DAPC 2023 Training Sessions
Session 2

Verwoerd

September 18, 2023

Session 2

- Team Tactics
- Utilizing the Test Session
- How to select problems
- Solutions to the Ad-hoc and Math Problems
- Solving Sorting and Search Problems

Team Tactics

General Tactics

- Know each other's strengths and weaknesses like:
    - types of problems (math, geometry, search, strings, graphs, etc.)
    - debugging skills
    - coding speed and accuracy
- Parallelize
- Work on paper (e.g. pseudocode or flow diagrams)
- Debug on paper
- Use rubber duck debugging when stuck

Team Tactics

- Plot of the contest: 3 contestants, 1 computer

Team Tactics

- Plot of the contest: 3 contestants, 1 computer
- Several tactics how to divide the computer efficiently

Team Tactics

- Plot of the contest: 3 contestants, 1 computer
- Several tactics how to divide the computer efficiently
- Shuffle Tactic
- Designated Tactic
- No best solution: Pick and mix what works best for your team

Team Tactics

- Plot of the contest: 3 contestants, 1 computer
- Several tactics how to divide the computer efficiently
- Shuffle Tactic
  - Rotate around who sits behind the pc
  - After submitting a problem, switch around if someone has a solution
  - Useful when programming in different languages
- Designated Tactic

- No best solution: Pick and mix what works best for your team

Team Tactics

- Plot of the contest: 3 contestants, 1 computer
- Several tactics how to divide the computer efficiently
- Shuffle Tactic
    - Rotate around who sits behind the pc
    - After submitting a problem, switch around if someone has a solution
    - Useful when programming in different languages
- Designated Tactic
    - Dedicated person behind computer
    - Other team members work on paper or read along on screen
    - Useful for teams with different disciplines
- No best solution: Pick and mix what works best for your team

Common Battle Plan

Start of contest Prepare computer, find and solve easiest problems, all problems should be read by at least a single team member.

First hours Prioritize solving the easiest problems, every team member works on their own problems

Mid contest Work on solving harder problems with 2 people, while the last person works alone on the last easy or specialized hard problems

End of the contest Work together with the whole team on a single problem, free submit mode

Common Errors

- Focus on the first problem you think you can solve
- Not reading all problems in the set
- Debugging on the computer while another solution can be implemented
- Fighting who can solve which problem
- Not rewriting code when it gets to messy

Utilizing the Test Session

Utilizing the Test Session

- The test session is a short version practice contest with a few simple problems
- Find your team workspace
- Practice start of the contest, e.g. where are the problems located, how is the start announced
- Practice end of the contest, e.g. freeze warning, countdown.

Testing the envirionment

- Get to know the programming environment and its quirks

Testing the envirionment

- Get to know the programming environment and its quirks
- Where is everything located: samples, documentation, important links

Testing the envirionment

- Get to know the programming environment and its quirks
- Where is everything located: samples, documentation, important links
- Try all IDE and tools you *might* use during the contest

Testing the envirionment

- Get to know the programming environment and its quirks
- Where is everything located: samples, documentation, important links
- Try all IDE and tools you *might* use during the contest
- Test out printing, if supported, from IDE, command line, and CCS

Testing the envirionment

- Get to know the programming environment and its quirks
- Where is everything located: samples, documentation, important links
- Try all IDE and tools you *might* use during the contest
- Test out printing, if supported, from IDE, command line, and CCS
- All files will be removed after the test session!

Testing the Contest System

- Test the possible problems, how are they reported back
    - Correct (AC)
    - Wrong Answer (WA)
    - Run Time Error (RTE), like segmentation fault, going over heap/stack space, null pointer exception, array index out of bounds
    - Time Limit Exceeded (TLE)
    - Compiler Error
- Test clarification requests
- Where are general clarifications displayed

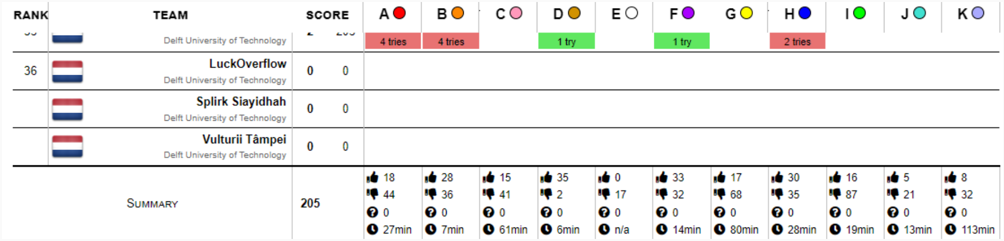
Hints on selecting the problems

Selecting the first problem

- Decide on a reading tactic
  - Do we all start reading the first problem?
  - Or does one person start at the end?
- There are usually several “simple” problems in a set
- Be careful: the easiest problems usually contain some pitfall corner cases!

