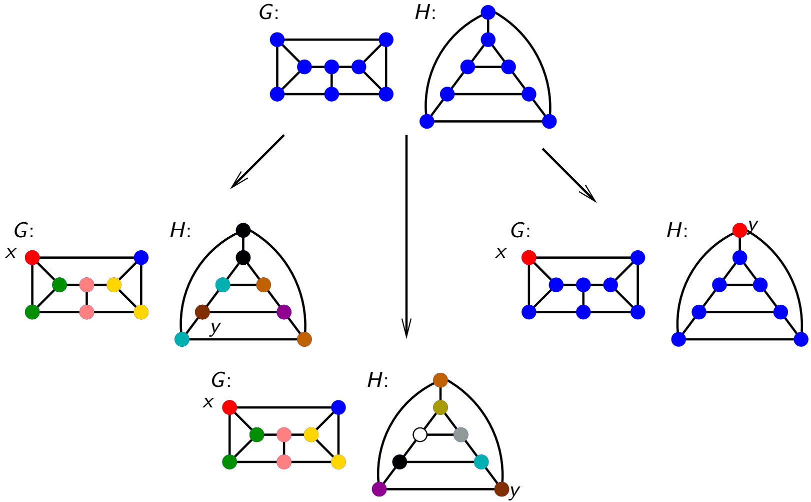
Example



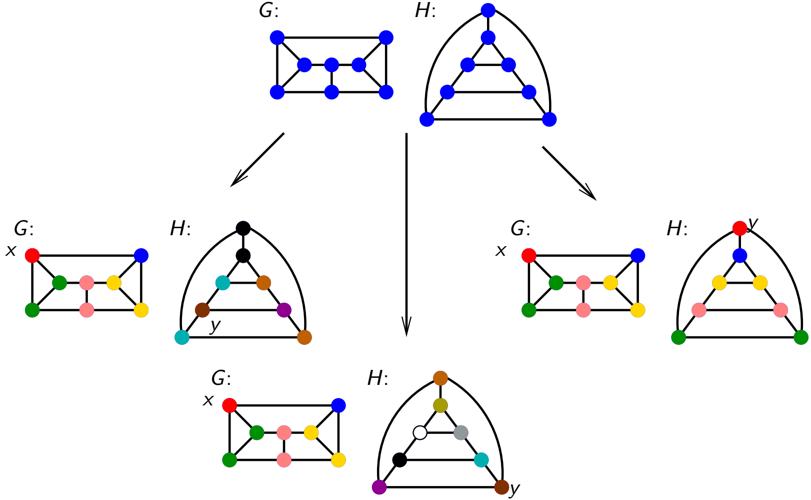
Example



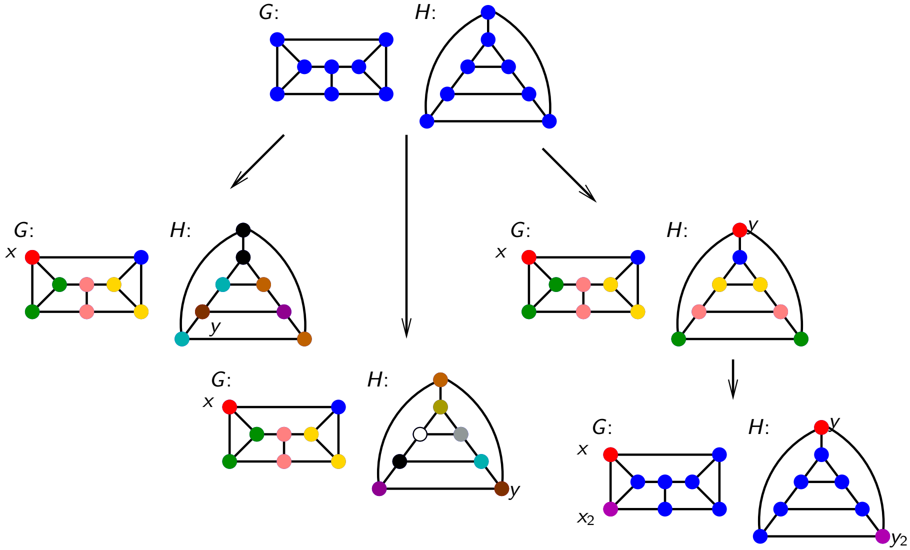
Example



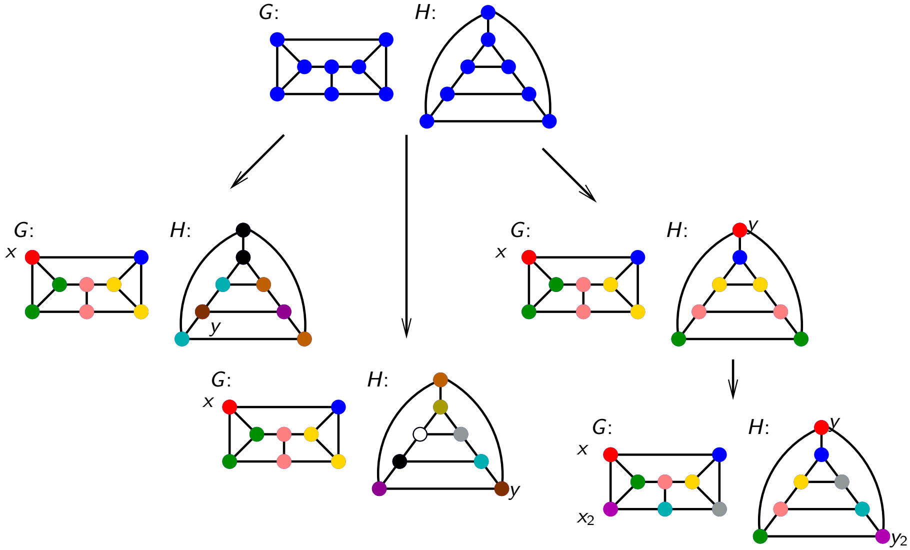
Example



Example



Example



Individualization Refinement - Implementation Details

- Let $D = (x_1, \dots, x_d)$ and $I = (y_1, \dots, y_d)$ be sequences of distinct vertices of G and H, respectively.
- A bijection $f : V(G) \to V(H)$ *follows D, I* if for all i: $f(x_i) = y_i$.
- Let $\alpha(D, I)$ be the coloring of $G \uplus H$ that assigns color i to x_i and y_i for $i \in \{1, \dots, d\}$, and color 0 to all vertices not in $D \cup I$.

Individualization Refinement - Implementation Details

- Let $D = (x_1, \dots, x_d)$ and $I = (y_1, \dots, y_d)$ be sequences of distinct vertices of G and H, respectively.
- A bijection $f : V(G) \to V(H)$ *follows* D, I if for all i: $f(x_i) = y_i$.
- Let $\alpha(D, I)$ be the coloring of $G \uplus H$ that assigns color i to x_i and y_i for $i \in \{1, \dots, d\}$, and color 0 to all vertices not in $D \cup I$.

Proposition

Let β be the coarsest stable coloring of $G \uplus H$ that refines $\alpha(D, I)$.
Every isomorphism f from G to H that follows D, I is β-color preserving.

Individualization Refinement - Implementation Details

- Let $D = (x_1, \ldots, x_d)$ and $I = (y_1, \ldots, y_d)$ be sequences of distinct vertices of G and H, respectively.
- A bijection $f : V(G) \to V(H)$ *follows* D, I if for all i: $f(x_i) = y_i$.
- Let $\alpha(D, I)$ be the coloring of $G \uplus H$ that assigns color i to x_i and y_i for $i \in \{1, \ldots, d\}$, and color 0 to all vertices not in $D \cup I$.

Proposition

Let β be the coarsest stable coloring of $G \uplus H$ that refines $\alpha(D, I)$.
Every isomorphism f from G to H that follows D, I is β-color preserving.

Corollary

