VeRoLog Seville 2019

grafo RESEARCH

Implementing efficient code without dying in the effort

Jesús Sánchez-Oro

Universidad Rey Juan Carlos

Outline

1. Motivation
2. Code organization
3. Data structures
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. Conclussions

Outline

1. **Motivation**
2. Code organization
3. Data structures
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. Conclussions

Why I started working in routing problems?

Motivation

- Two metrics are considered to evaluate the **quality** of an algorithm:
  - **Objective function** value
  - Computing **time**

How relevant is the programmer?

- We start from a **high quality algorithm**.
- Otherwise, the programmer has nothing to do.

How relevant is the programmer?

- If the algorithm is good, but the programmer is not…

How relevant is the programmer?

- If the algorithm is good, and the programmer is *reasonable* …

How relevant is the programmer?

- If both the algorithm and the programmer are excellent …

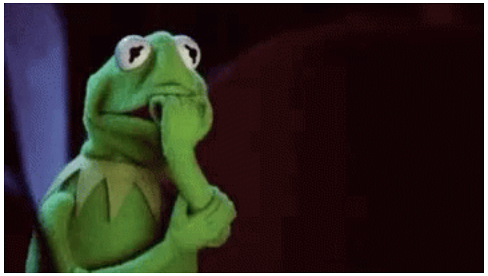
How relevant is the programmer?

- If the programmer is trying ***new things*** …

Select a programming language

MATLAB

C#



JavaScript

C/C++

Java

python™

Which is the best programming language?

- The best programming language **does not exist**
  - Otherwise, all of us will use the same language
- What are we looking for in a programming language?
  - Easy to learn
  - Performance
  - Debugging
  - External libraries

Why did I choose Java?

- It is **easy to learn** Java from scratch.
- JVM is responsible for **memory management**.
- Designed for **Object Oriented Programming**.
- A good code in Java **is not necessarily slower** than one in C/C++.

Why did I choose Java?

- **Developing time** in Java is rather smaller than in other languages.
- It has a lot of **external libraries** to help us with the code.

May 2019	May 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.005%	-0.38%
2	2		C	14.243%	+0.24%
3	3		C++	8.095%	+0.43%
4	4		Python	7.830%	+2.64%
5	6	^	Visual Basic .NET	5.193%	+1.07%
6	5	v	C#	3.984%	-0.42%
7	8	^	JavaScript	2.690%	-0.23%
8	9	^	SQL	2.555%	+0.57%
9	7	v	PHP	2.489%	-0.83%
10	13	^	Assembly language	1.816%	+0.82%

https://www.tiobe.com/tiobe-index/

Outline

1 Motivation

2 **Code organization**

3 Data structures

4 Test Problem: TSP

5 Code improvements

6 Parallelization

7 Conclussions

Code structure

- When we deal with a new problem, we first need to think about **code structure**.
- If the problem is similar to another one in which we have previously worked the **structure** will be **similar**.

Code structure

- Most of the **features** that we use for a certain problem are **repeated** for the rest of the problems.
- Is it really **necessary** to repeat the same again and again?

Code structure

- **First option**
  - **Copy and paste** the last project in which we have been working and modify the code

