Brute forcing the IOI

Schalk-Willem Krüger

Squad 2009

29 May 2009

Outline

1. Introduction
2. Type Printer
3. Islands
4. Fish
5. Summary

IOI 2008: Medal distribution

```
#define fully_solved(x) x.score>=90
```

Medal	Nr. of tasks “fully solved”						
	0	1	2	3	4	5	6
Gold	0 (0%)	0 (0%)	2 (8%)	17 (71%)	5 (21%)	0 (0%)	0 (0%)
Silver	1 (2%)	13 (28%)	31 (66%)	2 (4%)	0 (0%)	0 (0%)	0 (0%)
Bronze	4 (6%)	50 (71%)	16 (23%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
None	130 (48%)	12 (8%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

- 1 fully solved + few brute force $\approx$ BRONZE MEDAL
- 2 fully solved + few brute force $\approx$ SILVER MEDAL
- 3 fully solved + few brute force $\approx$ GOLD MEDAL

IOI 2008: Medal distribution

```
#define fully_solved(x) x.score>=90
```

Medal	Nr. of tasks “fully solved”						
	0	1	2	3	4	5	6
Gold	0 (0%)	0 (0%)	2 (8%)	17 (71%)	5 (21%)	0 (0%)	0 (0%)
Silver	1 (2%)	13 (28%)	31 (66%)	2 (4%)	0 (0%)	0 (0%)	0 (0%)
Bronze	4 (6%)	50 (71%)	16 (23%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
None	130 (48%)	12 (8%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

- 1 fully solved $+$ few brute force $\approx$ BRONZE MEDAL
- 2 fully solved $+$ few brute force $\approx$ SILVER MEDAL
- 3 fully solved $+$ few brute force $\approx$ GOLD MEDAL

IOI 2008: Medal distribution

```
#define fully_solved(x) x.score>=90
```

Medal	Nr. of tasks “fully solved”						
	0	1	2	3	4	5	6
Gold	0 (0%)	0 (0%)	2 (8%)	17 (71%)	5 (21%)	0 (0%)	0 (0%)
Silver	1 (2%)	13 (28%)	31 (66%)	2 (4%)	0 (0%)	0 (0%)	0 (0%)
Bronze	4 (6%)	50 (71%)	16 (23%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
None	130 (48%)	12 (8%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

- 1 fully solved $+$ few brute force $\approx$ BRONZE MEDAL
- 2 fully solved $+$ few brute force $\approx$ SILVER MEDAL
- 3 fully solved $+$ few brute force $\approx$ GOLD MEDAL

IOI 2008: Medal distribution

```
#define fully_solved(x) x.score>=90
```

Medal	Nr. of tasks “fully solved”						
	0	1	2	3	4	5	6
Gold	0 (0%)	0 (0%)	2 (8%)	17 (71%)	5 (21%)	0 (0%)	0 (0%)
Silver	1 (2%)	13 (28%)	31 (66%)	2 (4%)	0 (0%)	0 (0%)	0 (0%)
Bronze	4 (6%)	50 (71%)	16 (23%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
None	130 (48%)	12 (8%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

- 1 fully solved $+$ few brute force $\approx$ BRONZE MEDAL
- 2 fully solved $+$ few brute force $\approx$ SILVER MEDAL
- 3 fully solved $+$ few brute force $\approx$ GOLD MEDAL

Difficulties making this presentation

- No “official” IOI 2008 evaluator online! Can download official test data.
- Two “unofficial” evaluators I used: SMS and SPOJ
- Difficulties with SMS:
  - IOI has different marks per test group. SMS can’t handle it. (Converted score to get real score)
  - I couldn’t get the “real” score per test case of all the problems.
  - SMS not the same speed as the original evaluator.
  - Test data in different order (detailed feedback data are first).
- Difficulties with SPOJ:
  - No grouped test cases!
  - Can’t handle different marks per test case.
  - No detailed feedback - only score.
  - Not same speed.
- Summary: Scores might be inaccurate.
- I used SMS to approximate scores.

