Ivan Kazmenko Contest 3 — Problem Analysis

Ivan Kazmenko

St. Petersburg State University

Monday, August 29, 2022

This is a RunTwice contest.
In every problem, the solution runs twice on each test.
This problem format can accomodate various problem topics.
Here are the topics (sub-genres) in this contest:

- encoding and decoding
- bijection
- lossy compression
- lossy channel
- secret sharing
- tell which run
- suspend and resume
- adaptive algorithms
- mathematical tricks
- prepare and play
- laborious problem

Statement

Problem A: Bracket-and-bar Sequences

- In this problem, we have to convert a regular bracket-and-bar sequence into a number and back.

Statement

Problem A: Bracket-and-bar Sequences

- In this problem, we have to convert a regular bracket-and-bar sequence into a number and back.
- The enumeration (lexicographical or otherwise) is up to us.

Statement

Problem A: Bracket-and-bar Sequences

- In this problem, we have to convert a regular bracket-and-bar sequence into a number and back.
- The enumeration (lexicographical or otherwise) is up to us.
- **Genre:** encoding and decoding.

Solution: Recursive Enumeration

- Consider the first opening bracket.

Solution: Recursive Enumeration

- Consider the first opening bracket.
- Its bar is at some position, u triples of characters to the right.

Solution: Recursive Enumeration

- Consider the first opening bracket.
- Its bar is at some position, u triples of characters to the right.
- Its closing bracket is at some further position, v triples of characters to the right.

Solution: Recursive Enumeration

- Consider the first opening bracket.
- Its bar is at some position, u triples of characters to the right.
- Its closing bracket is at some further position, v triples of characters to the right.
- Enumerate all pairs (u, v), consider three subproblems:

$$R_n = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1-u} R_u R_v R_w \qquad \text{where } w = n - 1 - u - v.$$

Solution: Recursive Number to Object

- Decoding works with a similar recursion:

```
function decode (n, number):
  for u = 0, 1, ..., n - 1:
    for v = 0, 1, ..., n - 1 - u:
      w = n - 1 - u - v
      cur = r[u] * r[v] * r[w]
      if number >= cur:
        number -= cur
      else:
        part3 = number % r[w]
        number /= r[w]
        part2 = number % r[v]
        number /= r[v]
        part1 = number
        return '(' + decode (u, part1) +
               '|' + decode (v, part2) +
               ')' + decode (w, part3)
```

Solution: Recursive Object to Number

- Encoding works with a similar recursion too:

```
function encode (n, s):
  p = 0 = position of first '('
  q = position of corresponding '|'
  r = position of corresponding ')'
  u0 = (q - p) / 3
  v0 = (r - q) / 3
  number = 0
  for u = 0, 1, ..., n - 1:
    for v = 0, 1, ..., n - 1 - u:
      w = n - 1 - u - v
      if u == u0 and v == v0:
        cur = encode (u, s[1..q))
        cur *= r[v]
        cur += encode (v, s[q + 1..r))
        cur *= r[w]
        cur += encode (w, s[r + 1..end))
        return number + cur
      else:
        number += r[u] * r[v] * r[w]
```

Statement

Problem B: Even and Odd Combinations

- In this problem, we have to establish a bijection between even-sized and odd-sized subsets of $\{1, 2, \ldots, n\}$.

Statement

Problem B: Even and Odd Combinations

- In this problem, we have to establish a bijection between even-sized and odd-sized subsets of $\{1, 2, \ldots, n\}$.
- In each run, given elements of one set, print the corresponding elements of the other set.

Statement

Problem B: Even and Odd Combinations

- In this problem, we have to establish a bijection between even-sized and odd-sized subsets of $\{1, 2, \ldots, n\}$.
- In each run, given elements of one set, print the corresponding elements of the other set.
- Runs are not distinguished in the input.

Statement

Problem B: Even and Odd Combinations

- In this problem, we have to establish a bijection between even-sized and odd-sized subsets of $\{1, 2, \ldots, n\}$.
- In each run, given elements of one set, print the corresponding elements of the other set.
- Runs are not distinguished in the input.
- **Genre:** bijection.

Solution 1: Toggle the 1

- If 1 is present, remove it.

Solution 1: Toggle the 1

- If 1 is present, remove it.
- If 1 is missing, insert it.

Solution 1: Toggle the 1

- If 1 is present, remove it.
- If 1 is missing, insert it.
- Either way, the parity of the set size has changed.

Solution 1: Toggle the 1

- If 1 is present, remove it.
- If 1 is missing, insert it.
- Either way, the parity of the set size has changed.
- Example for $n = 3$:

$$\begin{aligned}
\varnothing &\longleftrightarrow \{1\} \\
\{2\} &\longleftrightarrow \{1, 2\} \\
\{3\} &\longleftrightarrow \{1, 3\} \\
\{2, 3\} &\longleftrightarrow \{1, 2, 3\}
\end{aligned}$$

Solution 2: Subsets and Numbers

- Convert set to its number (among all odd-sized or all even-sized).

Solution 2: Subsets and Numbers

- Convert set to its number (among all odd-sized or all even-sized).
- Convert number to the respective element of the other class.

