DAPC 2023 Training Sessions
Session 3

Verwoerd

September 18, 2023

Session 3

- Team Reference Document
- Solutions to Sorting and Search Problems
- Solving interactive problems
- Solving Dynamic Programming Problems

Slides are available on `https://chipcie.wisv.ch/` in the training news post.

Team Reference Document

Team Reference Document

- During the DAPC you may bring any analogue reference material you want
- Starting from the BAPC, you may only bring limited reference material (= same rules as World Finals)
- This reference is called the Team Reference Document (TRD)
- Sometimes the old term Team Contest Reference (TCR) is used

Team Reference Document: Rules

Each contestant may bring an (identical) copy of a Team Reference Document. This document may contain up to 25 pages of reference materials, single-sided, letter or A4 size, with pages numbered in the upper right-hand corner and your university name printed in the upper left-hand corner. Text and illustrations must be readable by a person with correctable eyesight without magnification from a distance of ½ meter. It may include handwritten comments and corrections on the fronts of pages only. The document should be in some type of notebook or folder with the name of your institution on the front.

TRD: Example

Formulas for Geometric Shapes

$$\begin{aligned}
\text{oppervlakte cirkel} &: \pi r^2 \\
\text{omtrek cirkel} &: \pi d \\
\text{oppervlakte ellips} &: \pi ab \\
\text{oppervlakte kegel} &: \pi r^2 + \pi r\sqrt{r^2 h^2} \\
\text{inhoud kegel} &: \frac{1}{3}\pi r^2 h \\
\text{oppervlakte bol} &: 4\pi r^2 \\
\text{inhoud bol} &: \frac{4}{3}\pi r^3 \\
\text{oppervlakte cillinder} &: 2\pi rh + 2\pi r^2 \\
\text{inhoud cilinder} &: \pi r^2 h
\end{aligned}$$

More Formulas

$$\text{least common multiple} : \mathrm{lcm}(m, n) = \frac{|m \cdot n|}{\mathrm{gcd(m,\ n)}}$$

$$\text{Catalan number} : C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n} \frac{n+k}{k}$$

$$\text{Catalan numbers} : C = \{1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796\}$$

$$\text{Triangle numbers} : T = \{1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, 120, 136\}$$

$$\text{Triangle numbers} : T_n = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \binom{n+1}{2}$$

Fibonacci Numbers

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584$$

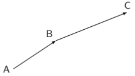
- When we take a pairs of large consecutive Fibonacci numbers, we can approximate the golden ratio by dividing them.
- The sum of any ten consecutive Fibonacci numbers is divisible by 11.
- Two consecutive Fibonacci numbers are co-prime.
- The Fibonacci numbers in the composite-number (i.e. non-prime) positions are also composite numbers.

Computational Geometry

Cross product

$$a \times b = \begin{bmatrix} a_x \\ a_b \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$$

Links of rechts ombuigen



$$\overrightarrow{AB} = \begin{bmatrix} p \\ q \end{bmatrix} \qquad \overrightarrow{n} = \begin{bmatrix} q \\ -p \end{bmatrix}$$

$$\overrightarrow{n} \cdot \overrightarrow{BC} < 0 \Rightarrow \text{linksaf}$$

$$\overrightarrow{n} \cdot \overrightarrow{BC} > 0 \Rightarrow \text{rechtsaf}$$

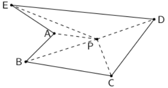
Punt in concaaf/convex polygon test

Tel het aantal doorsnijdingen van polygon met lijn P naar oneindig. Als het aantal doorsnijdingen oneven is, dan $P \in ABCDE$.

$$\alpha = \angle APB + \ldots + \angle DPE + \angle EPA$$

$$\alpha = 0 \Rightarrow P \notin ABCDE$$

$$\alpha = 2\pi \Rightarrow P \in ABCDE$$



Centroid of polygon

The centroid or geometric center of a plane figure is the arithmetic mean ("average") position of all the points in the shape. Informally, it is the point at which an infinitesimally thin cutout of the shape could be perfectly balanced on the tip of a pin.