Finding the easiest problems by results

- After a few minutes of contest, the first balloons will be handed out
- Check the scoreboard or balloon colours to see which problem is solved most



RANK	TEAM	SCORE		A ●	B ●	C ●	D ●	E ○	F ●	G ●	H ●	I ●	J ●	K ●
	Delft University of Technology			4 tries	4 tries		1 try		1 try		2 tries			
36	LuckOverflow Delft University of Technology	0	0											
	Splirk Siayidhah Delft University of Technology	0	0											
	Vulturii Tâmpei Delft University of Technology	0	0											
	SUMMARY	205		👍 18 👎 44 ❓ 0 🕐 27min	👍 28 👎 36 ❓ 0 🕐 7min	👍 15 👎 41 ❓ 0 🕐 61min	👍 35 👎 2 ❓ 0 🕐 6min	👍 0 👎 17 ❓ 0 🕐 n/a	👍 33 👎 32 ❓ 0 🕐 14min	👍 17 👎 68 ❓ 0 🕐 80min	👍 30 👎 35 ❓ 0 🕐 28min	👍 16 👎 87 ❓ 0 🕐 19min	👍 5 👎 21 ❓ 0 🕐 13min	👍 8 👎 32 ❓ 0 🕐 113min

- Or the problems page in DOMJudge (only newer versions)
- **Warning**: The first problem solved is not guaranteed the easiest!

My problem is wrong, what now

Print out the problem and let other people work on the computer, work out cases that might go wrong.

- When the result is RTE:
  - Check for possible null pointers, array overflows, or integer overflow
  - Check the input specification, don't forget 0 can do unexpected things
- When the result is TLE:
  - Check stop conditions, maybe an infinite loop?
  - Code is too slow, try optimizing or thinking of a faster solution
- When then result is WA:
  - Check for corner cases, don't forget zero
  - Check correctness of algorithm
- **Warning**: A problem can be WA and TLE at the same time, but only *one* is reported back!

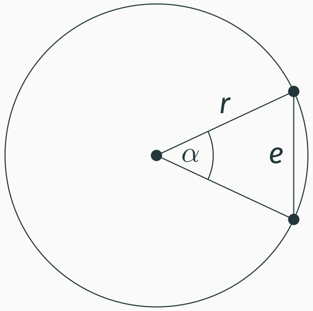
Solutions to the ad-hoc and math problems

Jabbing Jets

- Source BAPC Preliminaries 2022
- Time limit: 1s
- Given n concentric circles, find the maximal number of points on these circles such that the distance between any two points is at least e.

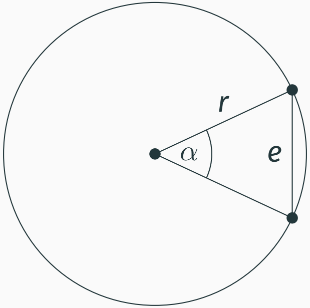
Jabbing Jets

- Observation 1: $n \leq 10^4$ and time limit of 1s so looking for $\mathcal{O}(n)$ solution



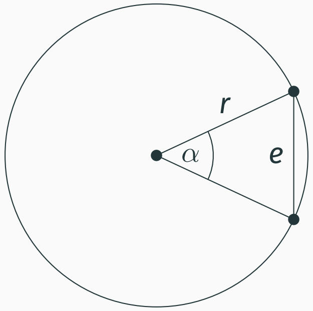
Jabbing Jets

- Observation 1: $n \leq 10^4$ and time limit of 1s so looking for $\mathcal{O}(n)$ solution
- Observation 2: Because $r_{i+1} - r_i \geq e$ every circle can be considered separately.