Solution 2: Subsets and Numbers

- Convert set to its number (among all odd-sized or all even-sized).
- Convert number to the respective element of the other class.
- With lexicographical order on binary representations of the sets, this solution is exactly the same as the previous one!

Statement

Problem C: Find the Parts

- In this problem, we have to compress the given large random matrix almost 10 times in size.

Statement

Problem C: Find the Parts

- In this problem, we have to compress the given large random matrix almost 10 times in size.
- And then, given its parts of size at least 10×10, locate them in the original matrix.

Statement

Problem C: Find the Parts

- In this problem, we have to compress the given large random matrix almost 10 times in size.
- And then, given its parts of size at least 10×10, locate them in the original matrix.
- **Genre:** lossy compression.

Solution: Each Tenth Line

- During the first run, store the dimensions and then only each 10-th line.

Solution: Each Tenth Line

- During the first run, store the dimensions and then only each 10-th line.
- Each 10×10 part has at least 10 consecutive values from one of the stored lines.

Solution: Each Tenth Line

- During the first run, store the dimensions and then only each 10-th line.
- Each 10×10 part has at least 10 consecutive values from one of the stored lines.
- Do some preprocessing to search efficiently.

Solution: Each Tenth Line

- During the first run, store the dimensions and then only each 10-th line.
- Each 10×10 part has at least 10 consecutive values from one of the stored lines.
- Do some preprocessing to search efficiently.
- Example: for each value `00`–`FF`, maintain a list of squares with that value. This way, we try 256x less starting squares for comparison.

Statement

Problem D: Noise Halving

- In this problem, we are given a short word and a long random integer sequence.

Statement

Problem D: Noise Halving

- In this problem, we are given a short word and a long random integer sequence.
- Transmit any subsequence.

Statement

Problem D: Noise Halving

- In this problem, we are given a short word and a long random integer sequence.
- Transmit any subsequence.
- During transmission, each element disappears with probability $1/2$.

Statement

Problem D: Noise Halving

- In this problem, we are given a short word and a long random integer sequence.
- Transmit any subsequence.
- During transmission, each element disappears with probability $1/2$.
- Restore the word at the other end of transmission.

Statement

Problem D: Noise Halving

- In this problem, we are given a short word and a long random integer sequence.
- Transmit any subsequence.
- During transmission, each element disappears with probability $1/2$.
- Restore the word at the other end of transmission.
- **Genre:** lossy channel.

Solution: Modulo and Repeat Symbol

- Let us encode each letter with a set of possible integers.

Solution: Modulo and Repeat Symbol

- Let us encode each letter with a set of possible integers.
- Example: $x \to x \bmod 26$ should be the letter number.

Solution: Modulo and Repeat Symbol

- Let us encode each letter with a set of possible integers.
- Example: $x \to x \bmod 26$ should be the letter number.
- For each letter, leave enough consecutive numbers corresponding to that letter ($10\,000/26/15 \approx 25.64$).

Solution: Modulo and Repeat Symbol

- Let us encode each letter with a set of possible integers.
- Example: $x \to x \bmod 26$ should be the letter number.
- For each letter, leave enough consecutive numbers corresponding to that letter ($10\,000/26/15 \approx 25.64$).
- Problem: double letters! How to distinguish "`ok`" and "`ook`"?

Solution: Modulo and Repeat Symbol

- Let us encode each letter with a set of possible integers.
- Example: $x \to x \bmod 26$ should be the letter number.
- For each letter, leave enough consecutive numbers corresponding to that letter ($10\,000/26/15 \approx 25.64$).
- Problem: double letters! How to distinguish "`ok`" and "`ook`"?
- Introduce the 27th letter "#", meaning "repeat the previous letter" ($10\,000/27/15 \approx 24.69$).

Solution: Modulo and Repeat Symbol

- Let us encode each letter with a set of possible integers.
- Example: $x \to x \bmod 26$ should be the letter number.
- For each letter, leave enough consecutive numbers corresponding to that letter ($10\,000/26/15 \approx 25.64$).
- Problem: double letters! How to distinguish "`ok`" and "`ook`"?
- Introduce the 27th letter "#", meaning "repeat the previous letter" ($10\,000/27/15 \approx 24.69$).
- This way, a hypothetical word "`aaabbbba`" would be encoded as "`a#ab#b#a`".

Statement

Problem E: Four Plus Four

- In this problem, we are given a dictionary, and we have to encode each 8-letter word (secret) with three 4-letter words (keys).

Statement

Problem E: Four Plus Four

- In this problem, we are given a dictionary, and we have to encode each 8-letter word (secret) with three 4-letter words (keys).
- And then, given *any two* of the three keys, we have to be able to restore the original secret.

Statement

Problem E: Four Plus Four

- In this problem, we are given a dictionary, and we have to encode each 8-letter word (secret) with three 4-letter words (keys).
- And then, given *any two* of the three keys, we have to be able to restore the original secret.
- **Genre:** secret sharing.

Statement

Problem E: Four Plus Four

- In this problem, we are given a dictionary, and we have to encode each 8-letter word (secret) with three 4-letter words (keys).
- And then, given *any two* of the three keys, we have to be able to restore the original secret.
- **Genre:** secret sharing.
- The dictionary published with the sample makes it an open-tests problem.