Difficulties making this presentation

- No “official” IOI 2008 evaluator online! Can download official test data.
- Two “unofficial” evaluators I used: SMS and SPOJ
- Difficulties with SMS:
  - IOI has different marks per test group. SMS can’t handle it. (Converted score to get real score)
  - I couldn’t get the “real” score per test case of all the problems.
  - SMS not the same speed as the original evaluator.
  - Test data in different order (detailed feedback data are first).
- Difficulties with SPOJ:
  - No grouped test cases!
  - Can’t handle different marks per test case.
  - No detailed feedback - only score.
  - Not same speed.
- Summary: Scores might be inaccurate.
- I used SMS to approximate scores.

Difficulties making this presentation

- No “official” IOI 2008 evaluator online! Can download official test data.
- Two “unofficial” evaluators I used: SMS and SPOJ
- Difficulties with SMS:
  - IOI has different marks per test group. SMS can’t handle it. (Converted score to get real score)
  - I couldn’t get the “real” score per test case of all the problems.
  - SMS not the same speed as the original evaluator.
  - Test data in different order (detailed feedback data are first).
- Difficulties with SPOJ:
  - No grouped test cases!
  - Can’t handle different marks per test case.
  - No detailed feedback - only score.
  - Not same speed.
- Summary: Scores might be inaccurate.
- I used SMS to approximate scores.

Difficulties making this presentation

- No “official” IOI 2008 evaluator online! Can download official test data.
- Two “unofficial” evaluators I used: SMS and SPOJ
- Difficulties with SMS:
    - IOI has different marks per test group. SMS can’t handle it. (Converted score to get real score)
    - I couldn’t get the “real” score per test case of all the problems.
    - SMS not the same speed as the original evaluator.
    - Test data in different order (detailed feedback data are first).
- Difficulties with SPOJ:
    - No grouped test cases!
    - Can’t handle different marks per test case.
    - No detailed feedback - only score.
    - Not same speed.
- Summary: Scores might be inaccurate.
- I used SMS to approximate scores.

Difficulties making this presentation

- No “official” IOI 2008 evaluator online! Can download official test data.
- Two “unofficial” evaluators I used: SMS and SPOJ
- Difficulties with SMS:
  - IOI has different marks per test group. SMS can’t handle it. (Converted score to get real score)
  - I couldn’t get the “real” score per test case of all the problems.
  - SMS not the same speed as the original evaluator.
  - Test data in different order (detailed feedback data are first).
- Difficulties with SPOJ:
  - No grouped test cases!
  - Can’t handle different marks per test case.
  - No detailed feedback - only score.
  - Not same speed.
- Summary: Scores might be inaccurate.
- I used SMS to approximate scores.

Difficulties making this presentation

- No “official” IOI 2008 evaluator online! Can download official test data.
- Two “unofficial” evaluators I used: SMS and SPOJ
- Difficulties with SMS:
  - IOI has different marks per test group. SMS can’t handle it. (Converted score to get real score)
  - I couldn’t get the “real” score per test case of all the problems.
  - SMS not the same speed as the original evaluator.
  - Test data in different order (detailed feedback data are first).
- Difficulties with SPOJ:
  - No grouped test cases!
  - Can’t handle different marks per test case.
  - No detailed feedback - only score.
  - Not same speed.
- Summary: Scores might be inaccurate.
- I used SMS to approximate scores.

Outline

1 Introduction

2 Type Printer

3 Islands

4 Fish

5 Summary

Task description

- Print N words on a movable type printer. The printer has the following operations:
  - Add a letter to the end of the current word.
  - Remove the last letter from the end of the current word.
  - Print the current word.
- Initially, the printer is empty.
- You are allowed to leave some letters in the printer.
- You are allowed to print the words in any order.
- Minimize the total number of operations.
- Summary: Given N words, find the **minimum number of operations** needed to print all the words in any order, and output one such sequence of operators.

Input, output

Input:

- The N ($1 \leq N \leq 25\,000$) words.
- Each word's length is between 1 and 20, inclusive.
- In 40% of the test cases, $N \leq 18$.

Output:

- Operations:
  - Add letter: letter itself.
  - Remove letter: '-'
  - Print word: 'P'

Input	Output
3	20
print	t
the	h
poem	e
	P
	-
	-
	-
	p
	o
	e
	m
	P
	-
	-
	-
	r
	i
	n
	t
	P

Solution: Brute force

- We can try to brute force it: DFS.
- Precompute the common prefix length between each pair of words.
- In the DFS function: Try to go to each and every word not in the stack.