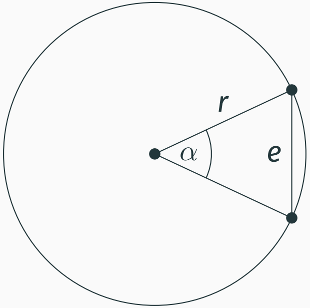
Jabbing Jets

- Observation 1: $n \leq 10^4$ and time limit of 1s so looking for $\mathcal{O}(n)$ solution
- Observation 2: Because $r_{i+1} - r_i \geq e$ every circle can be considered separately.
- 2 points are divided by angle α
- You can calculate $\alpha = 2 \arcsin\left(\frac{e}{2r}\right)$



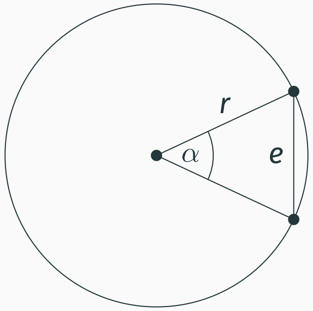
Jabbing Jets

- Observation 1: $n \leq 10^4$ and time limit of 1s so looking for $\mathcal{O}(n)$ solution
- Observation 2: Because $r_{i+1} - r_i \geq e$ every circle can be considered separately.
- 2 points are divided by angle α
- You can calculate $\alpha = 2\arcsin\left(\frac{e}{2r}\right)$
- The radius of a circle is 2π so the max number of points with distance e can be calculated by $\left\lfloor \frac{2\pi}{\alpha} \right\rfloor \equiv \left\lfloor \frac{2\pi}{2\arcsin\left(\frac{e}{2r}\right)} \right\rfloor$



Jabbing Jets

- Observation 1: $n \leq 10^4$ and time limit of 1s so looking for $\mathcal{O}(n)$ solution
- Observation 2: Because $r_{i+1} - r_i \geq e$ every circle can be considered separately.
- 2 points are divided by angle α
- You can calculate $\alpha = 2\arcsin\left(\frac{e}{2r}\right)$
- The radius of a circle is 2π so the max number of points with distance e can be calculated by $\left\lfloor \frac{2\pi}{\alpha} \right\rfloor \equiv \left\lfloor \frac{2\pi}{2\arcsin\left(\frac{e}{2r}\right)} \right\rfloor$
- Do this for every circle and sum the numbers
- Pitfall 1: if $2r < e$ then there can only be one point
- Pitfall 2: rounding can cause issues, add $0.5 \cdot 10^{-6}$



Solution for Jabbing Jets

```python
from math import *

n, e = (int(i) for i in input().split())
res = 0
for r in (int(i) for i in input().split()):
  res += int(pi/asin(e/2/(r+10**-10))) if 2*r >= e else 1
print(res)
```

Lots of Liquid

- Source BAPC Preliminaries 2022
- Time limit: 1s
- Find the length of the side of a cube that contains all liquid.

Lots of Liquid

- Observation: $n \leq 10^5$, so we are looking for a $\mathcal{O}(n)$ solution

Lots of Liquid

- Observation: $n \leq 10^5$, so we are looking for a $\mathcal{O}(n)$ solution
- The volume of a cube is c^3

Lots of Liquid

- Observation: $n \leq 10^5$, so we are looking for a $\mathcal{O}(n)$ solution
- The volume of a cube is c^3
- The length of cube with volume v is $\sqrt[3]{v}$

Lots of Liquid

- Observation: $n \leq 10^5$, so we are looking for a $\mathcal{O}(n)$ solution
- The volume of a cube is c^3
- The length of cube with volume v is $\sqrt[3]{v}$
- Sum all volumes of the cubes and calculate the length of the cube
- Print out

$$\sqrt[3]{\sum_{i=1}^{n} c_i}$$

Lots of Liquid

```python
input()
print(sum(map(lambda x: float(x) ** 3, input().split())) ** (1 / 3))
```

Bellevue

- Source BAPC 2022
- Time limit: 1s
- Given the profile of an island, find the point with the largest viewing angle of the sea.