Observations

- A secret can have one, two, or three different keys.

Observations

- A secret can have one, two, or three different keys.
- Two different secrets can not have a common *pair* of keys.

Observations

- A secret can have one, two, or three different keys.
- Two different secrets can not have a common *pair* of keys.
- The hardest secrets are the secrets with least keys.

Technicalities

- There are 28 558 secrets and 3919 keys. The 112 million checks might be too slow.

Technicalities

- There are 28 558 secrets and 3919 keys. The 112 million checks might be too slow.
- How to efficiently find all keys for a given secret?

Technicalities

- There are 28 558 secrets and 3919 keys. The 112 million checks might be too slow.
- How to efficiently find all keys for a given secret?
- Consider *sets* of letters. Map each set to the list of corresponding words.

Technicalities

- There are 28 558 secrets and 3919 keys. The 112 million checks might be too slow.
- How to efficiently find all keys for a given secret?
- Consider *sets* of letters. Map each set to the list of corresponding words.
- For a secret, consider keys matching *subsets* of its set, then do a check for each.

Technicalities

- There are 28 558 secrets and 3919 keys. The 112 million checks might be too slow.
- How to efficiently find all keys for a given secret?
- Consider *sets* of letters. Map each set to the list of corresponding words.
- For a secret, consider keys matching *subsets* of its set, then do a check for each.
- May be done as precalculation and then encoded in the submission.

Solution: Greedy

- Sort secrets by the number of available keys.

Solution: Greedy

- Sort secrets by the number of available keys.
- Sort keys by the number of corresponding secrets.

Solution: Greedy

- Sort secrets by the number of available keys.
- Sort keys by the number of corresponding secrets.
- Start matching the secrets to keys in these orders, pick the first available triple, then forbid all three pairs from that triple. This almost works.

Solution: Greedy

- Sort secrets by the number of available keys.
- Sort keys by the number of corresponding secrets.
- Start matching the secrets to keys in these orders, pick the first available triple, then forbid all three pairs from that triple. This almost works.
- To make it fully work, for example, consider only patterns `S`$\rightarrow$`KKK` (one key repeated three times) and `S`$\rightarrow$`KLM` (three different keys). In other words, discard pattern `S`$\rightarrow$`KKL` (one of the keys repeated twice).

Solution: Greedy

- Sort secrets by the number of available keys.
- Sort keys by the number of corresponding secrets.
- Start matching the secrets to keys in these orders, pick the first available triple, then forbid all three pairs from that triple. This almost works.
- To make it fully work, for example, consider only patterns `S`$\to$`KKK` (one key repeated three times) and `S`$\to$`KLM` (three different keys). In other words, discard pattern `S`$\to$`KKL` (one of the keys repeated twice).
- Or perhaps do some backtracking.

Statement

Problem F: Graph Mark

- In this problem, we are given a random graph, and we have to mark it.

Statement

Problem F: Graph Mark

- In this problem, we are given a random graph, and we have to mark it.
- Next time we see the graph, vertices and edges shuffled, we have to detect the mark.

Statement

Problem F: Graph Mark

- In this problem, we are given a random graph, and we have to mark it.
- Next time we see the graph, vertices and edges shuffled, we have to detect the mark.
- **Genre:** tell which run.

Observations

- The solution will be probabilistic.

Observations

- The solution will be probabilistic.
- The mark we choose should have little probability in random graph.

Observations

- The solution will be probabilistic.
- The mark we choose should have little probability in random graph.
- On the other hand, it should be very probable that it is doable in 5 switchings of edges.

Observations

- The solution will be probabilistic.
- The mark we choose should have little probability in random graph.
- On the other hand, it should be very probable that it is doable in 5 switchings of edges.
- Each edge is present with probability $1/250$ to $1/100$.

Solution: K5

- How probable is a clique K5 in a random graph?
  $(1/100)^{10} \cdot \mathrm{choose}(1000, 5) < 10^{-5}/120$.

Solution: K5

- How probable is a clique K5 in a random graph? $(1/100)^{10} \cdot \mathrm{choose}(1000, 5) < 10^{-5}/120$.
- How probable are 5 vertices already having at least 5 edges between them? The expectation is $(1/100)^5 \cdot (99/100)^5 \cdot \mathrm{choose}(1000, 5)$, which is on the order of $10^{+5}/120$.

Solution: K5

- How probable is a clique K5 in a random graph? $(1/100)^{10} \cdot \mathrm{choose}(1000, 5) < 10^{-5}/120$.
- How probable are 5 vertices already having at least 5 edges between them? The expectation is $(1/100)^5 \cdot (99/100)^5 \cdot \mathrm{choose}(1000, 5)$, which is on the order of $10^{+5}/120$.
- So, find any 5 vertices with at least 5 edges between them (may even be brute forced), add the remaining edges.

Solution: K5

- How probable is a clique K5 in a random graph?
$(1/100)^{10} \cdot \mathrm{choose}(1000, 5) < 10^{-5}/120$.
- How probable are 5 vertices already having at least 5 edges between them? The expectation is $(1/100)^5 \cdot (99/100)^5 \cdot \mathrm{choose}(1000, 5)$, which is on the order of $10^{+5}/120$.
- So, find any 5 vertices with at least 5 edges between them (may even be brute forced), add the remaining edges.
- During the check, find 5 vertices with all 10 edges between them (again, may be brute forced).