- Time complexity: $O(N!)$.
- For 40% of test cases: $18! = 6\,402\,373\,705\,728\,000$
- DFS will give you only 10%.

100% solution

Any suggestions?

100% solution

- Use a trie!
- We don’t have to delete the last word — we save a few ’minus’ commands.
- New goal: Maximize the length of the last word.
- Find the longest word and mark the nodes in the trie we visit when reading this word.
- Run DFS search on trie, but at each node, first process all the children that are not marked.
- This will force the DFS to end with the longest word.

100% solution

- Use a trie!
- We don't have to delete the last word — we save a few 'minus' commands.
- New goal: Maximize the length of the last word.
- Find the longest word and mark the nodes in the trie we visit when reading this word.
- Run DFS search on trie, but at each node, first process all the children that are not marked.
- This will force the DFS to end with the longest word.

100% solution

- Use a trie!
- We don’t have to delete the last word — we save a few ’minus’ commands.
- New goal: Maximize the length of the last word.
- Find the longest word and mark the nodes in the trie we visit when reading this word.
- Run DFS search on trie, but at each node, first process all the children that are not marked.
- This will force the DFS to end with the longest word.

100% solution

- Use a trie!
- We don’t have to delete the last word — we save a few ’minus’ commands.
- New goal: Maximize the length of the last word.
- Find the longest word and mark the nodes in the trie we visit when reading this word.
- Run DFS search on trie, but at each node, first process all the children that are not marked.
- This will force the DFS to end with the longest word.

100% solution

- Use a trie!
- We don’t have to delete the last word — we save a few ’minus’ commands.
- New goal: Maximize the length of the last word.
- Find the longest word and mark the nodes in the trie we visit when reading this word.
- Run DFS search on trie, but at each node, first process all the children that are not marked.
- This will force the DFS to end with the longest word.

Score distribution

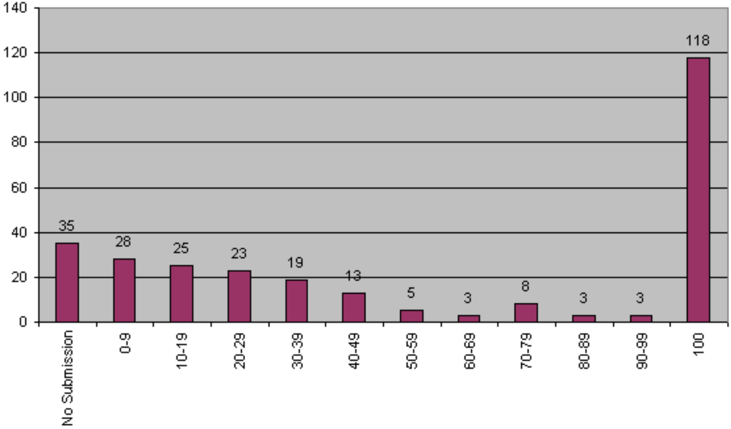


Figure: Score distribution

Outline

1 Introduction

2 Type Printer

3 Islands

4 Fish

5 Summary

Task description

- There are *N* islands.
- From each island, exactly one bi-directional bridge was constructed.
- There are *N* islands and *N* (bi-directional) bridges
- Each island has at least one bridge.
- Each bridge has a certain length.
- Also, there is a unique ferry that travels back and forth between each pair of islands.
- You must **maximize the sum of the lengths of the bridges you cross**.
- You can go from one island to another by:
  - Walking: Only possible if there is a bridge between the two islands.
  - Ferry: Only possible if the one island is not reachable from the other using any combination of bridges **and/or previously used ferries**.
- You do not have to visit all the islands, and it may be impossible to cross all the bridges.

Task description

- There are N islands.
- From each island, exactly one bi-directional bridge was constructed.
- There are N islands and N (bi-directional) bridges
- Each island has at least one bridge.
- Each bridge has a certain length.
- Also, there is a unique ferry that travels back and forth between each pair of islands.
- You must **maximize the sum of the lengths of the bridges you cross**.
- You can go from one island to another by:
    - Walking: Only possible if there is a bridge between the two islands.
    - Ferry: Only possible if the one island is not reachable from the other using any combination of bridges **and/or previously used ferries**.
- You do not have to visit all the islands, and it may be impossible to cross all the bridges.