Bellevue

- The answer is always an angle from the start or the end of the island to another point

Bellevue

- The answer is always an angle from the start or the end of the island to another point
- Calculate the angle from start and end to any other point
- Print out the solution

Bellevue

- The answer is always an angle from the start or the end of the island to another point
- Calculate the angle from start and end to any other point
- Print out the solution
- Alternatively, calculate the convex hull
    - The best angle is from either the first and second point in the hull or the last and previous

Bellevue

```python
import math
n = int(input())
ps = [list(map(int, input().split())) for _ in range(n)]

def solve(ps):
  start = ps[0]
  best = ps[1]
  for p in ps[1:]:
    if p[1] / abs(p[0] - start[0]) >= best[1] / abs(best[0] - start[0]):
      best = p
  return math.atan2(best[1], abs(best[0] - start[0])) * 360 / (2 * math.pi)

ans_left = solve(ps)
ans_right = solve(ps[::-1])
print(f"{max(ans_left, ans_right):0.7f}")
```

Equalising Audio

- Source BAPC 2022
- Time limit: 4s
- Given a list of frequencies, normalize it so the perceived loudness is

$$\frac{1}{n}\sum_{i=1}^{n} a_i^2 = x$$

Equalising Audio

- Observation: time limit is high for I/O operations

Equalising Audio

- Observation: time limit is high for I/O operations
- First calculate the current perceived loudness

$$x_{cur} = \frac{1}{n}\sum_{i=1}^{n} a_i^2$$

Equalising Audio

- Observation: time limit is high for I/O operations
- First calculate the current perceived loudness

$$x_{cur} = \frac{1}{n}\sum_{i=1}^{n} a_i^2$$

- Then print out the frequency reduced by $\sqrt{\frac{x}{x_{cur}}}$, since

$$\frac{1}{n}\sum_{i=1}^{n}\left(\sqrt{\frac{x}{x_{cur}}}a_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\sqrt{\frac{x}{x_{cur}}}\right)^2 a_i^2 = \frac{x}{x_{cur}}\cdot\frac{1}{n}\sum_{i=1}^{n}a_i^2 = \frac{x}{x_{cur}}\cdot x_{cur} = x$$

Equalising Audio

- Observation: time limit is high for I/O operations
- First calculate the current perceived loudness

$$x_{cur} = \frac{1}{n} \sum_{i=1}^{n} a_i^2$$

- Then print out the frequency reduced by $\sqrt{\frac{x}{x_{cur}}}$, since

$$\frac{1}{n} \sum_{i=1}^{n} \left(\sqrt{\frac{x}{x_{cur}}} a_i \right)^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\sqrt{\frac{x}{x_{cur}}} \right)^2 a_i^2 = \frac{x}{x_{cur}} \cdot \frac{1}{n} \sum_{i=1}^{n} a_i^2 = \frac{x}{x_{cur}} \cdot x_{cur} = x$$

- Pitfall: if the current perceived loudness (x_{cur}) is 0, the result is `0 0 … 0`

Equalising Audio

```python
from math import sqrt
n, x = map(int, input().split())
A = list(map(int, input().split()))
y = sum(map(lambda a: a**2, A)) / n
print(*[0 if y == 0 else sqrt(x / y) * a for a in A])
```

Failing Flagship

- Source BAPC 2022
- Time limit: 1s
- Compute the minimum angle in degrees between two wind directions.

Failing Flagship

- Convert the letters to degrees d_1 and d_2

Failing Flagship

- Convert the letters to degrees d_1 and d_2
- Execute the algorithm as described in the problem by adding and subtracting a delta, starting at $45\,\mathrm{deg}$

Failing Flagship

- Convert the letters to degrees d_1 and d_2
- Execute the algorithm as described in the problem by adding and subtracting a delta, starting at $45\,\mathrm{deg}$
- Every next letter represents a half of previous delta and either add or subtract it

Failing Flagship

- Convert the letters to degrees d_1 and d_2
- Execute the algorithm as described in the problem by adding and subtracting a delta, starting at $45\,\mathrm{deg}$
- Every next letter represents a half of previous delta and either add or subtract it
- Be careful of overflow with NW
- Print out $\min(d_2 - d_1, 360 + d_1 - d_2)$

Failing Flagship

```python
def calculate(wind):
  end = wind[-2:]
  if wind[-2:] not in "NESWN": end = end[::-1]
  low = "NESWN".index(end) * 90
  high = low + 90
  curr = low + 45
  for c in wind[-3::-1]:
    if end.index(c) == 0: high = curr
    else: low = curr
    curr = (low + high) / 2
  return curr

X, Y = sorted("NESW".index(d[0]) * 90 if len(d) == 1 else calculate(d)
              for d in input().split())
print(min(Y - X, 360 + X - Y))
```