Other Solutions

- Draw a pentagon or a hexagon without inner edges? Or perhaps some other shape.

Other Solutions

- Draw a pentagon or a hexagon without inner edges? Or perhaps some other shape.
- Consider vertex of maximum degree, add 5 to that degree.

Other Solutions

- Draw a pentagon or a hexagon without inner edges? Or perhaps some other shape.
- Consider vertex of maximum degree, add 5 to that degree.
- ...Use the imagination!

Statement

Problem G: Transfer of Duty

- In this problem, we operate one million switches as requested, and have to maintain the following information:
  - nothing is on, or
  - exactly one switch is on (tell which one then), or
  - more than one switch is on.

Statement

Problem G: Transfer of Duty

- In this problem, we operate one million switches as requested, and have to maintain the following information:
  - nothing is on, or
  - exactly one switch is on (tell which one then), or
  - more than one switch is on.
- Once, at some predetermined moment, the work pauses. We have a little piece of memory to store the current state. After resuming, we only remember what's in the memory.

Statement

Problem G: Transfer of Duty

- In this problem, we operate one million switches as requested, and have to maintain the following information:
  - nothing is on, or
  - exactly one switch is on (tell which one then), or
  - more than one switch is on.
- Once, at some predetermined moment, the work pauses. We have a little piece of memory to store the current state. After resuming, we only remember what's in the memory.
- **Genre:** suspend and resume.

Solution: Hashing

- Use set hashing.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.
- When a switch is toggled, xor the value C with the hash of that switch.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.
- When a switch is toggled, xor the value C with the hash of that switch.
- If $C = 0$, nothing is on.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.
- When a switch is toggled, xor the value C with the hash of that switch.
- If $C = 0$, nothing is on.
- If C is one of the individual hashes (stored in a map), then that individual switch is on.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.
- When a switch is toggled, xor the value C with the hash of that switch.
- If $C = 0$, nothing is on.
- If C is one of the individual hashes (stored in a map), then that individual switch is on.
- Otherwise, more than one switch is on.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.
- When a switch is toggled, xor the value C with the hash of that switch.
- If $C = 0$, nothing is on.
- If C is one of the individual hashes (stored in a map), then that individual switch is on.
- Otherwise, more than one switch is on.
- Memory used during suspend: $O(1)$.

Solution: Hashing

- Use set hashing.
- At start, assign a predetermined pseudorandom number (hash) to each switch. The current state C is 0.
- When a switch is toggled, xor the value C with the hash of that switch.
- If $C = 0$, nothing is on.
- If C is one of the individual hashes (stored in a map), then that individual switch is on.
- Otherwise, more than one switch is on.
- Memory used during suspend: $O(1)$.
- Crude probability estimate: if hashes are up to 10^{18}, the probability to mistakenly hit one of the 10^6 special states is 10^{-12} on each operation.

Statement

Problem H: Eager Sorting

- In this problem, we have to interactively sort a secret array.

Statement

Problem H: Eager Sorting

- In this problem, we have to interactively sort a secret array.
- We have 100 elements and 1500 operations.

Statement

Problem H: Eager Sorting

- In this problem, we have to interactively sort a secret array.
- We have 100 elements and 1500 operations.
- The operation is eager comparison: compare two elements and immediately swap them if unordered.

Statement

Problem H: Eager Sorting

- In this problem, we have to interactively sort a secret array.
- We have 100 elements and 1500 operations.
- The operation is eager comparison: compare two elements and immediately swap them if unordered.
- At some moment not known in advance, the interaction pauses. After resuming, we don't remember previous operations.

Statement

Problem H: Eager Sorting

- In this problem, we have to interactively sort a secret array.
- We have 100 elements and 1500 operations.
- The operation is eager comparison: compare two elements and immediately swap them if unordered.
- At some moment not known in advance, the interaction pauses. After resuming, we don't remember previous operations.
- **Genre:** adaptive algorithms.

Solution 1: Fast

- If we could reliably sort in 750 operations, one of the two runs would succeed.

Solution 1: Fast

- If we could reliably sort in 750 operations, one of the two runs would succeed.
- Perform a randomized quicksort! Or a mergesort.

Solution 1: Fast

- If we could reliably sort in 750 operations, one of the two runs would succeed.
- Perform a randomized quicksort! Or a mergesort.
- Quicksort: we have to adapt partition function to eager comparison.

Solution 1: Fast

- If we could reliably sort in 750 operations, one of the two runs would succeed.
- Perform a randomized quicksort! Or a mergesort.
- Quicksort: we have to adapt partition function to eager comparison.
- Maintain position of pivot.

Solution 1: Fast

- If we could reliably sort in 750 operations, one of the two runs would succeed.
- Perform a randomized quicksort! Or a mergesort.
- Quicksort: we have to adapt partition function to eager comparison.
- Maintain position of pivot.
- If pivot is not the leftmost one, compare it with the leftmost. Otherwise, compare it with the rightmost. In every case, either the left or the right border moves to the center.