Task description

- There are N islands.
- From each island, exactly one bi-directional bridge was constructed.
- There are N islands and N (bi-directional) bridges
- Each island has at least one bridge.
- Each bridge has a certain length.
- Also, there is a unique ferry that travels back and forth between each pair of islands.
- You must **maximize the sum of the lengths of the bridges you cross**.
- You can go from one island to another by:
    - Walking: Only possible if there is a bridge between the two islands.
    - Ferry: Only possible if the one island is not reachable from the other using any combination of bridges **and/or previously used ferries**.
- You do not have to visit all the islands, and it may be impossible to cross all the bridges.

Task description

- There are *N* islands.
- From each island, exactly one bi-directional bridge was constructed.
- There are *N* islands and *N* (bi-directional) bridges
- Each island has at least one bridge.
- Each bridge has a certain length.
- Also, there is a unique ferry that travels back and forth between each pair of islands.
- You must **maximize the sum of the lengths of the bridges you cross**.
- You can go from one island to another by:
  - Walking: Only possible if there is a bridge between the two islands.
  - Ferry: Only possible if the one island is not reachable from the other using any combination of bridges **and/or previously used ferries**.
- You do not have to visit all the islands, and it may be impossible to cross all the bridges.

Task description

- There are *N* islands.
- From each island, exactly one bi-directional bridge was constructed.
- There are *N* islands and *N* (bi-directional) bridges
- Each island has at least one bridge.
- Each bridge has a certain length.
- Also, there is a unique ferry that travels back and forth between each pair of islands.
- You must **maximize the sum of the lengths of the bridges you cross**.
- You can go from one island to another by:
    - Walking: Only possible if there is a bridge between the two islands.
    - Ferry: Only possible if the one island is not reachable from the other using any combination of bridges **and/or previously used ferries**.
- You do not have to visit all the islands, and it may be impossible to cross all the bridges.

Constraints

Constraints:

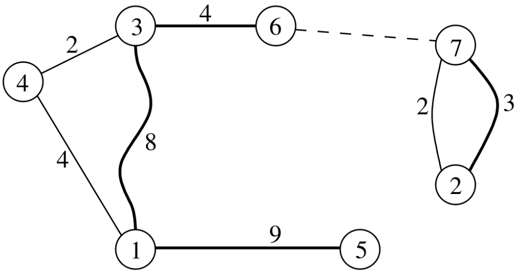
- $2 \leq N$ (Number of islands) $\leq 1\,000\,000$
- $1 \leq$ Length of bridge $\leq 100\,000\,000$
- **In 40% of the test cases,** $N \leq 4\,000$.

Input:

- For each island, the length of the bridge and the index of the island it is connected to, are given.

Sample

- Suppose there are seven islands ($N = 7$) and there are bridges connecting (1-3), (2-7), (3-4), (4-1), (5-1), (6-3), (7-2).
- One way to achieve maximum walking distance: $5 \rightarrow 1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 2$
- This gives a total walking distance of $9 + 8 + 4 + 3 = 24$.



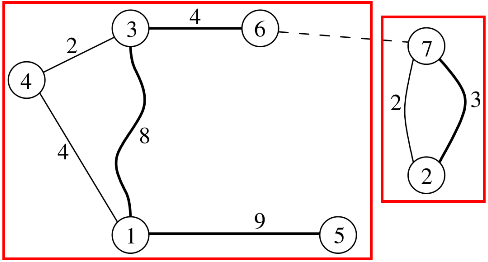
- Note: You cannot visit island 4.

Brute force

- Do a DFS: Go recursively from each island to each and every unvisited island that subjects to the constraints.
- Extremely slow - about $O(N!)$.
- This is similar to the DFS solution of “Type Printer”.
- If you can’t come up with a better solution, use this.
- You will score about 6 points.

40% solution

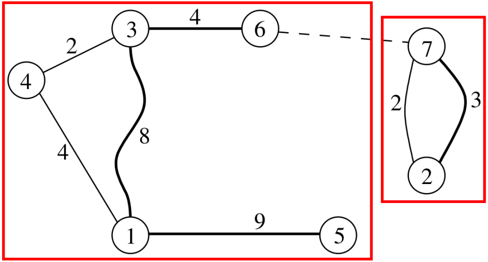
- Consider it as a graph with different connected components.