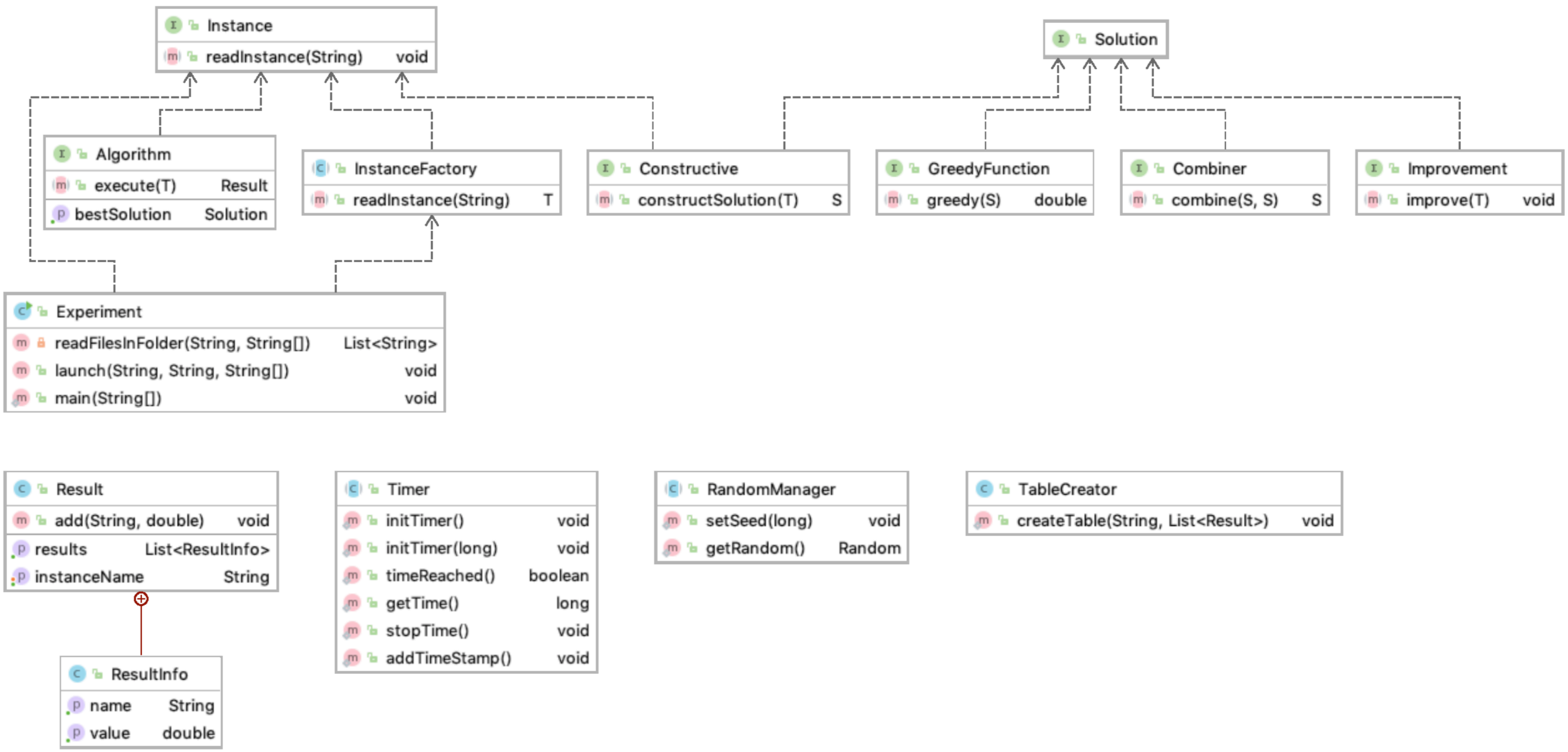
Code structure

- **Second option**
  - Take advantage of the **language features** in order to avoid repeating code.

Code structure

- **Proposal**: create a library which contains the basic functionality that will be required in any project.
  - Execute an algorithm over a set of instances in a folder.
  - Generate a table with the obtained results.
  - Control the computing time.
  - …

GrafoOptiLib



Outline

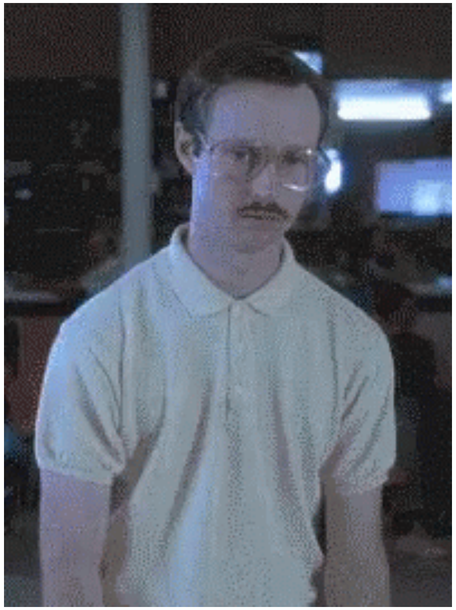
1. Motivation
2. Code organization
3. **Data structures**
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. Conclussions

Data Structures

- DS define the **data organization** of our problem.
- If we choose the correct DS, we will be able to add, modify or remove data **efficiently**.
- DS are usually one of the **key parts** of our code.

Data Structures

- Most languages offer their **own implementations** of several data structures, so we do not **usually** need to implement data structures.



Data Structures

- However, if we need more **complex or specific** structures, we will need to go deeper and implement them.

Data Structures

- We usually believe that the programming language is the **key for developing a fast algorithm**.
- Nevertheless, the actual key is the **complexity** of the data structures considered.

Data Structures

- If we perform **many operations** over the same data structure, we would like to make it as **efficient** as possible.
- We need to focus on **reducing the complexity** of the most **common** operations.
  - Which is the cost of inserting / searching /removing an element from a data structure?

Is it really so important?

- We will consider 1000 elements.
- Test:
  - **Search** for a random element

Why these results?

0	1	2	3	4	5
	5	4	3	1	2

Arrays

- Access to a given position in **constant** time
- Improvement in **memory storage**

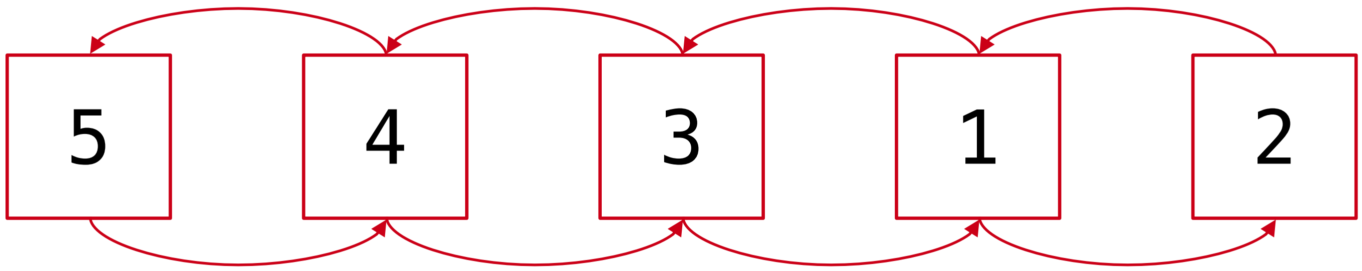
Why these results?

0	1	2	3	4	5
	5	4	3	1	2

ArrayList

- Similar to arrays in representation.
- **Overhead** to resize the data structure and offer more functionality (contains).