Solution 2: Adaptive

- How to sort faster if the array is *almost sorted*?

Solution 2: Adaptive

- How to sort faster if the array is *almost sorted*?
- Adaptive sorting: Shellsort.

```
for step in [57, 23, 10, 4, 1]:
  for shift in [0, 1, ..., step - 1]:
     insertion sort for {a[shift + k * step]}
```

Solution 2: Adaptive

- How to sort faster if the array is *almost sorted*?
- Adaptive sorting: Shellsort.

```
for step in [57, 23, 10, 4, 1]:
  for shift in [0, 1, ..., step - 1]:
      insertion sort for {a[shift + k * step]}
```

- Worst case: $O(n^{3/2})$, $O(n^{4/3})$, $O(n \log^2 n)$, or unknown depending on the *gap sequence*.

Solution 2: Adaptive

- How to sort faster if the array is *almost sorted*?
- Adaptive sorting: Shellsort.

```
for step in [57, 23, 10, 4, 1]:
  for shift in [0, 1, ..., step - 1]:
     insertion sort for {a[shift + k * step]}
```

- Worst case: $O(n^{3/2})$, $O(n^{4/3})$, $O(n \log^2 n)$, or unknown depending on the *gap sequence*.
- Each gap sequence has worst cases, but a particular case will likely not appear.

Solution 2: Adaptive

- How to sort faster if the array is *almost sorted*?
- Adaptive sorting: Shellsort.

```
for step in [57, 23, 10, 4, 1]:
  for shift in [0, 1, ..., step - 1]:
     insertion sort for {a[shift + k * step]}
```

- Worst case: $O(n^{3/2})$, $O(n^{4/3})$, $O(n \log^2 n)$, or unknown depending on the *gap sequence*.
- Each gap sequence has worst cases, but a particular case will likely not appear.
- Another case: Timsort.

Solution 3: Library

- Another case of adaptive sorting: Timsort.

Solution 3: Library

- Another case of adaptive sorting: Timsort.
- Try to use library `stable_sort`.

Solution 3: Library

- Another case of adaptive sorting: Timsort.
- Try to use library `stable_sort`.
- Problem: eager comparison messes up the order.

Solution 3: Library

- Another case of adaptive sorting: Timsort.
- Try to use library `stable_sort`.
- Problem: eager comparison messes up the order.
- Maintain a permutation of the initial positions that is forced by eager comparison. Sort the array according to that permutation.

Solution 3: Library

- Another case of adaptive sorting: Timsort.
- Try to use library `stable_sort`.
- Problem: eager comparison messes up the order.
- Maintain a permutation of the initial positions that is forced by eager comparison. Sort the array according to that permutation.
- In the end, do one linear pass to move i-th sorted element to i-th real position for each i.

Statement

Problem I: Telepathy

- In this problem, we have to play for two brothers:
  - Each brother receives his own long random binary sequence, and picks positions in the other brother's binary sequence, not seeing that sequence.
  - After that, we compare the picked subsequences.
  - In at least $2/3$ positions, they must be equal.

Statement

Problem I: Telepathy

- In this problem, we have to play for two brothers:
  - Each brother receives his own long random binary sequence, and picks positions in the other brother's binary sequence, not seeing that sequence.
  - After that, we compare the picked subsequences.
  - In at least $2/3$ positions, they must be equal.
- Short example:
  - Let sequence a be 00101011011110111001.
  - Let sequence b be 11000111010000011010.
  - First brother selects positions $2, 3, 5, 7, 11$.
  - Second brother selects positions $1, 4, 9, 16, 20$.
  - First subsequence: 00011.
  - Second subsequence: 10011.
  - There are 4 equalities out of 5.

Statement

Problem I: Telepathy

- In this problem, we have to play for two brothers:
  - Each brother receives his own long random binary sequence, and picks positions in the other brother's binary sequence, not seeing that sequence.
  - After that, we compare the picked subsequences.
  - In at least $2/3$ positions, they must be equal.
- Short example:
  - Let sequence a be 00101011011110111001.
  - Let sequence b be 11000111101000011010.
  - First brother selects positions $2, 3, 5, 7, 11$.
  - Second brother selects positions $1, 4, 9, 16, 20$.
  - First subsequence: 00011.
  - Second subsequence: 10011.
  - There are 4 equalities out of 5.
- **Genre:** mathematical tricks.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.
- The first brother sees one of 4 outcomes, and picks one of 2 positions.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.
- The first brother sees one of 4 outcomes, and picks one of 2 positions.
- The second brother sees one of 4 outcomes, and picks one of 2 positions.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.
- The first brother sees one of 4 outcomes, and picks one of 2 positions.
- The second brother sees one of 4 outcomes, and picks one of 2 positions.
- Let us write a program to consider all possible $2^4 \cdot 2^4$ strategies, then pick a strategy which has the best chance of success.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.
- The first brother sees one of 4 outcomes, and picks one of 2 positions.
- The second brother sees one of 4 outcomes, and picks one of 2 positions.
- Let us write a program to consider all possible $2^4 \cdot 2^4$ strategies, then pick a strategy which has the best chance of success.
- To check a strategy, enumerate all $2^2 \cdot 2^2$ possible input sequences, and for each, see whether the brothers won.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.
- The first brother sees one of 4 outcomes, and picks one of 2 positions.
- The second brother sees one of 4 outcomes, and picks one of 2 positions.
- Let us write a program to consider all possible $2^4 \cdot 2^4$ strategies, then pick a strategy which has the best chance of success.
- To check a strategy, enumerate all $2^2 \cdot 2^2$ possible input sequences, and for each, see whether the brothers won.
- The best probability turns out to be $10/16$, or 0.625, not yet enough.

