香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Misc. Problem Discussion

Ethen Yuen {ethening}
2026-05-02

Introduction

- After 10 weeks of training, learning various algorithms and data structures...

Attendance

2026-04-18 16:11:30.064
2026-04-18 14:31:29.033
2026-04-18 12:58:46.126
2026-04-18 10:31:21.630
2026-04-11 17:05:00.000
2026-04-11 12:26:02.012
2026-04-11 10:52:44.821
2026-03-21 17:15:00.000
2026-03-21 11:54:07.710
2026-03-21 10:20:02.726
2026-03-14 16:50:00.000
2026-03-14 16:50:00.000
2026-03-14 12:29:35.127
2026-03-14 11:10:48.097
2026-03-07 17:05:00.000
2026-02-28 16:50:00.000
2026-02-21 16:35:54.598
2026-02-21 14:47:23.913
2026-02-14 15:57:44.966
2026-02-14 14:27:33.421
2026-02-14 11:43:19.000

Introduction

- You may expect the results of minicomp to look like this:

M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Accepted	0.701
M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Accepted	0.563
M2634 - Paint and Guess *2026 Mini Competition III*	C++20	Accepted	0.017
M2631 - Message Recovery *2026 Mini Competition III*	C++20	Accepted	0.056
M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Accepted	0.230
M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Accepted	0.727
M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Accepted	0.681
M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Accepted	0.643

Introduction

- In reality...

M2633 - Marathon *2026 Mini Competition III*	C++20	Runtime Error	0 *Score*
M2633 - Marathon *2026 Mini Competition III*	C++20	Security Violation	0 *Score*
M2631 - Message Recovery *2026 Mini Competition III*	C++20	Wrong Answer	40 *Score*
M2634 - Paint and Guess *2026 Mini Competition III*	C++20	Partial Score (60.028)	60.028 *Score*
M2631 - Message Recovery *2026 Mini Competition III*	C++20	Wrong Answer	0 *Score*
M2631 - Message Recovery *2026 Mini Competition III*	C++20	Wrong Answer	0 *Score*
M2634 - Paint and Guess *2026 Mini Competition III*	C++20	Partial Score (15.000)	15 *Score*
M2631 - Message Recovery *2026 Mini Competition III*	C++20	Wrong Answer	24 *Score*
M2634 - Paint and Guess *2026 Mini Competition III*	C++20	Wrong Answer	0 *Score*
M2632 - Sightseeing Attractions *2026 Mini Competition III*	C++20	Runtime Error	0 *Score*

What went wrong?

Introduction

- Throughout our lectures, we have talked about topics that *we think you need to know*.
- Those are the **standard** stuff that every opponents of yours would learn, so you need to understand all of them.

- However, most of the problems you meet in contest are **not as straightforward**.

Introduction

- While previous lectures focus more on **algorithmic knowledge**, this lecture focuses on **problem solving skills**.

- To perform well in contests, **both are necessary**

Introduction

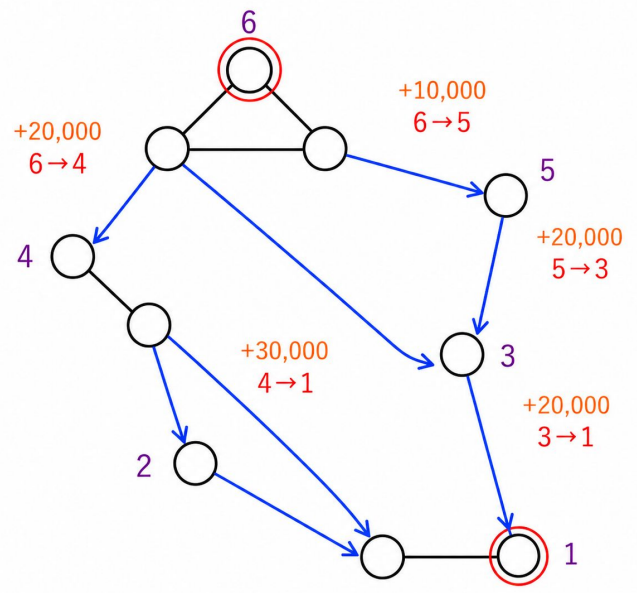
In 2023, we have gone through the most important technique of problem-solving: **"Reduce an unknown problem to a known problem"**

"If you can remodel a strange, unknown problem to a "standard" problem, the remaining step is simple: just solve it with your known tricks!"