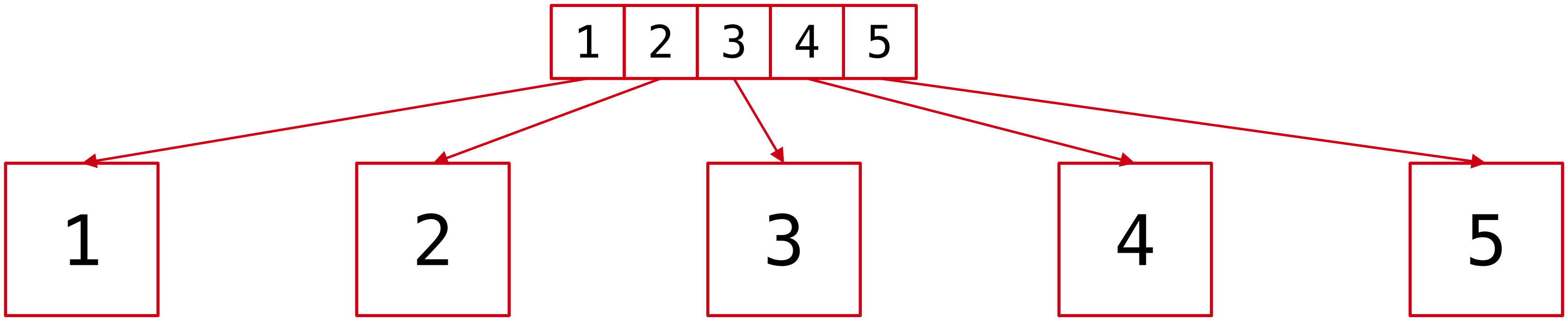
Why these results?



LinkedList

- Access to the **first and last** elements on the list.
- If we need to access k element, we need to move k positions starting at the first one.

Why these results?



HashSet

- Each element is identified by a **unique number**.
  - We need to define the **mapping** between element and its corresponding number (**hash code**).
- Check if an element is in the DS in **constant time**.

What data structure should I use?

- There is not a ***best*** data structure.
- It totally depends on the **most common operations** performed in your code.
- **THINK BEFORE CODING!!!**

Outline

1. Motivation
2. Code organization
3. Data structures
4. **Test Problem: TSP**
5. Code improvements
6. Parallelization
7. Conclussions

Test Problem: TSP

- **Input**: a set of n locations and the distance between each pair of locations.
- **Objective**: find the shortest possible route that visits every city exactly once and returns to the starting point.
  - Starting point is always the first node.

Test Problem: TSP

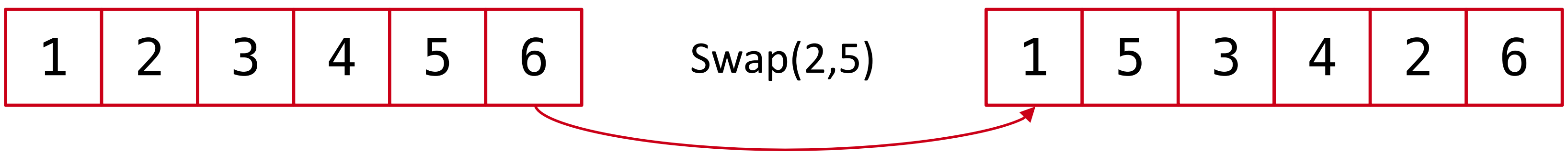
- What do we have to know about the **instance**?
  - Number of cities
  - Distances between cities

Test Problem: TSP

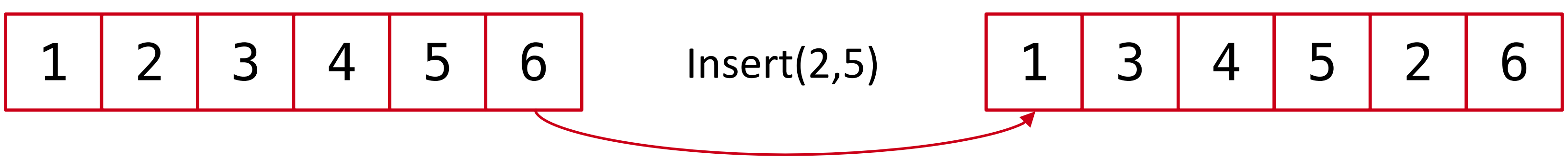
- What do we have to know about the **solution**?
  - Which instance are we solving?
  - Which is the selected route?
  - Which is the total distance for the route?

Test Problem: TSP

- Two basic **movements**:
  - **Swap** between two cities



  - **Insertion** of a city in a different position



Test Problem: TSP

- **G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
  - Construction phase
  - Improvement phase

Test Problem: TSP

$CL \leftarrow \{v \in V\}$

$v_f \leftarrow \text{Random}(CL)$

$S \leftarrow \{v_f\}$

$CL \leftarrow CL \setminus \{v_f\}$

while $CL \neq \emptyset$ **do**

    $g_{max} \leftarrow \min_{v \in CL} g(v)$

    $g_{max} \leftarrow \max_{v \in CL} g(v)$

    $\mu \leftarrow g_{min} + \alpha \cdot (g_{max} - g_{min})$

    $RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$

    $v_s \leftarrow \text{Random}(RCL)$

    $S \leftarrow S \cup \{v_s\}$

    $CL \leftarrow CL \setminus \{v_s\}$

endwhile

return S

Test Problem: TSP

$CL \leftarrow \{v \in V\}$

$v_f \leftarrow \text{Random}(CL)$

$S \leftarrow \{v_f\}$

$CL \leftarrow CL \setminus \{v_f\}$

while $CL \neq \emptyset$ **do**

  $g_{max} \leftarrow \min_{v \in CL} g(v)$

  $g_{max} \leftarrow \max_{v \in CL} g(v)$

  $\mu \leftarrow g_{min} + \alpha \cdot (g_{max} - g_{min})$

  $RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$

  $v_s \leftarrow \text{Random}(RCL)$

  $S \leftarrow S \cup \{v_s\}$

  $CL \leftarrow CL \setminus \{v_s\}$

endwhile

return S

The **Candidate List** contains all the nodes but the first one, which is randomly chosen.