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$A = \frac{1}{2}\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)$$

TRD: Example

TU Delft

Math

```
int ceildiv(int a, int b) {
    return (a + b - 1) / b;
}
```

Euler Totient Function (aantal coprimes $\leq n$)

```
public int totient (int n) {
    int ans = n;
    for (int i = 2 ; i * i <= n ; i++) {
        if (n % i == 0) ans -= ans / i;
        while (n % i == 0) n /= i;
    }
    if (n > 1) ans -= ans / n;
    return ans;
}
```

Discrete logaritme $a^x \equiv b \pmod{m}$, retourneert de kleinste x die hieraan voldoet anders -1 (maakt gebruik van egcd).

```
public long modLog(long a, long b, long m) {
    if (b % exgcd(a, m)[2] != 0) return -1;
    if (m == 0) return 0;
    long n = (long)sqrt(m) + 1;
    Map<Long, Long> map = new HashMap<Long, Long>();
    long an = 1;
    for (long j = 0; j < n; j++) {
        if (!map.containsKey(an)) map.put(an, j);
        an = an * a % m;
    }
    long ain = 1, res = Long.MAX_VALUE;
    for (long i = 0; i < n; i++) {
        long[] is = congruence(ain, b, m);
        for (long aj = is[0]; aj < m; aj += is[1]) {
            if (map.containsKey(aj)) {
                long j = map.get(aj);
                res = min(res, i * n + j);
            }
        }
        if (res < Long.MAX_VALUE) return res;
        ain = ain * an % m;
    }
    return -1;
}
```

Rekent $(a^b) \mod c$ uit:

```
int modpow(int a, int b, int c){
  long x=1,y=a; // long is taken to avoid overflow of intermediate results
  while(b > 0){
    if(b%2 == 1){
      x=(x*y)%c;
    }
    y = (y*y)%c; // squaring the base
    b /= 2;
  }
  return x%c;
}
```

Rekent $(a \cdot b) \mod c$ uit:

```
long mulmod(long a, long b, long c){
    long x = 0, y = a % c;
    while (b > 0) {
        if(b % 2 == 1){
            x = (x + y) % c;
        }
        y = (y * 2) % c;
        b /= 2;
    }
    return x % c;
}
```

Aantal mogelijke manieren om een nummer te splitsen in positieve getallen. Bijvoorbeeld: $f(4) = \{4, 3+1, 2+2, 2+1+1, 1+1+1+1\}$.

```
int partition(int n) {
    int[] dp = new int[n + 1];
    dp[0] = 1;
    for (int i = 1; i <= n; i++) {
        for (int j = 1, r = 1; i - (3*j * j - j) / 2 >= 0; j++, r *= -1) {
            dp[i] += dp[i - (3 * j * j - j) / 2] * r;
            if (i - (3 * j * j + j) / 2 >= 0) {
                dp[i] += dp[i - (3 * j * j + j) / 2] * r;
            }
        }
    }
    return dp[n];
}
```

TRD: Example

Contest (1)

template.cpp — 14 lines

```
#include <bits/stdc++.h>
using namespace std;

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

int main() {
  cin.tie(0)->sync_with_stdio(0);
  cin.exceptions(cin.failbit);
}
```

.bashrc — 3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++14 \
  -fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps = <>
```

.vimrc — 6 lines

```
set cin aw ai is ts=4 sw=4 tm=50 nu noeb bg=dark ru cul
sy on   |   im jk <esc>   |   im kj <esc>   |   no ; :
" Select region and then type :Hash to hash your selection.
" Useful for verifying that there aren't mistypes.
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
 \| md5sum \| cut -c-6
```

hash.sh — 3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]'| md5sum |cut -c-6
```

troubleshoot.txt — 52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Mathematics (2)

2.1 Equations

$$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The extremum is given by $x = -b/2a$.