Introduction

In 2023, we have gone through the most important technique of problem-solving: **“Reduce an unknown problem to a known problem”**

e.g. Dijkstra does not work on Graph with negative edge -> Reweight the graph to make the whole graph **non-negative**



Introduction

In 2023, we have gone through the most important technique of problem-solving: **“Reduce an unknown problem to a known problem”**

e.g. To **minimize the differences of two colours** in bipartite graph coloring ->
Knapsack on the delta of each connected component

Introduction

In 2024, we have gone through the practicing of **“pattern matching”** for competitive programming.

“How do you identify what transformations to use? Look for common patterns that appear more than one time!”

The Metaphor of Perfect Pitch: Some people are good at it naturally, however, the instinct can be trained.

Introduction

In 2024, we have gone through the practicing of **“pattern matching”** for competitive programming.

e.g. Maximizing / Minimizing a ratio (sum of Ai) / (sum of Bi)
-> Always Binary search on answer (sum of Ai) / (sum of Bi) >= M
-> (sum of Ai) >= M * (sum of Bi)
-> (sum of **(Ai - M * Bi)**) >= 0

Introduction

In 2024, we have gone through the practicing of **“pattern matching”** for competitive programming.

e.g. Finding the walk on a graph that yields **maximum median** edge weight

-> Always Binary search on answer

-> Edge >= M become 1, Edge < M become -1

-> Find a long enough walk with positive sum

Introduction

In 2024, we have gone through the practicing of **“pattern matching”** for competitive programming.

e.g. Latter operations overrides earlier operations (e.g. **M24BG** painting on a grid)

-> Going in order is difficult because operation **depends on the future**

-> Going in backward order is always easier because each operation decide the final state

Introduction

- What is the main philosophy of these lectures?
- **The tasks you see in contest more likely to consist unoriginal ideas instead of completely new ideas.**
- Learn the **idea** instead of the task!
- Train yourself to identify the **patterns**!
- How?
  - 1st step: **do a lot of tasks**
  - 2nd step: learn from your mistakes: **ok to fail the first time**, do not fail when you see it the second time

Outline of Today's Lecture

- Let's look at various tasks that you may or may not done before, and try to come up with solutions together.

M26CG Game of XOR

- https://judge.hkoi.org/task/M26CG

- There is an unknown array A of length N.
- Given Q range XOR result of the form (L[i], R[i], X[i]) where (^ is XOR)
- A[L[i]] ^ A[L[i] + 1] ^ ... ^ A[R[i]] = X[i]

- Recover any possible A or report impossible.

M26CG Game of XOR

- A[L[i]] ^ A[L[i] + 1] ^ ... ^ A[R[i]] = X[i]

- Seeing any **range relations** should always prompt you to think about turning it into **point relations** on partial sum (partial xor) array.
- P[L[i] - 1] ^ P[R[i]] = X[i]

- Why? There are so many variable within a range, but in the partial xor, it comes down to a relationship between two variables
  - The DFS to check whether all constraints can be satisfied is trivial

M26CG Game of XOR

- Any range relations should always prompt you to think about turning it into point relations on partial sum (partial xor) array.

- Can you think of any other task that has this pattern?

J243 Neat Corridor

- Given a 0-1 array A of length N
- We want to final state to be in the format of 000...000111...111
- (including all 0 and all 1)

- You can flip any consecutive range of length K in the array.
- e.g. N = 5, K = 3, A = 10100
- Flip position 0 ~ 2: **101**00 -> **010**00

- Minimize flips

J243 Neat Corridor

- Consider this task under the differences array D of the bit instead.
- You can flip any consecutive range of length K in the array A.
- **You can flip any D[i] and D[i + K] together in the array**

- We want to final state of A to be in the format of 000...000111...111
- **We want to final state of D to be in the format of 000...0001000...000**
- (or all 0)

- We can see that we can only act on positions in the same group mod K

J243 Neat Corridor

- **You can flip any D[i] and D[i + K] together in the array**
- **We want to final state of D to be in the format of 000...0001000...000**
- We can see that we can only act on positions in the same group **mod K**

- Some properties immediately appears under this new perspective
  - **At most one group mod K** can have odd number of 1
  - We can consider only that particular group to find putting the 1 in which position attain the minimum cost

Range Relations <-> Point Relations

- A good way to learn is to think about: what other scenarios can a pattern be applied?

- As we know we can flatten a tree onto a list by their dfs-order, such that each subtree is a subarray in the list.
  - Relations on **subtree sum** ->
    Relations of **range sum on the dfs order** of the tree ->
    Relations of **two points on the partial sum** of the dfs order
- You should not be fooled by the disguise when the underlying pattern can be something that you are familiar with.