Test Problem: TSP

$CL \leftarrow \{v \in V\}$
$v_f \leftarrow \text{Random}(CL)$
$S \leftarrow \{v_f\}$
$CL \leftarrow CL \setminus \{v_f\}$
while $CL \neq \emptyset$ **do**
    $g_{max} \leftarrow \min_{v \in CL} g(v)$
    $g_{max} \leftarrow \max_{v \in CL} g(v)$
    $\mu \leftarrow g_{min} + \alpha \cdot (g_{max} - g_{min})$
    $RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
    $v_s \leftarrow \text{Random}(RCL)$
    $S \leftarrow S \cup \{v_s\}$
    $CL \leftarrow CL \setminus \{v_s\}$
endwhile
return S

The **Restricted Candidate List** contains all the nodes whose objective function value is better than a certain threshold.

Test Problem: TSP

$CL \leftarrow \{v \in V\}$

$v_f \leftarrow \mathrm{Random}(CL)$

$S \leftarrow \{v_f\}$

$CL \leftarrow CL \setminus \{v_f\}$

while $CL \neq \emptyset$ **do**

$\quad g_{max} \leftarrow \min_{v \in CL} g(v)$

$\quad g_{max} \leftarrow \max_{v \in CL} g(v)$

$\quad \mu \leftarrow g_{min} + \alpha \cdot (g_{max} - g_{min})$

$\quad RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$

$\quad v_s \leftarrow \mathrm{Random}(RCL)$

$\quad S \leftarrow S \cup \{v_s\}$

$\quad CL \leftarrow CL \setminus \{v_s\}$

endwhile

return S

A random node from the RCL is selected as the next city of the route, updating the **CL**

Test Problem: TSP

Improvement phase

- We test two local search methods, one for each movement.
- First improvement approach.
- Random exploration of the neighborhood.

Test Problem: TSP

- Let's test a **direct implementation** without any improvement.
- We will try different data structures to represent a route.

Outline

1. Motivation
2. Code organization
3. Data structures
4. Test Problem: TSP

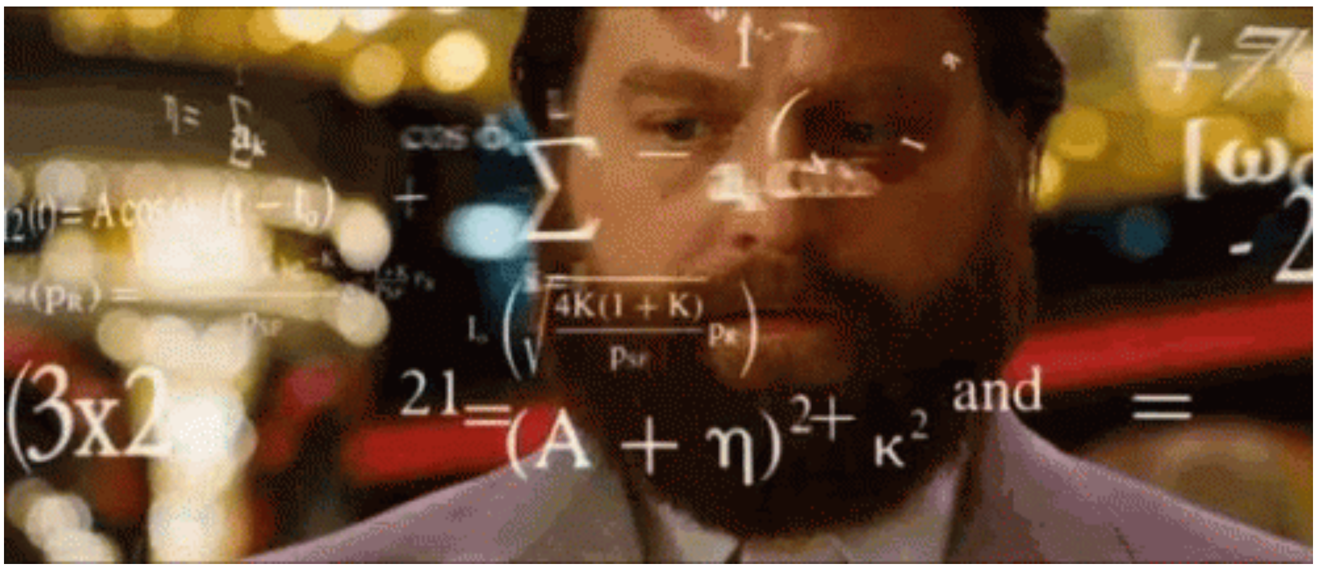
5. **Code improvements**
6. Parallelization
7. Conclussions

Code improvements

- This code is **too slow**!!
- Bad performance is usually related to **repeating computations unnecessarily**.
- It is very common in the **objective function evaluation**.

Code improvements

- Is it really necessary to evaluate the **complete objective function** after performing a **single movement**?

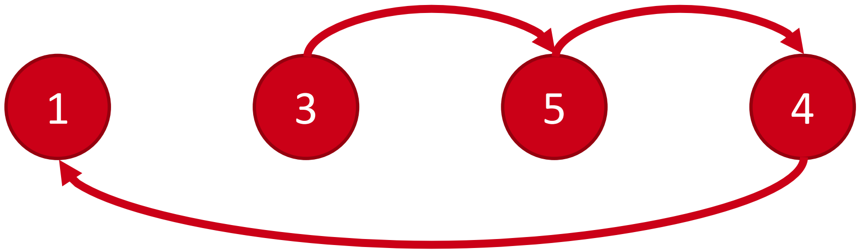


Code improvements

- We need to study which evaluations are **strictly necessary** to save computing time.
- For instance, in the TSP:
  - How can I update the total distance when **adding** a new city?
  - How does a **movement** affect the objective function value?