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \dfrac{ed - bf}{ad - bc} \\ y = \dfrac{af - ec}{ad - bc} \end{matrix}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

2.2 Recurrences

If $a_n = c_1 a_{n-1} + \cdots + c_k a_{n-k}$, and $r_1, \ldots, r_k$ are distinct roots of $x^k + c_1 x^{k-1} + \cdots + c_k$, there are $d_1, \ldots, d_k$ s.t.

$$a_n = d_1 r_1^n + \cdots + d_k r_k^n.$$

Non-distinct roots r become polynomial factors, e.g. $a_n = (d_1 n + d_2) r^n$.

2.3 Trigonometry

$$\sin(v + w) = \sin v \cos w + \cos v \sin w$$
$$\cos(v + w) = \cos v \cos w - \sin v \sin w$$

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$
$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$
$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w.

$$a \cos x + b \sin x = r \cos(x - \phi)$$
$$a \sin x + b \cos x = r \sin(x + \phi)$$

where $r = \sqrt{a^2 + b^2}, \phi = \operatorname{atan2}(b, a)$.

2.4 Geometry

2.4.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \dfrac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \dfrac{abc}{4A}$

TRD: Example

6.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + \ldots + n_1 p + n_0$ and $m = m_k p^k + \ldots + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod{p}$.

6.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1 + \cdots + k_n}{k_1, k_2, \ldots, k_n} = \frac{(\sum k_i)!}{k_1! k_2! \ldots k_n!}$.

a0a312, 6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t - 1}$ (FFT-able). $B[0, \ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty} \frac{B_k}{k!} f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n, k) = c(n-1, k-1) + (n-1)c(n-1, k), \quad c(0, 0) = 1$$

$$\textstyle\sum_{k=0}^{n} c(n, k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8, k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n, 2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

6.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

$$E(n, k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$$

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k)$$

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$$

6.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

6.3.6 Labeled unrooted trees

\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)! \cdots (d_n-1)!)$

6.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1, \ C_{n+1} = \frac{2(2n+1)}{n+2} C_n, \ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Graph (7)

7.1 Fundamentals

BellmanFord.h

Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get dist = inf; nodes reachable through negative-weight cycles get dist = -inf. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

830a8f, 23 lines

```
const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
  nodes[s].dist = 0;
  sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

  int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
  rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
      dest.prev = ed.a;
      dest.dist = (i < lim-1 ? d : -inf);
    }
  }
  rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
      nodes[e.b].dist = -inf;
  }
}
```

FloydWarshall.h

Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m, where $m[i][j] =$ inf if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j, inf if no path, or -inf if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

531245, 12 lines

```
const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
  int n = sz(m);
  rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
  rep(k,0,n) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) {
      auto newDist = max(m[i][k] + m[k][j], -inf);
      m[i][j] = min(m[i][j], newDist);
    }
  rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
    if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

TopoSort.h

Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$

66a137, 14 lines