Solution: 2-Blocks

- Restrict the problem to both brothers seeing just two binary digits, and pick one position from these digits.
- The first brother sees one of 4 outcomes, and picks one of 2 positions.
- The second brother sees one of 4 outcomes, and picks one of 2 positions.
- Let us write a program to consider all possible $2^4 \cdot 2^4$ strategies, then pick a strategy which has the best chance of success.
- To check a strategy, enumerate all $2^2 \cdot 2^2$ possible input sequences, and for each, see whether the brothers won.
- The best probability turns out to be $10/16$, or 0.625, not yet enough.
- If it were enough, we would then cut out $k = 10^5$ blocks from the start and apply this solution to each block.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.
- The second brother sees one of 8 outcomes, and picks one of 3 positions.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.
- The second brother sees one of 8 outcomes, and picks one of 3 positions.
- Consider all possible $3^8 \cdot 3^8$ strategies, which is a bit too much already.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.
- The second brother sees one of 8 outcomes, and picks one of 3 positions.
- Consider all possible $3^8 \cdot 3^8$ strategies, which is a bit too much already.
- So, start with a random strategy, then make random changes and see if it got better (hill climbing).

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.
- The second brother sees one of 8 outcomes, and picks one of 3 positions.
- Consider all possible $3^8 \cdot 3^8$ strategies, which is a bit too much already.
- So, start with a random strategy, then make random changes and see if it got better (hill climbing).
- Or just precompute.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.
- The second brother sees one of 8 outcomes, and picks one of 3 positions.
- Consider all possible $3^8 \cdot 3^8$ strategies, which is a bit too much already.
- So, start with a random strategy, then make random changes and see if it got better (hill climbing).
- Or just precompute.
- To check a strategy, enumerate all $2^3 \cdot 2^3$ possible input sequences, and for each, see whether the brothers won.

Solution: 3-Blocks

- Restrict the problem to both brothers seeing three binary digits, and pick one position from these digits.
- The first brother sees one of 8 outcomes, and picks one of 3 positions.
- The second brother sees one of 8 outcomes, and picks one of 3 positions.
- Consider all possible $3^8 \cdot 3^8$ strategies, which is a bit too much already.
- So, start with a random strategy, then make random changes and see if it got better (hill climbing).
- Or just precompute.
- To check a strategy, enumerate all $2^3 \cdot 2^3$ possible input sequences, and for each, see whether the brothers won.
- The best probability turns out to be $44/64$, or 0.6875, which should be enough already.

Solution: d-Blocks

- Restrict the problem to both brothers seeing d binary digits, and pick one position from these digits.

Solution: d-Blocks

- Restrict the problem to both brothers seeing d binary digits, and pick one position from these digits.
- Use local optimization methods like hill climbing or threshold accepting.

Solution: d-Blocks

- Restrict the problem to both brothers seeing d binary digits, and pick one position from these digits.
- Use local optimization methods like hill climbing or threshold accepting.
- Precalculate a good strategy, then just use it in the submission.

Solution: d-Blocks

- Restrict the problem to both brothers seeing d binary digits, and pick one position from these digits.
- Use local optimization methods like hill climbing or threshold accepting.
- Precalculate a good strategy, then just use it in the submission.
- The probability seems to approach 0.7.

Statement

Problem J: Tetra-puzzle

- In this problem, we play a tetris-like puzzle:
  - there is a 5×5 board;
  - we place the given tetraminos on it interactively, one by one;
  - after each placement, all full rows and columns are simultaneously cleaned up;
  - we lose if we can't place the given tetramino;
  - the goal is to survive for 1000 turns.

Statement

Problem J: Tetra-puzzle

- In this problem, we play a tetris-like puzzle:
  - there is a 5×5 board;
  - we place the given tetraminos on it interactively, one by one;
  - after each placement, all full rows and columns are simultaneously cleaned up;
  - we lose if we can't place the given tetramino;
  - the goal is to survive for 1000 turns.
- Additionally, before the game, we get 1000 *pairs* of tetraminos non-interactively, and we have to choose one of the two pieces for each move in the main game.

Statement

Problem J: Tetra-puzzle

- In this problem, we play a tetris-like puzzle:
  - there is a 5×5 board;
  - we place the given tetraminos on it interactively, one by one;
  - after each placement, all full rows and columns are simultaneously cleaned up;
  - we lose if we can't place the given tetramino;
  - the goal is to survive for 1000 turns.
- Additionally, before the game, we get 1000 *pairs* of tetraminos non-interactively, and we have to choose one of the two pieces for each move in the main game.
- **Genre:** prepare and play.

Base Game: How to Place

- Establish a *scoring function*.