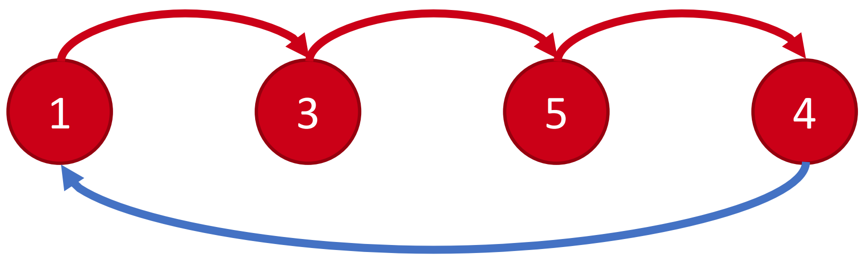
Code improvements

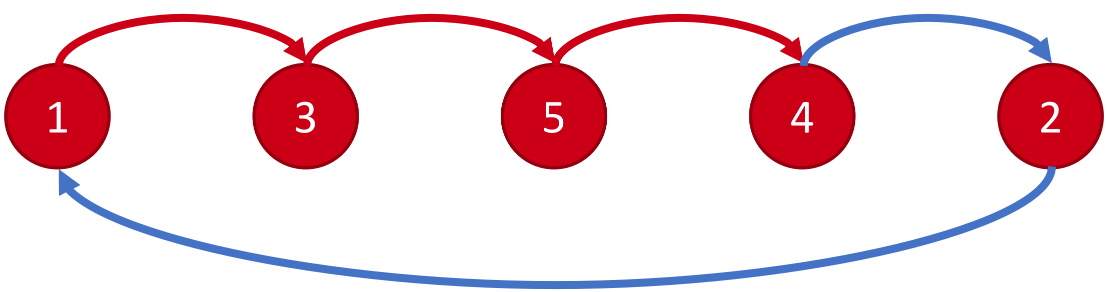
- When **adding** a new city:



Code improvements

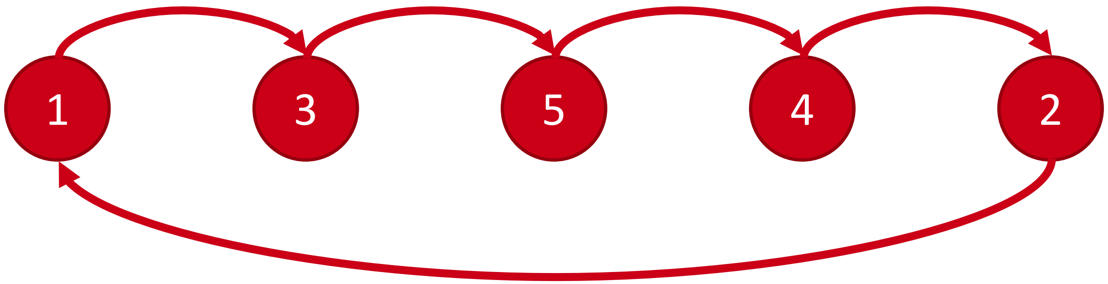
- When **adding** a new city:





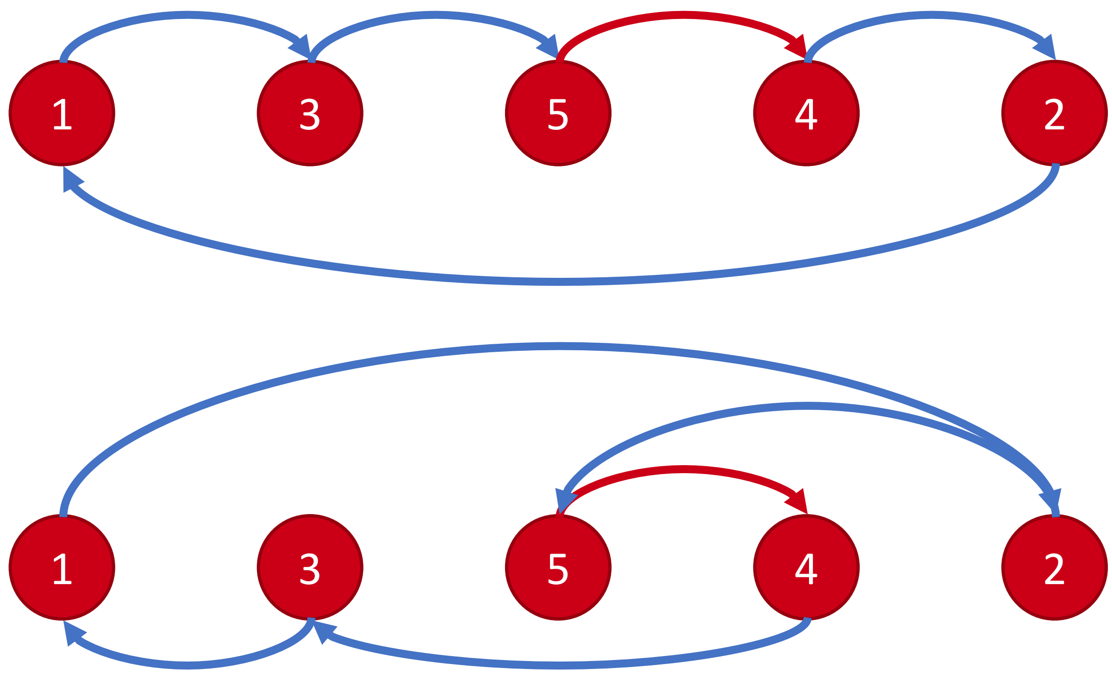
Code improvements

- When **swapping** two cities:
  - Swap(2,3)