- You cannot use a ferry to jump within a connected component.
- You only have to find the longest weighted path in each component.
- The answer is the sum of the longest weighted path in each component.
- If you brute force each component to get the longest path, you will score 40 points!

40% solution

- Consider it as a graph with different connected components.



- You cannot use a ferry to jump within a connected component.
- You only have to find the longest weighted path in each component.
- The answer is the sum of the longest weighted path in each component.
- If you brute force each component to get the longest path, you will score 40 points!

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from *any* vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from *any* vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
    - Start from any vertex: A.
    - Find the furthest vertex, B, from it.
    - Find the furthest vertex, C, from it.
    - The longest path is the distance between B and C.
    - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from any vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C.
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from any vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C.
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from any vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C.
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
    - Start from any vertex A.
    - Find the furthest vertex, B, from it.
    - Find the furthest vertex, C, from it.
    - The longest path is the distance between B and C.
    - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from any vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C.
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
    - Start from any vertex A.
    - Find the furthest vertex, B, from it.
    - Find the furthest vertex, C, from it.
    - The longest path is the distance between B and C.
    - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from any vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C.
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Faster solution

- For each connected component: $E = V$.
- Each connected component will have exactly ONE cycle.
- If we remove one edge of the cycle, the connected component becomes a tree — there is no cycle.
- For each edge in a cycle, try to remove it and calculate the longest path.
- To get the longest path in a tree:
  - Start from any vertex A.
  - Find the furthest vertex, B, from it.
  - Find the furthest vertex, C, from it.
  - The longest path is the distance between B and C.
  - Proof: Kosie and Francois's presentation during the first training camp of 2009.
- Complexity: $O(NC)$ where C is the number of vertices on a cycle.

Even faster solution

- I'm not going into more detail.
- With some more optimization, you can get a $O(C^2)$ solution.
- Complexity of 100% solution: $O(N)$.

Score distribution

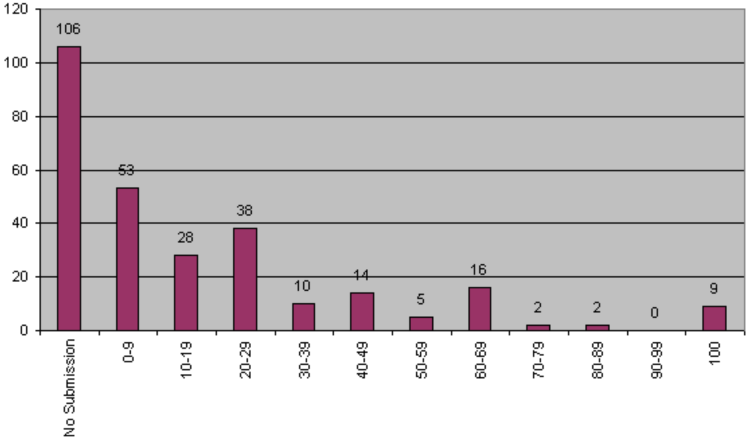


Figure: Islands score distribution among IOI contestants

Outline

1. Introduction
2. Type Printer
3. Islands
4. Fish
5. Summary

Task description

- There is a lake with F ($1 \leq F \leq 500\,000$) fish in it.
- Each fish was given one of K different gemstones.
- After that some fish ate some other fish.
- A fish can only eat another fish only if its length is at least twice as long.
- One fish might eat several smaller fish.
- The length of the fish doesn't change.
- Given: Length of each fish, kind of gemstone originally swallowed by each fish.
- Task: How many different combinations of gems could you obtain by catching a single fish.

Suggestions?

- In 70% of the test cases, $K \leq 7\,000$.
- In 25% of the test cases, $K \leq 20$.
- Any suggestions?

Suggestions?

- In 70% of the test cases, $K \leq 7\,000$.
- In 25% of the test cases, $K \leq 20$.
- Any suggestions?

Suggestions?

- In 70% of the test cases, $K \leq 7\,000$.
- In 25% of the test cases, $K \leq 20$.
- Any suggestions?