Range Relations <-> Point Relations

- A good way to learn is to think about: what other scenarios can a pattern be applied?

- Let's also try to extend the scenarios in another way:
- The previous examples are about partial **XOR**.
  - There is an unknown array A of length N.
  - Given Q range **XOR** result of the form (L[i], R[i], X[i]) where (^ is **XOR**)
  - A[L[i]] ^ A[L[i] + 1] ^ ... ^ A[R[i]] = X[i]

Range Relations <-> Point Relations

- A good way to learn is to think about: what other scenarios can a pattern be applied?

- Let's also try to extend the scenarios in another way:
- We can change it to partial **SUM** and it still works
  - There is an unknown array A of length N.
  - Given Q range **SUM** result of the form (L[i], R[i], X[i]) where
  - A[L[i]] + A[L[i] + 1] + ... + A[R[i]] = X[i]
- It changes from **XOR** relation between P[] elements, to **difference** between P[] elements.

M26CG Game of XOR (Extended)

- How about this?
- There is an unknown array A of length N.
- Given Q range **alternating sum** result of the form (L[i], R[i], X[i]) where

  A[L[i]] **-** A[L[i] + 1] + A[L[i] + 2] - ... +/- A[R[i]] = X[i]

- Is it possible?

M26CG Game of XOR (Extended)

- A[L[i]] - A[L[i] + 1] + A[L[i] + 2] - ... +/- A[R[i]] = X[i]

- We can define an array B
  - If i is **odd**, B[i] = A[i]
  - If i is **even**, B[i] = -A[i]
- Then the above alternating sum becomes:
  - If L[i] is **odd**, B[L[i]] + B[L[i] + 1] + ... + B[R[i]] = **X[i]**
  - If L[i] is **even**, B[L[i]] + B[L[i] + 1] + ... + B[R[i]] = **-X[i]**
- We can first solve B then compute A
- Note: This pattern of **flipping sign based on parity** is common too (e.g. see M26CH)

M26CG Game of XOR (Extended)

- Basically this comes down to: what operators can be used during normal partial sum?
  - OK: e.g. sum, product, xor, alternating sum

- Answer: Any operations that has **inverse** would works fine
  - Even operations without commutivity would work: e.g. non-singular matrix multiplication
  - Note: Because you can always get A[L[i]] X A[L[i] + 1] X … X A[R[i]] by (P[L[i] - 1])^-1 X P[R[i]]

M26CG Game of XOR (Extended)

- Basically this comes down to: what operators can be used during normal partial sum?
  - OK: e.g. sum, product, xor, alternating sum

- Answer: Any operations that has **inverse** would works fine
  - Even operations without commutivity would work: e.g. non-singular matrix multiplication
  - Note: Because you can always get A[L[i]] X A[L[i] + 1] X … X A[R[i]] by (P[L[i] - 1])^-1 X P[R[i]]

- Wait? 0 **does not have inverse** in multiplication?

M26CG Game of XOR (Extended II)

- Wait? 0 **does not have inverse** in multiplication?
- There is an unknown array A of length N.
- Given Q range **MULTIPLICATION** result of the form (L[i], R[i], X[i]) where
- A[L[i]] * A[L[i] + 1] * ... * A[R[i]] = X[i]

- If A[i] always non-0, we can just maintain the partial-product
- What if A[i] can be 0?
  - All partial-product on that point onwards just become 0!
  - It is like a black hole sucking everything in!

M26CG Game of XOR (Extended II)

- Let's forget about the task for a bit and think about how we can maintain partial product so we can get range product of element in O(1).

M26CG Game of XOR (Extended II)

- Let's forget about the task for a bit and think about how we can maintain partial product so we can get range product of element in O(1).
- The only thing is blocking us is 0
  - We could maintain a **partial sum on number of 0 in a prefix**
  - P0[i] = P0[i - 1] + [A[i] == 0]
  - We can also maintain a **partial product on non-0 elements in a prefix**
  - If A[i] != 0: P1[i] = P1[i - 1] * A[i]
  - Else: P1[i] = P1[i - 1]
- Then, we can query the product by
  - 1. Check if there's any 0 in range, if yes, the answer is 0
  - 2. Otherwise, use the P1[] to get the range product

M26CG Game of XOR (Extended II)

- How this helps us on the task?
- This shows us that we should handle 0 separately in a kind of a two stage method. We **decouple 0 and non-0 cases to handle differently**.