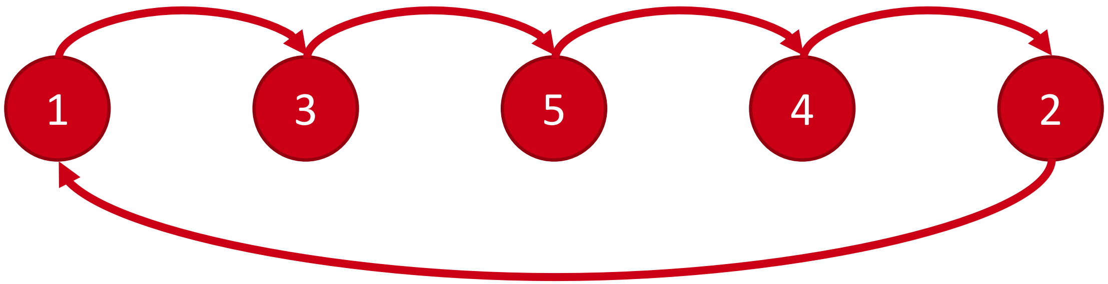
Code improvements

- When **swapping** two cities:
  - Swap(2,3)



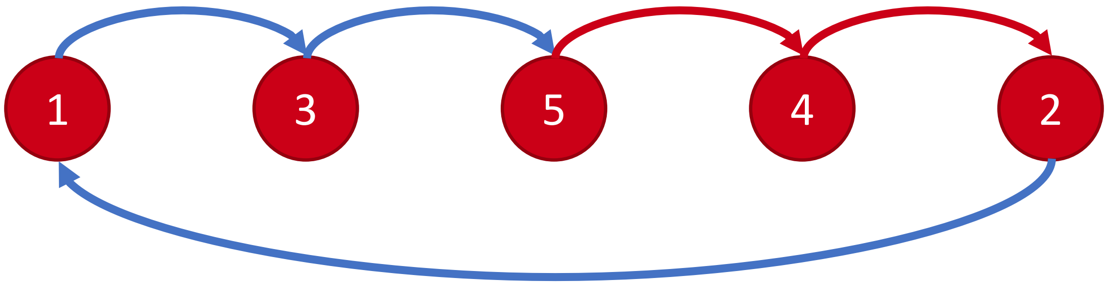
Code improvements

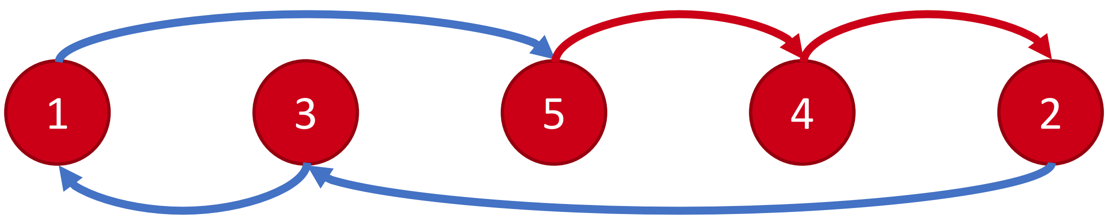
- When **inserting** a city in a different position:
  - Insert(3,2)



Code improvements

- When **inserting** a city in a different position:
  - Insert(3,2)





Code improvements

- We must analyze the **complexity** of the most common operations in the data structures.
- Complexity of adding / removing elements in:
  - ArrayList
  - LinkedList

Code improvements

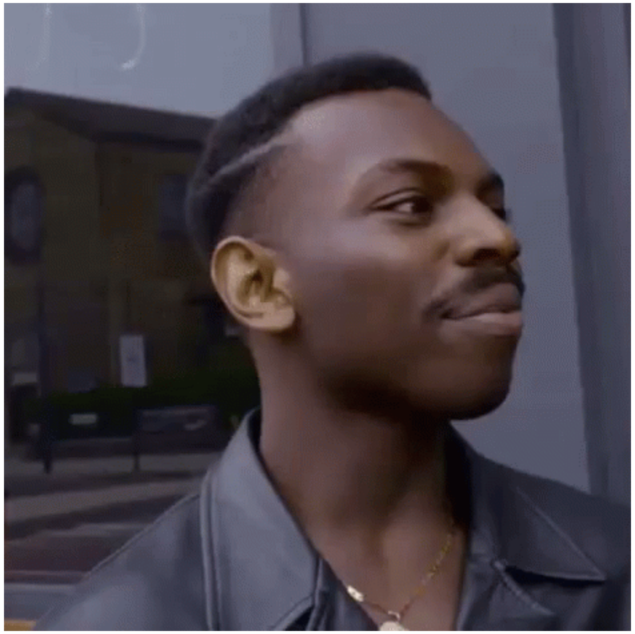
- LinkedList **should** be the best data structure for the problem.
- However, Java implementation of LinkedList offers poor performance.

Code improvements

- Is it enough for us?

Code improvements

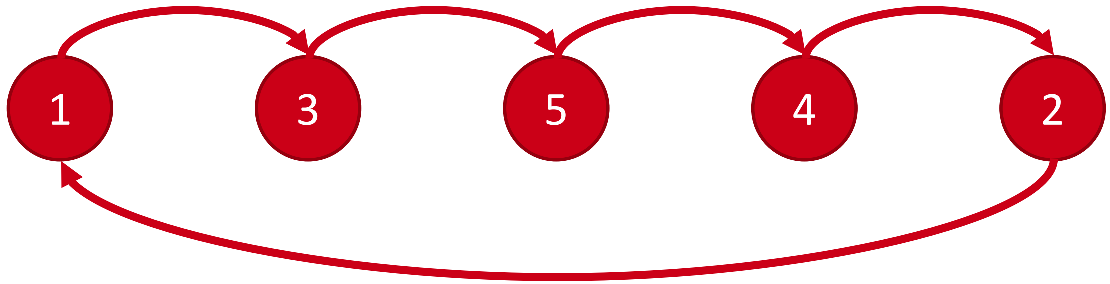
- What if **we implement** a new LinkedList which overcomes the disadvantages of the original one?