Solving Sorting and Search Problems

Natural Order Sorting Algorithms

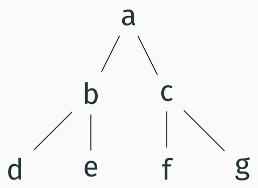
- Sorting algorithms sort items in a natural order
- Use built-in sort as much of possible
- C++17: `std::sort()`
- Python: `list.sort()` or `sorted()`
- Java/Kotlin: `Arrays.sort()` or `Collections.sort()`
- C++ uses IntroSort and Python/Java/Kotlin use TimSort, both $\mathcal{O}(n \log n)$
- For special case sorting, write a specific comparator or key function (Python)

Binary Search

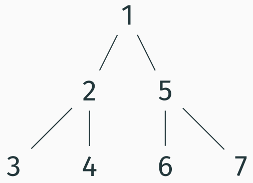
- If the input is sorted or monotonically increasing you can use binary search
- Start in the middle and half the search space based on if the value is to high or to low
- Results in an $\mathcal{O}(\log n)$ algorithm
- C++17: `std::binary_search()`
- Python: `bisect.bisect()`
- Java/Kotlin: `Arrays.binarySearch()` or `Collections.binarySearch()`
- Implementing your own binary search is error prone!

Depth first search (DFS)

- Greedy algorithm to search a graph or tree
- Visit every node when it is discovered
- Usually implemented by using a stack data structure
- Use an array to keep track of discovered nodes
- Example application is solving a maze
- **Note**: Using a boolean array is faster then using a set



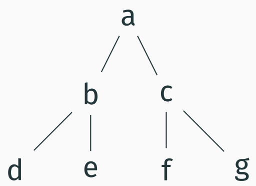
DFS Traverses the path



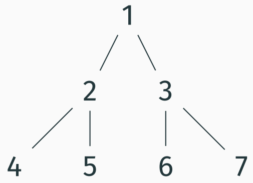
resulting order: a b d e c f g

Breath first search (BFS)

- Similar to DFS, but visits based on discovery order
- Usually implemented using a queue
- Example application is flood-filling to find a closest node



DFS Traverses the path



resulting order: a b c d e f g

Search using a heap

- Search a space, selecting the next position by a condition
- For example: the closest discovered node in Dijkstra’s Shortest Path Algorithm
- C++17: `std::priority_queue`
- Python: `heapq.heapify()`
- Java/Kotlin: `java.util.PriorityQueue()`

Problem: Bubble-bubble Sort

- Source BAPC Preliminaries 2022
- Problem name: Bubble-bubble Sort
- Time limit: 2s

Problem: Bubble-bubble Sort (1)

Bubbles! As a fanatical supporter of the Bubbles Are Perfect Creatures movement, you have accumulated a large collection of bubbles in all colours and sizes. Being a long time member, your bubbles are among the best in the world, and now is the time to show this. Tomorrow, the yearly Bubble Exposition will be held, and your goal is to win the Bubble Prize and become the Bubble Champion!

However, this is not an easy competition. In order to win, you do not only need the most beautiful bubbles, you also need the best-looking placement of bubbles. You have decided to order the bubbles by bubbliness: less bubblier bubbles to the left, more bubblier bubbles to the right. However, it is hard to compare all the bubbles in your collection at once. In fact, you can only compare up to k bubbles by eye before losing track of all the bubbles. Since your collection consists of more than k bubbles, you need a fancier sorting algorithm.

Problem: Bubble-bubble Sort (2)

Your first thought is to use the best sorting algorithm for bubbly purposes, namely Bubble Sort. However, this is the most prestigious bubble competition, so you decide to do better: Bubble-bubble Sort. It works as follows.

Initially, your bubbles are placed in an arbitrary order. Every hour, you do the following: you look at the first k bubbles and place them in the optimal order. Then, you look at bubbles 2 to $k+1$ and place those in the correct order. Then, you look at bubbles 3 to $k+2$, and so on, until you have placed the last k bubbles in the correct order. You then admire how the bubble collection looks so far until the next hour begins and you start at the first bubbles again.

Is this algorithm fast enough to place all your bubbles, or do you need to go further and invent a Bubble-bubble-bubble Sort algorithm? To be precise, after how many hours are the bubbles in the optimal positions?