If β unbalanced, $\nexists$ isomorphism from G to H following D, I. If β defines bijection f, f is the unique *isomorphism from G to H following D, I.*

Algorithm 2: Individualization Refinement

Subroutine $\textsc{CountIsomorphism}(D, I)$:
$\textsc{Input}$: D and I are equal length sequences of vertices of G resp. H.
$\textsc{Output}$: The number of isomorphisms from G to H that follow D, I.

Compute the coarsest stable coloring β of $G \uplus H$ that refines $\alpha(D, I)$
if β is unbalanced:
    return 0
if β defines a bijection:
    return 1
Choose a color class C with $|C| \geq 4$.
Choose $x \in C \cap V(G)$.
num $= 0$
for all $y \in C \cap V(H)$:
    num $:=$ num$+\textsc{CountIsomorphism}(D + x, I + y)$
return num

Correctness of Individualization Refinement

Theorem

Algorithm 2 is correct; it returns the number of isomorphisms from G to H that follow D, I.

Correctness of Individualization Refinement

Theorem

Algorithm 2 is correct; it returns the number of isomorphisms from G to H that follow D, I.

Proof Sketch: By induction over the recursion tree:
Base: the unbalanced / bijection case (use the corollary).
Induction step: use the previous proposition. $\square$

Correctness of Individualization Refinement

Theorem

Algorithm 2 is correct; it returns the number of isomorphisms from G to H that follow D, I.

Proof Sketch: By induction over the recursion tree:
Base: the unbalanced / bijection case (use the corollary).
Induction step: use the previous proposition. □

Theorem

Algorithm 2 terminates.

Correctness of Individualization Refinement

Theorem

Algorithm 2 is correct; it returns the number of isomorphisms from G to H that follow D, I.

Proof Sketch: By induction over the recursion tree:
Base: the unbalanced / bijection case (use the corollary).
Induction step: use the previous proposition. □

Theorem

Algorithm 2 terminates.