- A[L[i]] * A[L[i] + 1] * ... * A[R[i]] = X[i]
- If X[i] is non-zero: Every element in the **range [L[i], R[i]] is non-zero**
  - We can then consider the relation between P[L[i] - 1] and P[R[i]] as previously
  - We should also mark the constraints that L[i] .. R[i] must all be **non-zero**
- If X[i] is zero: There should be **at least one zero** in the range

M26CG Game of XOR (Extended II)

- A[L[i]] * A[L[i] + 1] * ... * A[R[i]] = X[i]
- If X[i] is non-zero: Every element in the **range [L[i], R[i]] is non-zero**
  - We can then consider the relation between P[L[i] - 1] and P[R[i]] as previously
  - We should also mark the constraints that L[i] .. R[i] must all be **non-zero**
- If X[i] is zero: There should be **at least one zero** in the range

- We can first process all non-zero X[i] constraints first, marking range to be non-zero using differences array.
- Then, we can **safely marked all other elements as zero**!
- If a (X[i] is zero) range's all elements are marked as non-zero **-> Impossible**

M26CG Game of XOR (Extended II)

- Then, we can **safely marked all other elements as zero**!
- If a (X[i] is zero) range's all elements are marked as non-zero **-> Impossible**

- **Why** we can do this?
- These are the elements that are not cared about by any (X[i] is non-zero) range!
- We can fill them in as whatever, and it **would not have any side-effect**!
- To satisfy the (X[i] is zero) range as much as possible: we should just fill them all as zero

M26CG Game of XOR (Extended II)

- Then, we can **safely marked all other elements as zero**!
- If a (X[i] is zero) range's all elements are marked as non-zero **-> Impossible**

- Let's viewed it from another perspective: these zero zones actually split the array into multiple isolated subarrays, and the edges will only be connected within those subarray.
- Hence, we can still do our original solution after this isolation!
- This is the **"turning unknown problem to known problem"** skills!

M26CG Game of XOR (Extended II)

- Then, we can **safely marked all other elements as zero**!
- If a (X[i] is zero) range's all elements are marked as non-zero **-> Impossible**

- Here is **another extension** of this task:
- We about if we want to **minimize the number of 0 in A[i]?**

M26CG Game of XOR (Extended II)

- Then, we can **safely marked all other elements as zero**!
- If a (X[i] is zero) range's all elements are marked as non-zero **-> Impossible**

- Here is **another extension** of this task:
- We about if we want to **minimize the number of 0 in A[i]?** It turns into a classical greedy interval selection problem!

M26CG Game of XOR (Extended II)

- Basically this comes down to: what operators can be used during normal partial sum?
  - OK: e.g. sum, product, xor, alternating sum
- Answer: Any operations that has **inverse** would works fine

- What about operations that lacks inverse and would “destroy information” while doing partial sum?
  - Not OK: e.g. min, max, gcd
  - We cannot use partial sum on these: We turn to doing Seg Tree / Sparse Table in DS(III).
- But, can we still do this task with a different approach?

M26CG Game of XOR (Extended III)

- There is an unknown array A of length N.
- Given Q range **MINIMUM** result of the form (L[i], R[i], X[i]) where
- Min(A[L[i]], A[L[i] + 1], ..., A[R[i]]) = X[i]

M26CG Game of XOR (Extended III)

- There is an unknown array A of length N.
- Given Q range **MINIMUM** result of the form (L[i], R[i], X[i]) where
- Min(A[L[i]], A[L[i] + 1], ..., A[R[i]]) = X[i]

- There are actually two constraints from one (L, R, X) relation
- 1. **Lower bound**: All elements in [L, R] >= X
- 2. **Existence**: Exist one element in [L, R] == X

M26CG Game of XOR (Extended III)

- There are actually two constraints from one (L, R, X) relation
- 1. **Lower bound**: All elements in [L, R] >= X
- 2. **Existence**: Exist one element in [L, R] == X
- Suppose element A[i] is impacted by multiple lower bounds: L[k1], L[k2], ..., L[km].

  Then, **A[i] >= max(L[k1], L[k2], ..., L[km])**

- We should always greedily make A[i] == max(L[k1], L[k2], ..., L[km])
  - there no point making it even larger
  - to maximize the fullfillment of the “Existence” constraints

M26CG Game of XOR (Extended III)

- We should always greedily make A[i] == max(L[k1], L[k2], ..., L[km])

- Hence, we can do a sweepline over the array, maintaining **current active lower bound with a max heap**.
  - At L[i], we add in the bound of X[i]
  - At R[i] + 1, we remove the bound of X[i]
- Then, we have the “best construction” of A[] that we can achieve
- We only need to check whether this construction satisfies the all the “existence” constraints!
  - e.g. using Sparse Table to quickly check them

M26CG Game of XOR (Extended IV)

How about GCD?