```
vi topoSort(const vector<vi>& gr) {
  vi indeg(sz(gr)), ret;
  for (auto& li : gr) for (int x : li) indeg[x]++;
  queue<int> q; // use priority_queue for lexic. largest ans.
  rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
  while (!q.empty()) {
    int i = q.front(); // top() for priority queue
    ret.push_back(i);
    q.pop();
    for (int x : gr[i])
      if (--indeg[x] == 0) q.push(x);
  }
  return ret;
}
```

Potential subjects in a TRD

1. Mathematics
   - Formulas and Theories
   - Trigonometry
2. Data Structures
   - Segement Tree, Treap, RMQ
   - HashMap, PriorityQueue
3. Numerical Methods
   - Simplex, Integration
   - Linear Problem-solving
4. Number Theory
   - Primality, Divisability
5. Combinatorial
   - Permutations, Partitions
6. Graph
   - Search algorithms
   - Flow algorithms
   - Spanning Tree, Connected Components
7. Geometry
   - Line intersection, length
   - Triangles and Circles
   - Polygons
8. Strings
9. Templates
10. Tests and reminders

Tips on TRD

- Only put stuff in that you know how/when to use
- Ensure that the code is correct and complete
- Add short description, complexity, and hash
- Evaluate document after each contest for improvements
- Several templates available at `https://chipcie.wisv.ch/resources`

Solutions to the sorting and search problems

Abbreviated Aliases

- Source BAPC Preliminaries 2022
- Time limit: 2s
- For every username calculate the size of the shortest unique prefix.

Abbreviated Aliases

- Observation: $n \cdot l \leq 10^7$, so we are aiming for $\mathcal{O}(n \log n)$

Abbreviated Aliases

- Observation: $n \cdot l \leq 10^7$, so we are aiming for $\mathcal{O}(n \log n)$
- Comparing every username with all other usernames is $\mathcal{O}(n^2)$, which is too slow

Abbreviated Aliases

- Observation: $n \cdot l \leq 10^7$, so we are aiming for $\mathcal{O}(n \log n)$
- Comparing every username with all other usernames is $\mathcal{O}(n^2)$, which is too slow
- We only need to compare the two usernames where the prefix is most similar
  `james` is closest to `jacob` and `janos`, there is no other username that will increase the prefix

Abbreviated Aliases

- Observation: $n \cdot l \leq 10^7$, so we are aiming for $\mathcal{O}(n \log n)$
- Comparing every username with all other usernames is $\mathcal{O}(n^2)$, which is too slow
- We only need to compare the two usernames where the prefix is most similar
  `james` is closest to `jacob` and `janos`, there is no other username that will increase the prefix
- If we sort the list, we only have to compare with the username before and after

Abbreviated Aliases

- Observation: $n \cdot l \leq 10^7$, so we are aiming for $\mathcal{O}(n \log n)$
- Comparing every username with all other usernames is $\mathcal{O}(n^2)$, which is too slow
- We only need to compare the two usernames where the prefix is most similar
  `james` is closest to `jacob` and `janos`, there is no other username that will increase the prefix
- If we sort the list, we only have to compare with the username before and after
- Alternatively, build a compressed Trie, and for each leaf, count the distance to the root

Abbreviated Aliases

```
n,l = [int(i) for i in input().split()]
a = sorted([input() for i in range(n)])
t = 1
p = 0
for i in range(1,n):
  for j in range(l):
    if a[i-1][j] != a[i][j]:
      t += j+1 + max(j-p,0)
      p = j
      break
print(t)
```

Dimensional Debugging

- Source BAPC Preliminaries 2022
- Time limit: 2s
- Given n algorithms that only work when their input φ is small enough ($\varphi \leq H$), can you verify the correctness on sufficient large inputs ($\varphi \geq L$).

Dimensional Debugging

- Observation: $n \leq 10^3$, so we are aiming for an $\mathcal{O}(n^2)$ algorithm.

Dimensional Debugging

- Observation: $n \leq 10^3$, so we are aiming for an $\mathcal{O}(n^2)$ algorithm.
- We can verify all algorithms with $L = \varphi_0$

Dimensional Debugging

- Observation: $n \leq 10^3$, so we are aiming for an $\mathcal{O}(n^2)$ algorithm.
- We can verify all algorithms with $L = \varphi_0$
- Add those algorithms to verified algorithms, then find any unverified where $H_j \leq L_i$

Dimensional Debugging

- Observation: $n \leq 10^3$, so we are aiming for an $\mathcal{O}(n^2)$ algorithm.
- We can verify all algorithms with $L = \varphi_0$
- Add those algorithms to verified algorithms, then find any unverified where $H_j \leq L_i$
- In this way you can create a graph between the different algorithms.

Dimensional Debugging

- Observation: $n \leq 10^3$, so we are aiming for an $\mathcal{O}(n^2)$ algorithm.
- We can verify all algorithms with $L = \varphi_0$
- Add those algorithms to verified algorithms, then find any unverified where $H_j \leq L_i$
- In this way you can create a graph between the different algorithms.
- Use a flood fill by BFS/DFS to count the number of algorithms you can reach.

Dimensional Debugging

- Observation: $n \leq 10^3$, so we are aiming for an $\mathcal{O}(n^2)$ algorithm.
- We can verify all algorithms with $L = \varphi_0$
- Add those algorithms to verified algorithms, then find any unverified where $H_j \leq L_i$
- In this way you can create a graph between the different algorithms.
- Use a flood fill by BFS/DFS to count the number of algorithms you can reach.
- This results in an $\mathcal{O}(n^2)$ algorithm.

Dimensional Debugging