Problem: Bubble-bubble Sort: Input and Output

Input The input consists of:

- One line with two integers n and k ($2 \leq k < n \leq 2500$), the number of bubbles and the number of bubbles you can sort at once.
- One line with n integers a ($0 \leq a \leq 10^9$), the bubbliness of each bubble in the initial placement of your bubble collection.

Output

Output the number of hours needed to sort your bubble collection.

Problem: Bubble-bubble Sort: Samples

Sample Input 1	Sample Output 1
5 2 3 4 1 5 2	3

Sample Input 2	Sample Output 2
8 3 60 8 27 7 68 41 53 44	2

Sample Input 3	Sample Output 3
6 3 3 2 4 2 3 1	3

Problem: Bubble-bubble Sort: Example 2: After 1 hour

Window *k* is 3.

Hour								
0	60	8	27	7	68	41	53	44
step 1	8	27	60					
step 2		7	27	60				
step 3			27	60	68			
step 4				41	60	68		
step 5					53	60	68	
step 6						44	60	68
hour 1	8	7	**27**	**41**	53	44	**60**	**68**

List is not sorted yet, we need an other hour.

Problem: Bubble-bubble Sort: Example 2: After 2 hours

Window k is 3.

Hour								
hour 1	8	7	27	41	53	44	60	68
step 1	7	8	27					
step 2		8	27	41				
step 3			27	41	53			
step 4				41	44	53		
step 5					44	53	60	
step 6						53	60	68
hour 2	7	8	27	41	44	53	60	68

List is sorted, so solution is 2.

Problem: Bubble-bubble Sort: Naive Solution

- Observation: $n \leq 2500$, so the target is an $\mathcal{O}(n^2)$ solution

Problem: Bubble-bubble Sort: Naive Solution

- Observation: $n \leq 2500$, so the target is an $\mathcal{O}(n^2)$ solution
- Worst case is described in example 3:

hour 0	3	2	4	2	3	**1**
hour 1	2	2	3	**1**	3	4
hour 2	2	**1**	2	3	3	4
hour 3	**1**	2	2	3	3	4

Problem: Bubble-bubble Sort: Naive Solution

- Observation: $n \leq 2500$, so the target is an $\mathcal{O}(n^2)$ solution
- Worst case is described in example 3:

hour 0	3	2	4	2	3	**1**
hour 1	2	2	3	**1**	3	4
hour 2	2	**1**	2	3	3	4
hour 3	**1**	2	2	3	3	4

- The lowest number is at the right most position and moves $k-1$ steps to the left, while total steps needed are $n-1$.
- So the worst case number of hours is $\left\lceil \frac{n-1}{k-1} \right\rceil$

Problem: Bubble-bubble Sort: Naive Solution

- Observation: $n \leq 2500$, so the target is an $\mathcal{O}(n^2)$ solution
- Worst case is described in example 3:

hour 0	3	2	4	2	3	**1**
hour 1	2	2	3	**1**	3	4
hour 2	2	**1**	2	3	3	4
hour 3	**1**	2	2	3	3	4

- The lowest number is at the right most position and moves $k-1$ steps to the left, while total steps needed are $n-1$.
- So the worst case number of hours is $\left\lceil \frac{n-1}{k-1} \right\rceil$
- The described algorithm is $\mathcal{O}\left(\frac{n-1}{k-1}n\right) \simeq \mathcal{O}(n^2)$

Problem: Bubble-bubble Sort: Smart Solution

- First sort the list to determine the new position of each item a_i.
- Warning: Be mindful of duplicates, link the indexes

Problem: Bubble-bubble Sort: Smart Solution

- First sort the list to determine the new position of each item a_i.
- Warning: Be mindful of duplicates, link the indexes
- Then print out with i is the index of the input

$$\max D_i = \begin{cases} 1 & \text{if } i < a_i \text{ (The item moves right)} \\ 0 & \text{if } i = a_i \text{ (The item is in the correct position)} \\ \left\lceil \frac{i-a_i-1}{k-1} \right\rceil & \text{(the item moves left)} \end{cases}$$

Problem: Bubble-bubble Sort: Smart Solution

- First sort the list to determine the new position of each item a_i.
- Warning: Be mindful of duplicates, link the indexes
- Then print out with i is the index of the input

$$\max D_i = \begin{cases} 1 & \text{if } i < a_i \text{ (The item moves right)} \\ 0 & \text{if } i = a_i \text{ (The item is in the correct position)} \\ \left\lceil \frac{i-a_i-1}{k-1} \right\rceil & \text{(the item moves left)} \end{cases}$$