- There is an unknown array A of length N.
- Given Q range **GCD** result of the form (L[i], R[i], X[i]) where
- GCD(A[L[i]], A[L[i] + 1], ..., A[R[i]]) = X[i]

M26CG Game of XOR (Extended IV)

How about GCD?

- There is an unknown array A of length N.
- Given Q range **GCD** result of the form (L[i], R[i], X[i]) where
- GCD(A[L[i]], A[L[i] + 1], ..., A[R[i]]) = X[i]

What actually is GCD? (and, why did I put GCD right after min?)

M26CG Game of XOR (Extended IV)

What actually is GCD?

GCD(a, b) = p1^(min(a1, b1)) * p2^(min(a2, b2)), ..., pk^(min(ak, bk))

M26CG Game of XOR (Extended IV)

What actually is GCD?

GCD(a, b) = p1^(min(a1, b1)) * p2^(min(a2, b2)), ..., pk^(min(ak, bk))

GCD is basically **taking min simultaneously in multiple dimension**!

- Each dimension represents a single prime factor
- Hence, we can maintain multiple max heap, one for each prime factor, and do something like the previous solution!

Mini-summary

- **Convert to relations in prefix array & DFS**:
  - XOR, Sum, Product, ...
  - These are all operations with Inverse Element
- **Greedy construction + Sparse table verification**:
  - Min, Max, GCD, Bitwise AND, Bitwise OR
  - There are all operations that
  - **1. Idempotence:** OP(X, X) = X. The operations won't change the element even if applied multiple times, e.g. Min(5, 5) = 5.
  - **2. Partial Ordering**: There is a clear bound when multiple operations are applied. A[i] >= 5 && A[i] >= 10 => A[i] >= 10

Mini-summary

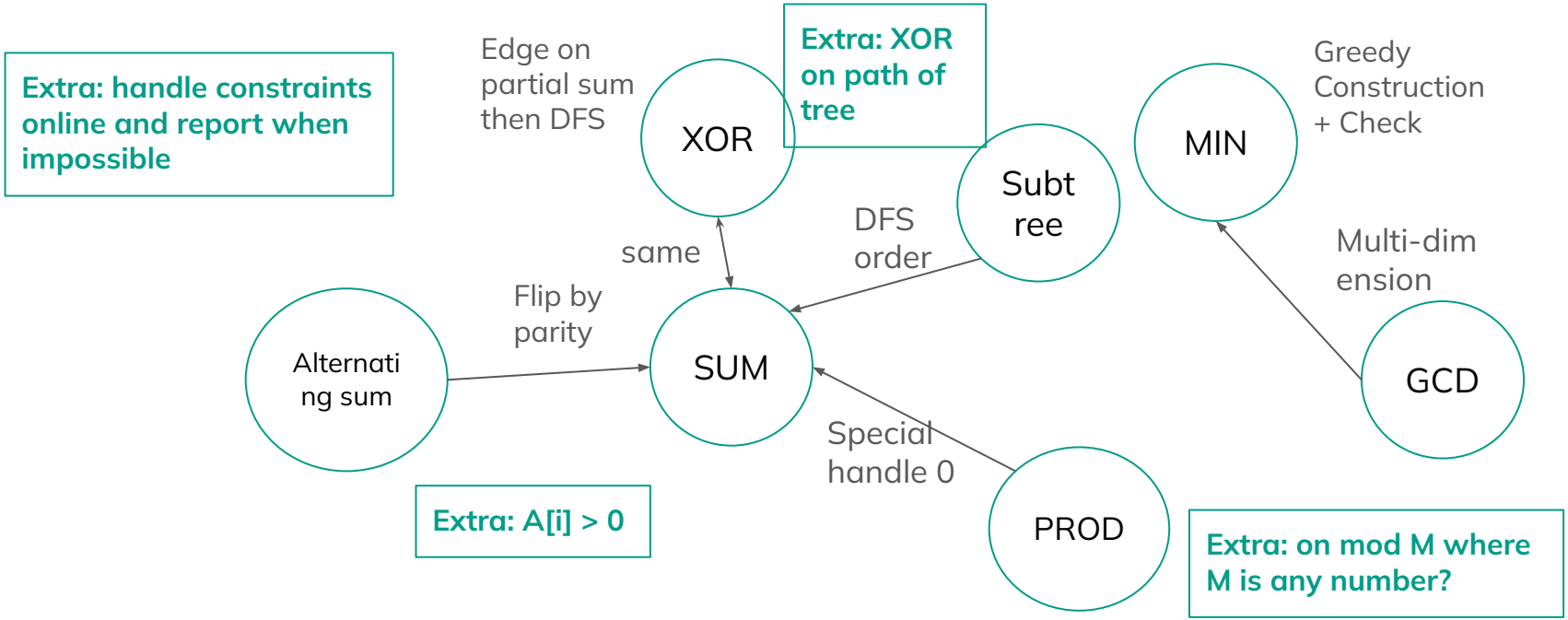
- **Convert to relations in prefix array & DFS**:
  - XOR, Sum, Product, ...
  - These are all operations with Inverse Element
- **Greedy construction + Sparse table verification**:
  - Min, Max, GCD, Bitwise AND, Bitwise OR
  - **1. Idempotence**
  - **2. Partial Ordering**
  - They are important for us to greedily construct each element individually, and no need care about other elements values