Code improvements

- MyLinkedList uses two integer arrays to represent a route:
  - prev[v] indicates the city located just before v
  - next[v] indicates the city located just after v
- All the operations are performed in **constant time**.

Code improvements



	0	1	2	3	4	5
prev	0	0	4	1	5	3
next	0	3	1	5	2	4

Outline

Parallelization

- Most of our computers have **more than one core**.
  - If not, please go now and renew your computer.
- Then, why are we still developing **sequential code**?

Parallelization

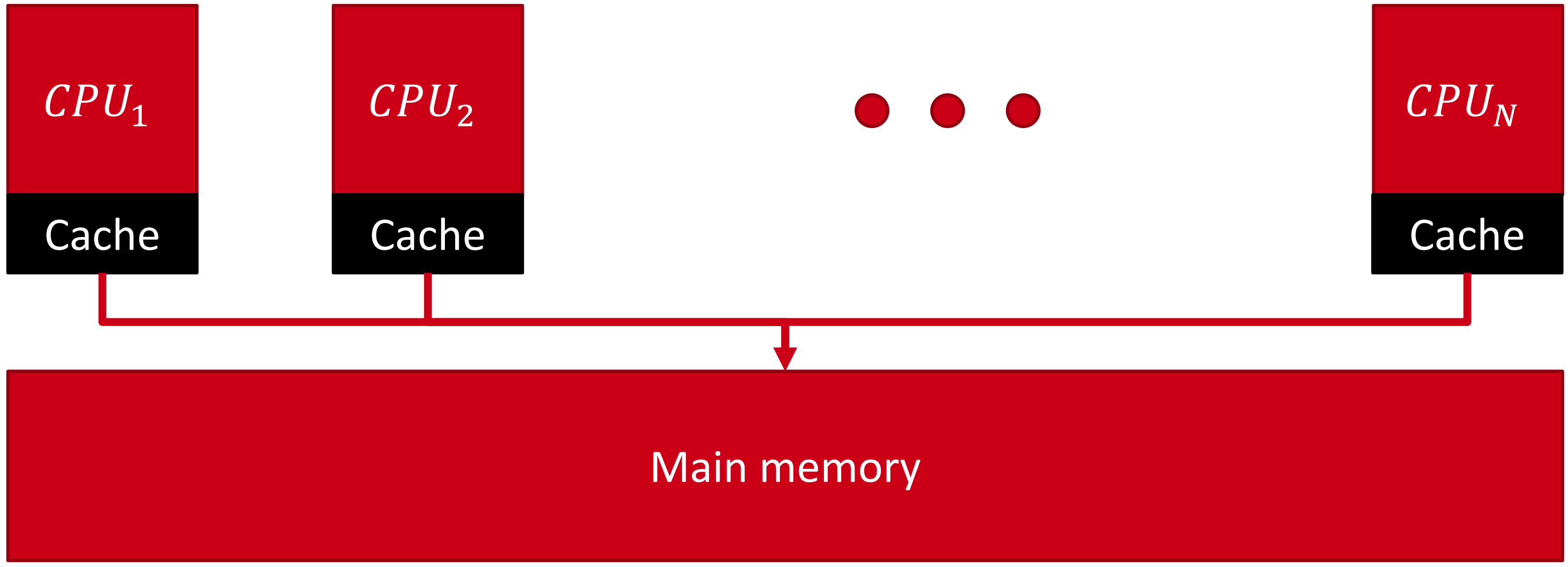
- In a sequential program we have a **single process** and a **single control flow**.
- In a **parallel program** we have two or more processes **cooperating** to finish a task.
  - We must ensure a correct **communication** and **synchronization** among processes.

Parallelization

- **Be careful!** Power is nothing without control.
- We should learn how to code parallel programs.
  - Otherwise, it could be **slower than the sequential version**

Memory model

- We use a **shared memory model**.
- A **single memory** is shared among all the processors



How can I parallelize code?

- Using a compiler that **automatically** convert sequential code in parallel code.
- Advantages:
  - We do nothing.
- Disadvantages:
  - Not available for all programming languages.
  - The parallelization achieved is not the best one.

How can I parallelize code?

- Using **operating system resources**: processes, threads, semaphores, files, ...
- Advantages:
  - Available in every programming language.
- Disadvantages:
  - Ridiculously hard

How can I parallelize code?

- Using **libraries** that simplify the parallelization, like OpenMP.
- Advantages:
  - We just need to slightly modify our sequential code.
- Disadvantages:
  - Not available for every language.

How can I parallelize code?

- Using a **programming language** prepared for parallelism.
- Advantages:
  - Every modern computer language is prepared for it.
- Disadvantages:
  - We need to deeply modify our code.