Base Game: How to Place

- Establish a *scoring function*.
- Start with the number of filled squares: the less, the better.

Base Game: How to Place

- Establish a *scoring function*.
- Start with the number of filled squares: the less, the better.
- Take into account the total perimeter of filled squares: the less, the better.

Base Game: How to Place

- Establish a *scoring function*.
- Start with the number of filled squares: the less, the better.
- Take into account the total perimeter of filled squares: the less, the better.
- When filled, center squares are more dangerous than border squares.

Base Game: How to Place

- Establish a *scoring function*.
- Start with the number of filled squares: the less, the better.
- Take into account the total perimeter of filled squares: the less, the better.
- When filled, center squares are more dangerous than border squares.
- More refinements are possible...

Base Game: How to Place

- Establish a *scoring function*.
- Start with the number of filled squares: the less, the better.
- Take into account the total perimeter of filled squares: the less, the better.
- When filled, center squares are more dangerous than border squares.
- More refinements are possible. . .
- Still, a good scoring function alone is unlikely to win.

Preparation: How to Choose

- Order the five kinds of tetraminos by their danger level.

Preparation: How to Choose

- Order the five kinds of tetraminos by their danger level.
- Out of each two, pick the best one according to this order.

Preparation: How to Choose

- Order the five kinds of tetraminos by their danger level.
- Out of each two, pick the best one according to this order.
- For example, "O" is one of the most dangerous kinds.

Preparation: How to Choose

- Order the five kinds of tetraminos by their danger level.
- Out of each two, pick the best one according to this order.
- For example, "O" is one of the most dangerous kinds.
- On different tests, different orders allow to survive for 1000 turns: with a reasonable scoring function, usually 2–10 permutations out of the possible 120.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.
- From each state, try the two given tetraminos, now we got up to $2w$ states for the next turn.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.
- From each state, try the two given tetraminos, now we got up to $2w$ states for the next turn.
- Sort them according to their score, and leave only up to w best states.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.
- From each state, try the two given tetraminos, now we got up to $2w$ states for the next turn.
- Sort them according to their score, and leave only up to w best states.
- After we arrive to turn 1000, restore the history of choices, and make it our pick for the preparation phase.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.
- From each state, try the two given tetraminos, now we got up to $2w$ states for the next turn.
- Sort them according to their score, and leave only up to w best states.
- After we arrive to turn 1000, restore the history of choices, and make it our pick for the preparation phase.
- As our choices given the tetraminos were deterministic, we will interactively play exactly the same game in the base phase.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.
- From each state, try the two given tetraminos, now we got up to $2w$ states for the next turn.
- Sort them according to their score, and leave only up to w best states.
- After we arrive to turn 1000, restore the history of choices, and make it our pick for the preparation phase.
- As our choices given the tetraminos were deterministic, we will interactively play exactly the same game in the base phase.
- The time limit allows w up to hundreds for casual implementation, perhaps more if we optimize.

Solution: Beam Search

- First, fix a deterministic way to play the base game: resolve ties of scoring function deterministically.
- In preparation mode, on each turn, maintain not one but up to w possible game states. Here, w is the width of the beam.
- From each state, try the two given tetraminos, now we got up to $2w$ states for the next turn.
- Sort them according to their score, and leave only up to w best states.
- After we arrive to turn 1000, restore the history of choices, and make it our pick for the preparation phase.
- As our choices given the tetraminos were deterministic, we will interactively play exactly the same game in the base phase.
- The time limit allows w up to hundreds for casual implementation, perhaps more if we optimize.
- With a reasonable scoring function, even $w = 10$ is enough to pass all the tests.

Statement

Problem K: Trijection

- In this problem, we have to invent a function from $A_n \cup B_n \cup C_n$ to itself, where
  - A_n is the set of skew polyominoes with perimeter $2n + 2$,
  - B_n is the set of 321-avoiding permutations of size n, and
  - C_n is the set of triangulations of a convex $(n + 2)$-gon.

Statement

Problem K: Trijection

- In this problem, we have to invent a function from $A_n \cup B_n \cup C_n$ to itself, where
  - A_n is the set of skew polyominoes with perimeter $2n + 2$,
  - B_n is the set of 321-avoiding permutations of size n, and
  - C_n is the set of triangulations of a convex $(n + 2)$-gon.
- The result $f(x)$ must be an object from a different set: for example, if x is from A_n, then $f(x)$ has to be from either B_n or C_n.

Statement

Problem K: Trijection

- In this problem, we have to invent a function from $A_n \cup B_n \cup C_n$ to itself, where
  - A_n is the set of skew polyominoes with perimeter $2n + 2$,
  - B_n is the set of 321-avoiding permutations of size n, and
  - C_n is the set of triangulations of a convex $(n + 2)$-gon.
- The result $f(x)$ must be an object from a different set: for example, if x is from A_n, then $f(x)$ has to be from either B_n or C_n.
- Finally, $f(f(x)) = x$ must hold for all x.