Mini-summary

- Let's ask more questions!
- **Greedy construction + Sparse table verification**:
  - Min, Max, GCD, Bitwise AND, Bitwise OR
  - Why we **cannot use the DFS approach** on these type of operations?
  - 1. For operations with inverse, range relations collapse to point relation on prefix array. We cannot determine range min in [L, R] by just looking at prefix range min of L-1 and R.
  - 2. For operations with inverse, range relations are actually equations. We can uniquely determine node v when node u is determined. In min, this transitivity is not available.

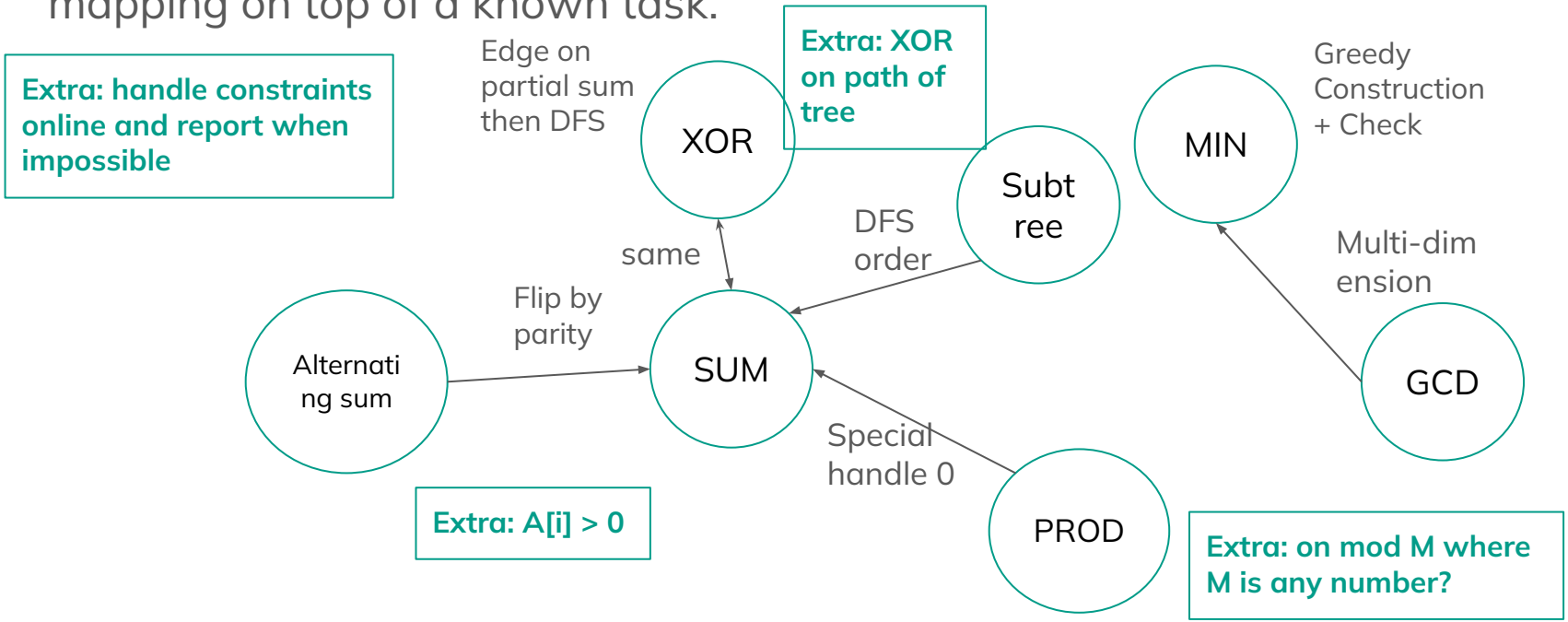
Summary

- Let's look at the different variations of this task



Summary

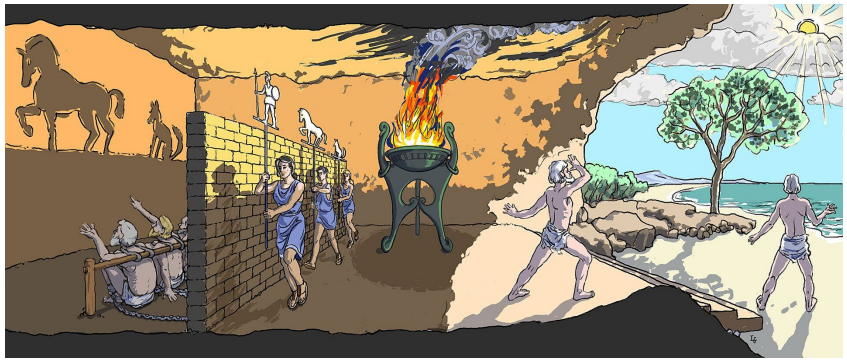
- Don't be fear by tasks with a lot of complications. It may just be layers of mapping on top of a known task.



Summary

- To learn more effectively, we reason about the inner patterns of the tasks, rather than the tasks themselves.
- Why?
  - These patterns are **more general** and could be applied to more tasks

Plato's Allegory of the cave:



To prepare for TFT...

Linux environment

- Use the command line to change directories and compile your program locally.
- Basic actions, such as opening, deleting, and moving files, can be performed using the graphical user interface.
- See: Introduction to Linux
- Install WSL if you use Windows / Use the terminal if you use Mac

Debug non-batch task

- Build a query function that output the queries and call the provided function.
- Modify the sample grader to display your queries (make sure you understand the sample grader before do so).
- E.g. M2643

```cpp
int paint(std::vector<bool> C);

int _paint(std::vector<bool> C)
{