Proof Sketch: In every recursive call D and I grow by one element.
If $|D| = |I| = |V(G)| = |V(H)|$, the algorithm is done. Thus, the recursion depth is at most $|V(G)|$ and the algorithm terminates. □

Correctness of Individualization Refinement

Theorem

Algorithm 2 is correct; it returns the number of isomorphisms from G to H that follow D, I.

Proof Sketch: By induction over the recursion tree:
Base: the unbalanced / bijection case (use the corollary).
Induction step: use the previous proposition. □

Theorem

Algorithm 2 terminates.

Proof Sketch: In every recursive call D and I grow by one element.
If $|D| = |I| = |V(G)| = |V(H)|$, the algorithm is done. Thus, the recursion depth is at most $|V(G)|$ and the algorithm terminates. □

Quiz: How many recursive calls on C?

Correctness of Individualization Refinement

Theorem

Algorithm 2 is correct; it returns the number of isomorphisms from G to H that follow D, I.

Proof Sketch: By induction over the recursion tree:
Base: the unbalanced / bijection case (use the corollary).
Induction step: use the previous proposition. □

Theorem

Algorithm 2 terminates.

Proof Sketch: In every recursive call D and I grow by one element.
If $|D| = |I| = |V(G)| = |V(H)|$, the algorithm is done. Thus, the recursion depth is at most $|V(G)|$ and the algorithm terminates. □

Quiz: How many recursive calls on C? It is $|C|/2$ many !

Summary of Individualization Refinement

- Algorithm 2 computes the number of isomorphisms between two graphs G and H. (When called with D and I both empty.)
- The number of recursive calls is at least the number of isomorphisms between G and H, which can be exponential. (worst case: $n!$)
- Implementation for GI (decision problem): terminate as soon as one isomorphism is found!
- By choosing $H := G$, we can solve the #Aut problem (counting the number of automorphisms of G).

Improvement 1: Branching Rules

Improvements are possible by choosing the color class C and vertex x for branching cleverly.

Implementation idea: Try out and compare different *branching rules*.
Find at least one rule that performs better than straightforward rules, and support this with computational experiments.

Improvement 2: Preprocessing

Examples of problematic structures:

Definition

Two vertices $u, v \in V(G)$ are twins *if $uv \in E(G)$ and $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. They are* false twins *if $N(u) = N(v)$.*

Improvement 2: Preprocessing

Examples of problematic structures:

Definition

Two vertices $u, v \in V(G)$ are twins *if $uv \in E(G)$ and $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. They are* false twins *if $N(u) = N(v)$.*

Observation

Let α be a coloring that assigns the same color to two (false) twins u, v. The coarsest stable coloring that refines α also assigns same color to u and v.

Improvement 2: Preprocessing

Examples of problematic structures:

Definition

Two vertices $u, v \in V(G)$ are twins if $uv \in E(G)$ and $N(u)\setminus\{v\} = N(v)\setminus\{u\}$. They are false twins if $N(u) = N(v)$.

Observation

Let α be a coloring that assigns the same color to two (false) twins u, v. The coarsest stable coloring that refines α also assigns same color to u and v.

If G has a set of k (false) twins, then this can add a factor $k!$ to the number of recursive calls made by Algorithm 2!

Implementation Idea: use *preprocessing algorithm* that detects (false) twins before applying individualization refinement.

Improvement 3: Trees or Forests

If G and H are trees or forests, our branching algorithm works very well for the GI problem, but not so well for the #Aut problem, when there are many automorphisms.

Improvement 3: Trees or Forests

If G and H are trees or forests, our branching algorithm works very well for the GI problem, but not so well for the #Aut problem, when there are many automorphisms.

Implementation Idea: It can be shown that branching algorithm solves the GI problem for trees in *polynomial time*. However there is also a polynomial time algorithm for the #Aut problem on trees. . . .

Project: This Week

- Implement in Python an individualization refinement algorithm, that can correctly solve the GI and the #Aut Problem.
- This requires the *color refinement* algorithm to work - if you have not completed this, you need to do that ASAP. If you need help, ask during the project session.
- On canvas, there are additional *instances* of the GI and #Aut problems for testing, which require individualization refinement.
- Use small instances (for verifying correctness), and large instances (for tuning the performance).
- Think about which additional tricks you want to implement (twins, trees/forests, etc.). Test how they influence performance.

Project: This Week

- Implement in Python an individualization refinement algorithm, that can correctly solve the GI and the #Aut Problem.
- This requires the *color refinement* algorithm to work - if you have not completed this, you need to do that ASAP. If you need help, ask during the project session.
- On canvas, there are additional *instances* of the GI and #Aut problems for testing, which require individualization refinement.
- Use small instances (for verifying correctness), and large instances (for tuning the performance).
- Think about which additional tricks you want to implement (twins, trees/forests, etc.). Test how they influence performance.

Milestone:

- End of this week, Python program that can correctly solve the GI problem and #Aut problem for small instances.

- Along the way: *improving the computational complexity* is useful (for getting *bonus points* at the end).