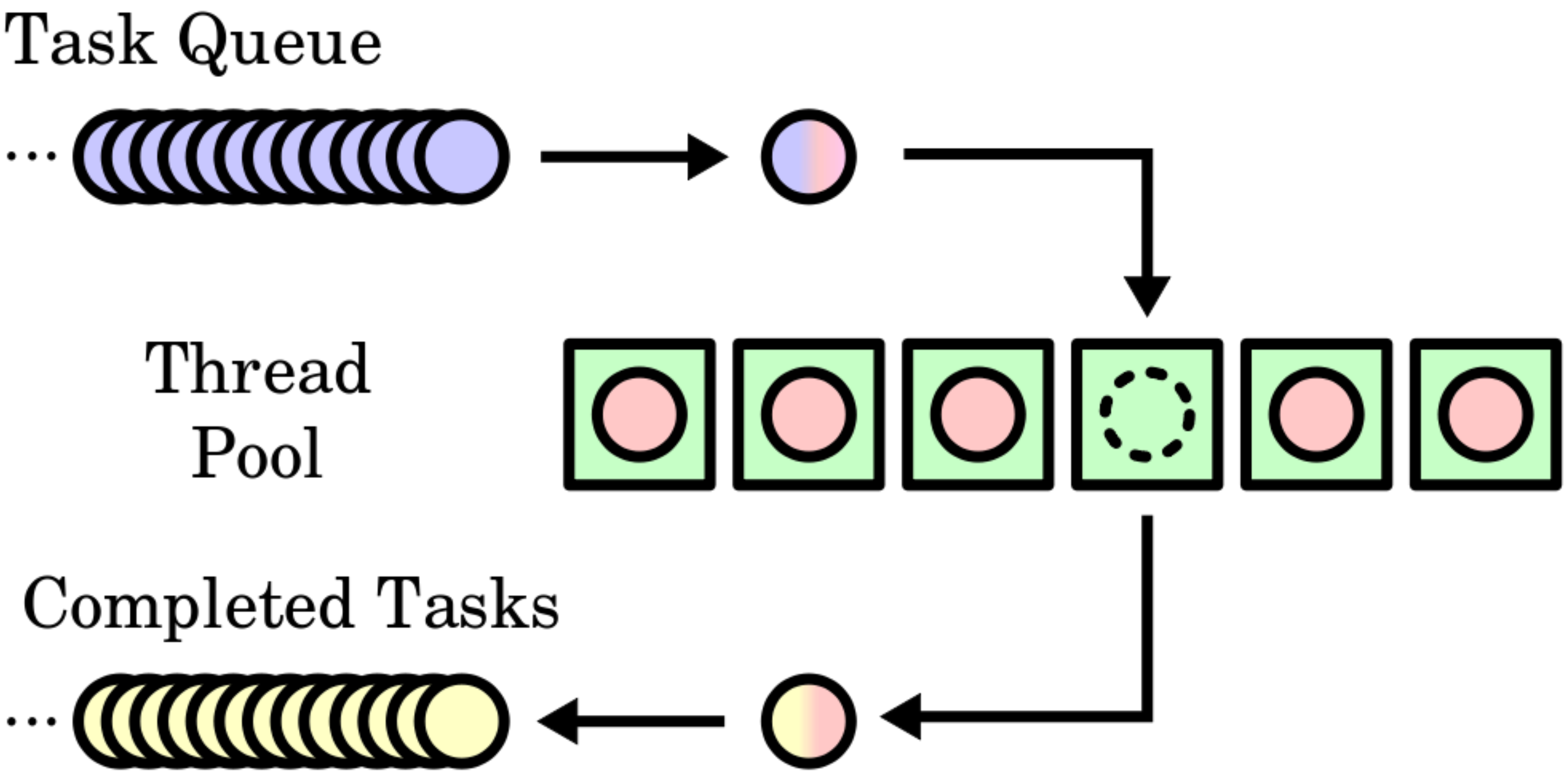
Parallelization in Java

- Java is prepared for developing parallel code **easily**.
- We can use the low level tools, but Java offers a set of **high level tools** to parallelize code ignoring details.

How can I parallelize a metaheuristic?

- Parallelize independent code fragments, **without algorithm redesign**.
  - Small scientific contribution.
  - Very easy.
- **Redesign the algorithm** to make the most of available hardware.
  - Relevant scientific contribution.
  - Harder.

Java Thread Pool



Want more parallelism?

- The second option implies **redesigning the algorithm** in a parallel way.
- Most of the metaheuristics already have a **parallel design**.
  - Alba, E. (2005). Parallel metaheuristics: a new class of algorithms (Vol. 47). John Wiley & Sons.
  - Crainic, T. G., & Toulouse, M. (2010). Parallel meta-heuristics. In Handbook of metaheuristics (pp. 497-541). Springer, Boston, MA.
  - Crainic, T. G. (2016). Parallel Meta-heuristic Search. Handbook of Heuristics, 1-39.

Parallelization objectives

- Parallelizing **does not necessarily** implies reducing computing time.
- It can be also used for exploring a **wider portion** of the **search space**.
  - We can guide the search in several directions simultaneously, instead of following a single direction.

Outline

1. Motivation
2. Code organization
3. Data structures
4. Test Problem: TSP
5. Code improvements
6. Parallelization
7. **Conclussions**

Which language should I use?

- Look for:
  - Smooth learning curve.
  - Efficiency.
  - External libraries.
  - Documentation, support forums, …
  - Parallelizing possibilities.
  - Is it used in the heuristics community?

How should I organize the code?

- We must ***waste time*** deciding the structure of our code.
- If we usually work on similar problems, we should think about **developing our own library** to avoid repeating common tasks.

Which is the key to efficiency?

- We should know the **data structures** that we use.
- **Incremental evaluation** of the objective function is one of the first optimizations to consider.
- Can I use an **alternative objective function**?
- Should I try a **parallel design**?

Thanks!!

Implementing efficient code without dying in the effort

Jesús Sánchez-Oro

Universidad Rey Juan Carlos