    std::cerr << "Query:";
    for (bool i:C)
        std::cerr << ' ' << i;
    std::cerr << '\n';
    return paint(C);
}
```

Build a query function

```cpp
int paint(vector<bool> C) {

    std::cerr << "Query:";
    for (bool i:C)
        std::cerr << ' ' << i;
    std::cerr << '\n';

    if(++query_count > MAX_QUERY_COUNT) {
        wrong_answer("You may not call paint() more than 40000 times");
    }
```

Modify the sample grader

Debug non-batch task

- Visualiser may be provided in the bottom of statement.
- Run the visualiser according to the instructions.
- E.g. T254

DISPLAY TOOL

The attachment package `robo-visualizer.zip` contains a file named `display.html`. When opened in a browser, a graphical interface shows up and you can visualize Robo's execution under different initial configurations. The main features are as follows:

- You can observe the status of the realm, including the latest locations of Robo and the box. You can also observe which instruction Robo is currently executing.
- You can browse through the steps of Robo by clicking the left and right buttons. You can also jump to a specific step.
- You can play the simulation automatically by pressing the play button. The speed of execution can be adjusted using the slide bar in the bottom left corner.
- You can limit the number of steps simulated by imposing a step limit (**including labels**). Its default value is 10^5.

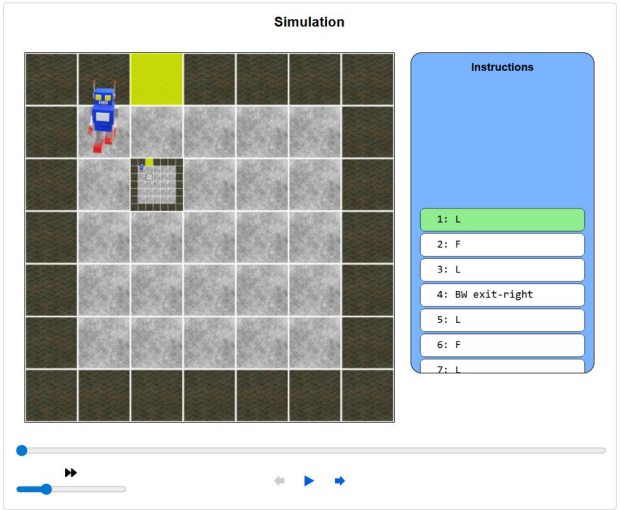
You should load in the sequence of instructions by selecting the file `program.robo`. The values of R and N are defined inside the file and are fixed. Alternatively, you may load the two files provided, `sample1.robo` and `sample2.robo`. They correspond to the two examples shown above.