- The complexity is $\mathcal{O}(n \log n)$ (sorting) + $\mathcal{O}(n)$ (finding the max D_i)

Problem: Bubble-bubble Sort: Solution

```python
n,k = [int(i) for i in input().split()]
a = sorted([(int(c),i) for i,c in enumerate(input().split())])
best = 0
for j,(c,i) in enumerate(a):
    best = max(best, i-j)
print(((best-1) // (k-1)) + 1)
```

Problem: Imperfect Imperial Units

- Source BAPC 2022
- Problem name: Imperfect Imperial Units
- Time limit: 4s

Problem: Imperfect Imperial Units

You are writing a paper for the Beta Astronomy Physics Conference about your recent discovery on grey holes. One of your collaborators has performed a huge number of measurements,which you would like to analyse in order to draw some conclusions. The only problem is: the data is measured in a wide variety of units, and to your disgust, they appear to use a mix of the imperial and metric systems. To simplify your analysis, you need to convert all these measurements into a different unit.

Problem: Imperfect Imperial Units: Input

The input consists of:

- One line with two integers n and q ($1 \leq n \leq 100$, $1 \leq q \leq 10,000$), the number of unit conversion equations and the number of queries to answer.
- n lines, each defining a unit conversion in the format "`1 <unit> = <value> <unit>`".
- q lines, each with a query in the format "`<value> <unit> to <unit>`".

In these formats, "`<value>`" is a floating-point number v ($0.001 \leq v \leq 1000$, with at most 9 digits after the decimal point) and "`<unit>`" is a string of at most 20 English lowercase letters (`a-z`). A unit in a query is guaranteed to be defined in at least one unit conversion equation. Every unit can be converted into every other unit in *at most* one way.

Problem: Imperfect Imperial Units: Output

For every query, output the value of the requested unit, or “`impossible`” if the query cannot be answered.

Your answers should have a *relative* error of at most 10^{-6}.

Problem: Imperfect Imperial Units: Samples (1)

Input Sample 1	Output Sample 1
4 3	
1 foot = 12 inch	
1 yard = 3 foot	
1 meter = 100 centimeter	
1 centimeter = 10 millimeter	
750 millimeter to meter	0.75
42 yard to inch	1512
10 meter to foot	impossible

Problem: Imperfect Imperial Units: Samples (2)

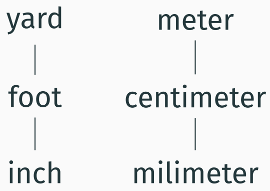
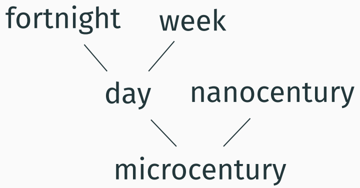
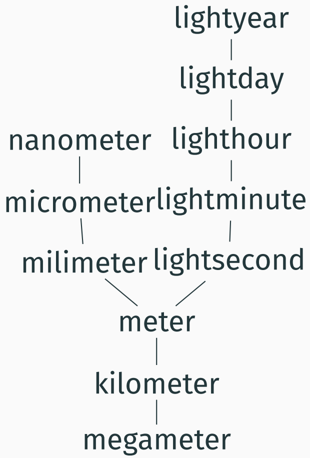
Input Sample 2	Output Sample 2
4 3	
1 fortnight = 14 day	
1 microcentury = 0.036525 day	
1 microcentury = 1000 nanocentury	
1 week = 7 day	
22.2 fortnight to nanocentury	8509240.2464065708427
2.5 nanocentury to week	1.3044642857142857142e-05
3.14 day to fortnight	0.22428571428571428572

Problem: Imperfect Imperial Units: Samples (3)

Input Sample 3	Output Sample 3
10 2	
1 micrometer = 1000 nanometer	
1 millimeter = 1000 micrometer	
1 meter = 1000 millimeter	
1 kilometer = 1000 meter	
1 megameter = 1000 kilometer	
1 lightsecond = 299.792458 meter	
1 lightminute = 60 lightsecond	
1 lighthour = 60 lightminute	
1 lightday = 24 lighthour	
1 lightyear = 365.25 lightday	
42 nanometer to lightyear	4.439403502903384947e-18
42 lightyear to nanometer	3.9735067984839359997e+20