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can’t come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can't come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force

- Use STL set and multiset.
- Generate each combination using DFS and put it in the set.
- Save the combination in a multiset.
- Running time: Extremely slow.
- Adding the multiset each time to the set makes the algorithm even slower.
- Only use if you can’t come up with something better.
- You will get two of the 20 test cases right (about 6% - 10%).
- Extract from source:

```
typedef multiset<int> combination;
set<combination> already;
vector<pair<int,int> > fishinfo; // Length, gemstone
...
void add_fish(combination fish, int current, int add) { // DFS
...
}
printf("%d\n", already.size()%M);
```

Brute force: Subsets

- Generate all subsets with DFS / using bits.
- Keep array with counters of each kind of gemstrone.
- Test each subset - if it is valid, add array (DFS) or integer (bits) to the set.
- Will score about 20 points.
- Complexity: $O(2^N)$ (Main loop). Adding it each time to the set slows it down.

Brute force: Subsets

- Generate all subsets with DFS / using bits.
- Keep array with counters of each kind of gemstrone.
- Test each subset - if it is valid, add array (DFS) or integer (bits) to the set.
- Will score about 20 points.
- Complexity: $O(2^N)$ (Main loop). Adding it each time to the set slows it down.

Optimizations

- Unless a fish is the longest one of its kind, it will have no combinations mapped to it.
- Will score about 25 points.
- Complexity: $O(2^K)$ (Main loop). Adding it each time to the set slows it down.

100% solution

- Any suggestions?
- Will be left as an exercise ;-)

100% solution

- Any suggestions?
- Will be left as an exercise ;-)

Score distribution

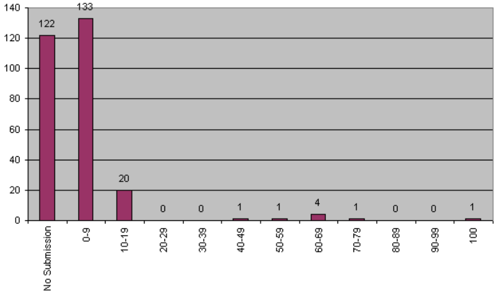


Figure: Fish score distribution among IOI contestants

Note:

- This problem is hard.
- Most of the contestants do try to write an easy brute force solution for harder problems. It can determine whether you just make that medal or not.

Score distribution

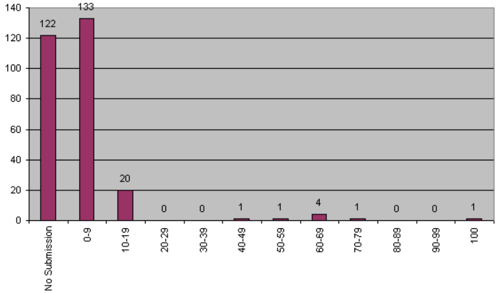


Figure: Fish score distribution among IOI contestants

Note:

- This problem is hard.
- Most of the contestants do try to write an easy brute force solution for harder problems. It can determine whether you just make that medal or not.

Outline

1 Introduction

2 Type Printer

3 Islands

4 Fish

5 Summary

Medal Cut-Offs

Year	Bronze	Silver	Gold	Top score
2008	127	229	356	558
2007	187	286	388	574
2006	219	314	385	480
2005	275	393	496	600
2004	265	365	445	565
2003	173	258	351	455
2002	135	226	296	510
Average	**197**	**295**	**388**	**534**

Summary

- Suppose you score 100 points for type, 40 points for islands and 20 points for fish — that is a total of 160 points!
- The bronze medal cutoff for last year was 135 points.
- You solved only half of the problems, and you already have a bronze medal.
- Add another 70 points for day 2 and you have a silver medal.

Why brute force?

- Every point matters! Even if you only get 10 more points, why not?
- Grouped test cases: Problems must get correct answers.
- Use brute force as a backup. *if (small input): brute_force() else: optimal()*.
- Remember to test your brute force solutions!

Why brute force?

- Every point matters! Even if you only get 10 more points, why not?
- Grouped test cases: Problems must get correct answers.
- Use brute force as a backup. *if (small input): brute_force() else: optimal()*.
- Remember to test your brute force solutions!

Why brute force?

- Every point matters! Even if you only get 10 more points, why not?
- Grouped test cases: Problems must get correct answers.
- Use brute force as a backup. *if (small input): brute_force() else: optimal()*.
- Remember to test your brute force solutions!

Why brute force?

- Every point matters! Even if you only get 10 more points, why not?
- Grouped test cases: Problems must get correct answers.
- Use brute force as a backup. *if (small input): brute_force() else: optimal()*.
- Remember to test your brute force solutions!

Questions?

?