Statement

Problem K: Trijection

- In this problem, we have to invent a function from $A_n \cup B_n \cup C_n$ to itself, where
  - A_n is the set of skew polyominoes with perimeter $2n + 2$,
  - B_n is the set of 321-avoiding permutations of size n, and
  - C_n is the set of triangulations of a convex $(n + 2)$-gon.
- The result $f(x)$ must be an object from a different set: for example, if x is from A_n, then $f(x)$ has to be from either B_n or C_n.
- Finally, $f(f(x)) = x$ must hold for all x.
- These three sets have their size equal to Catalan number c_n.

Statement

Problem K: Trijection

- In this problem, we have to invent a function from $A_n \cup B_n \cup C_n$ to itself, where
  - A_n is the set of skew polyominoes with perimeter $2n + 2$,
  - B_n is the set of 321-avoiding permutations of size n, and
  - C_n is the set of triangulations of a convex $(n + 2)$-gon.
- The result $f(x)$ must be an object from a different set: for example, if x is from A_n, then $f(x)$ has to be from either B_n or C_n.
- Finally, $f(f(x)) = x$ must hold for all x.
- These three sets have their size equal to Catalan number c_n.
- The given n is such that c_n is even.

Statement

Problem K: Trijection

- In this problem, we have to invent a function from $A_n \cup B_n \cup C_n$ to itself, where
  - A_n is the set of skew polyominoes with perimeter $2n + 2$,
  - B_n is the set of 321-avoiding permutations of size n, and
  - C_n is the set of triangulations of a convex $(n + 2)$-gon.
- The result $f(x)$ must be an object from a different set: for example, if x is from A_n, then $f(x)$ has to be from either B_n or C_n.
- Finally, $f(f(x)) = x$ must hold for all x.
- These three sets have their size equal to Catalan number c_n.
- The given n is such that c_n is even.
- **Genre:** laborious problem.

Solution: Upper Level

- Enumerate the elements of each set.

Solution: Upper Level

- Enumerate the elements of each set.
- Establish a circular order on the three kinds of objects.

Solution: Upper Level

- Enumerate the elements of each set.
- Establish a circular order on the three kinds of objects.
- When the number of an object is $2k + 1$, transform it to object $2k$ of the next kind.

Solution: Upper Level

- Enumerate the elements of each set.
- Establish a circular order on the three kinds of objects.
- When the number of an object is $2k + 1$, transform it to object $2k$ of the next kind.
- When the number of an object is $2k$, transform it to object $2k + 1$ of the previous kind.

Solution: Middle Level

- It is convenient to transform all Catalan objects to one of the classic representations: regular bracket sequences or unlabeled binary trees.

Solution: Middle Level

- It is convenient to transform all Catalan objects to one of the classic representations: regular bracket sequences or unlabeled binary trees.
- For the classic representation, transform it to a number and back.

Solution: Middle Level

- It is convenient to transform all Catalan objects to one of the classic representations: regular bracket sequences or unlabeled binary trees.
- For the classic representation, transform it to a number and back.
-
```
struct Catalan
{
  bool [] brackets;

  string toBrackets ();
  static Catalan fromBrackets (string input);
  string toPolyomino ();
  static Catalan fromPolyomino (string input);
  string toPermutation ();
  static Catalan fromPermutation (string input);
  string toTriangulation ();
  static Catalan fromTriangulation (string input);
  Num toNumber ();
  static Catalan fromNumber (Num input, int n);
}
```

Solution: Lower Level Pointers

- Here are examples of conversion procedures from the problem's objects to bracket sequences.

Solution: Lower Level Pointers

- Here are examples of conversion procedures from the problem's objects to bracket sequences.
- Skew polyominoes:
  - There are two paths from bottom left to top right along the perimeter.
  - Consider the lower right path (drop the last edge) as X.
  - Consider the upper left path (drop the first edge) as Y.
  - Interleave X (right means "(", up means ")") and Y (up means "(", right means ")") to get a regular bracket sequence.

Solution: Lower Level Pointers

- Here are examples of conversion procedures from the problem's objects to bracket sequences.
- 321-avoiding permutations:
  - Greedily pick one increasing subsequence, S.
  - What's left is another increasing subsequence, T.
  - For example, $P =$ "2 4 1 5 6 8 3 7" transforms into $S =$ "2 4 5 6 8" and $T =$ "1 3 7".
  - Now proceed on the original permutation from left to right.
  - The elements of S become "(" and are put into a queue.
  - The elements of T become "()".
  - After each element of T, for each element of the queue that has no lesser elements left, produce an additional ")" and remove it from the queue.

Solution: Lower Level Pointers

- Here are examples of conversion procedures from the problem's objects to bracket sequences.
- Triangulations of an $(n+2)$-gon:
  - Start from the edge 1–$(n+2)$.
  - It is part of a triangle with vertices $p < q < r$, and we arrived from the edge p–r.
  - Recursively find another triangle with edge p–q and another triangle with edge q–r.
  - If the answers for the two triangles above are "A" and "B", the result for our triangle is "$(A)\,B$".

Contest Developer:

- Ivan Kazmenko

Special Thanks:

- Mikhail Ginzburg
- Natalya Ginzburg
- Oleg Hristenko
- Andrei Lopatin
- and all my family :)

Solutions, checkers, interactors, validators, channels, and generators written in the D programming language (`https://dlang.org`).

Questions?