Problem: Imperfect Imperial Units: Solution

Create a graph from for the *n* rules, which has max 100 nodes

Sample 1	Sample 2	Sample 3
		 lightyear lightday lighthour lightminute lightsecond nanometer micrometer milimeter meter kilometer megameter

Problem: Imperfect Imperial Units: Solution

- For every query use use a BFS/DFS to search for a path from the first unit to the requested node
- The problem specification ensures at most 1 path can exist, if no path is found, print out “impossible”
- This will give a $\mathcal{O}(n \cdot q)$ complexity of the algorithm
- This is fast enough.

Problem: Imperfect Imperial Units: Solution

```
(n, q), units = map(int, input().split()), dict()

for _ in range(n):
    a, b = input()[2:].split(" = ")
    x, b = b.split()
    x = float(x)
    if a not in units: units[a] = dict()
    if b not in units: units[b] = dict()
    units[a][b], units[b][a] = x, 1 / x

for _ in range(q):
    a, b = input().split(" to ")
    x, a = a.split()
    seen, todo = {a: float(x)}, [a]
    while todo:
        curr = todo.pop()
        if curr == b:
            print(seen[b])
            break
        for k, v in units[curr].items():
            if k not in seen:
                seen[k] = seen[curr] * v
                todo.append(k)
    else:
        print("impossible")
```

Imperfect Imperial Units: Improved Solution

- Our solution performs duplicate calculations, since the number of queries is significantly larger then number of units.

Imperfect Imperial Units: Improved Solution

- Our solution performs duplicate calculations, since the number of queries is significantly larger then number of units.
- You can cache found (sub)conversions for future queries

Imperfect Imperial Units: Improved Solution

- Our solution performs duplicate calculations, since the number of queries is significantly larger then number of units.
- You can cache found (sub)conversions for future queries
- Or we can add some preprocessioning, calculating the conversion unit for every possible unit reachable
- There are $\frac{n(n-1)}{2}$ combinations to calculate first ($\mathcal{O}(n^2)$)
- Then, for every query, we can just look up the factor and calculate the difference

Imperfect Imperial Units: Improved Solution

- Our solution performs duplicate calculations, since the number of queries is significantly larger then number of units.
- You can cache found (sub)conversions for future queries
- Or we can add some preprocessioning, calculating the conversion unit for every possible unit reachable
- There are $\frac{n(n-1)}{2}$ combinations to calculate first ($\mathcal{O}(n^2)$)
- Then, for every query, we can just look up the factor and calculate the difference
- Resulting in a faster $\mathcal{O}(n^2 + q)$

Imperfect Imperial Units: Improved Solution

- Our solution performs duplicate calculations, since the number of queries is significantly larger then number of units.
- You can cache found (sub)conversions for future queries
- Or we can add some preprocessioning, calculating the conversion unit for every possible unit reachable
- There are $\frac{n(n-1)}{2}$ combinations to calculate first ($\mathcal{O}(n^2)$)
- Then, for every query, we can just look up the factor and calculate the difference
- Resulting in a faster $\mathcal{O}(n^2 + q)$
- **Pitfall**: The result will fit in a double, but be careful with printing precision in C++. Avoid `std::cout << std::fixed` here!

Problem: Imperfect Imperial Units: Improved Solution

```python
(n, q), units, conversions = map(int, input().split()), dict(), dict()

for _ in range(n):
    _, a, _, x, b = (t(s) for t, s in zip((str, str, str, float, str), input().split()))
    units = {a: dict(), b: dict(), **units}
    units[a][b], units[b][a] = x, 1 / x

for a in units:
    seen, todo = {a: 1.0}, [a]
    while todo:
        curr = todo.pop()
        for k, v in units[curr].items():
            if k not in seen: seen[k], _ = seen[curr] * v, todo.append(k)
    conversions[a] = seen

for x, a, _, b in ((t(s) for t, s in zip((float, str, str, str), input().split())) for _ in range(q)):
    print(float(x) * conversions[a][b] if a in conversions and b in conversions[a] else "impossible")
```

Conclusion

Guest speaker

Next Session

Next session is on Monday the 18th of September.

`https://domjudge.ewi.tudelft.nl/`