Note that simulating with large grids and large step limits might cause your browser to hang. You are advised to simulate with the default step limit of 10^5 and on small grids.

Debug non-batch task

- Visualiser may be provided in the bottom of statement.
- Run the visualiser according to the instructions.
- E.g. T254

```
0 5
L
F
L
BW exit-right
L
F
L
F
F
L
F
L
F
F
J end
[label] exit-right
L
F
L
F
L
F
R
F
L
F
L
F
F
[label] end
```



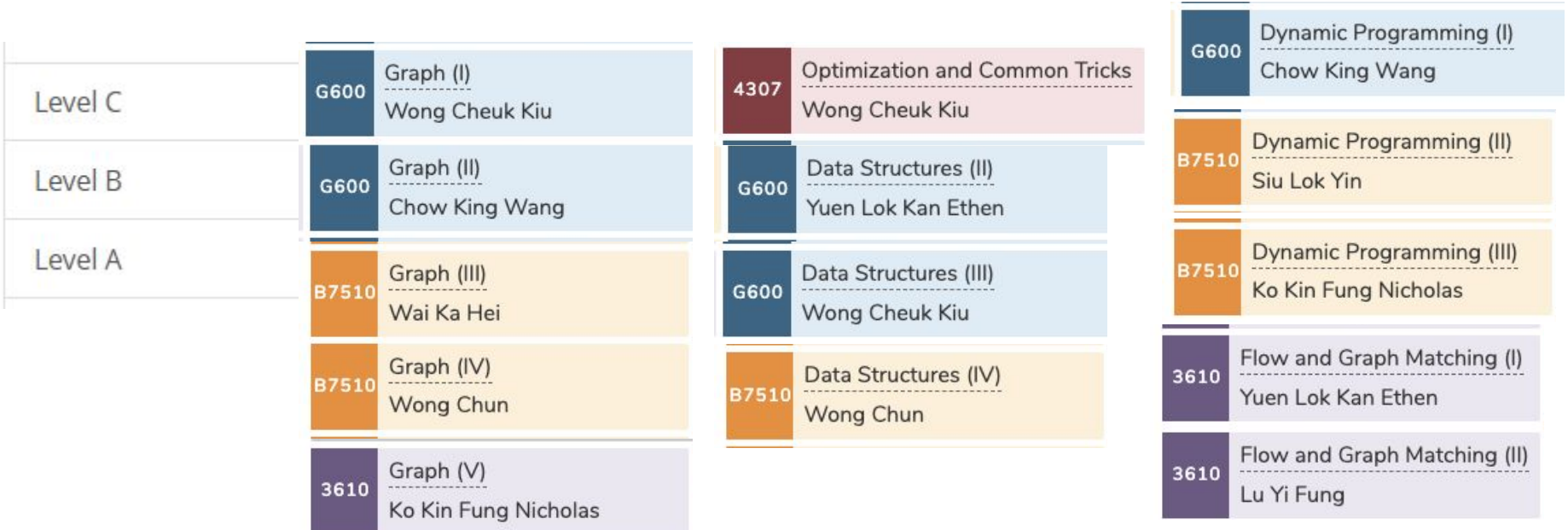
Practice

- Get familiar with standard algorithms.

Level C	37
Level B	20
Level A	30

Practice

- Get familiar with standard algorithms.
  Have you understand everything from the lectures?



Practice

- Get familiar with standard algorithms.
- Do past TFT/minicomps tasks.
- Play virtual TFT.

Virtual Contests

Name
[Virtual] 2025 Team Formation Test
[Virtual] 2024 Team Formation Test
[Virtual] 2023 Team Formation Test
[Virtual] 2022 Team Formation Test

Practice

- Get familiar with standard algorithms.
- Do past TFT/minicomps tasks.
- Play virtual TFT.
- Do tasks from international contests.
  - IOI
  - USACO
  - JOI (JOI Open / JOISC / JOI Final)

Mentality

- Believe in yourself
- (If this is your first time participating in TFT) 5 hour contest is a huge test on your pressure-handling ability
  - Prepare for a long fight
- First 2-days TFT this year
  - Don’t give up after Day 1
  - Don’t give up until last second
- Practice well & rest well