```
n, k = map(int, input().split())
algs = {tuple(zip(*(map(int, input().split()) for _ in ".."))) for _ in range(n)}
stack = [[(0, 0) for _ in range(k)]]
while stack:
  base = stack.pop()
  add = {alg for alg in algs if all(l <= b for (a, b), (l, h) in zip(base, alg))}
  stack.extend(add)
  algs = algs.difference(add)
print(n - len(algs))
```

Extended Braille

- Source BAPC Preliminaries 2022
- Time limit: 8s
- Given n braille characters by their points, determine how many of them are distinct up to translation.

Extended Braille

- Observation 1: time limit of 8s is due to high input size

Extended Braille

- Observation 1: time limit of 8s is due to high input size
- Observation 2: at most 10^6 dots, so we are looking for $\mathcal{O}(n \log n)$

Extended Braille

- Observation 1: time limit of 8s is due to high input size
- Observation 2: at most 10^6 dots, so we are looking for $\mathcal{O}(n \log n)$
- Per Braille character, sort the dots on x then y

Extended Braille

- Observation 1: time limit of 8s is due to high input size
- Observation 2: at most 10^6 dots, so we are looking for $\mathcal{O}(n \log n)$
- Per Braille character, sort the dots on x then y
- Move the first ordered dot to $(0, 0)$ by subtracting the first point coordinate from all the dots
  $\forall_{i=1}^{m} (x_i', y_i') = (x_i - x_1, y_i - y_1)$

Extended Braille

- Observation 1: time limit of 8s is due to high input size
- Observation 2: at most 10^6 dots, so we are looking for $\mathcal{O}(n \log n)$
- Per Braille character, sort the dots on x then y
- Move the first ordered dot to $(0, 0)$ by subtracting the first point coordinate from all the dots
  $\forall_{i=1}^{m} (x_i', y_i') = (x_i - x_1, y_i - y_1)$
- Add the transposed characters to a HashMap or Set and count the unique keys

Extended Braille

- Observation 1: time limit of 8s is due to high input size
- Observation 2: at most 10^6 dots, so we are looking for $\mathcal{O}(n \log n)$
- Per Braille character, sort the dots on x then y
- Move the first ordered dot to $(0, 0)$ by subtracting the first point coordinate from all the dots
  $\forall_{i=1}^{m}(x'_i, y'_i) = (x_i - x_1, y_i - y_1)$
- Add the transposed characters to a HashMap or Set and count the unique keys
- Resulting a $\mathcal{O}(n \log n)$ or amortized $\mathcal{O}(n)$

Extended Braille

```
n, chars = int(input()), set()
for _ in range(n):
  cc = [list(map(int, input().split())) for _ in range(int(input()))]
  min_x, min_y = (min(xs) for xs in zip(*cc))
  chars.add(tuple(sorted([(x - min_x, y - min_y) for x, y in cc])))
print(len(chars))
```

Knitting Pattern

- Source BAPC Preliminaries 2022
- Time limit: 3s
- Given a knitting pattern and amount of wool it costs for letting the wool strand unused, using the wool in a stitch, and for starting or ending the use of wool. Compute the minimal amount of wool required for every colour of wool.

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$
- For gap between colours you have 2 options

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$
- For gap between colours you have 2 options
    1. let the strand continue

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$
- For gap between colours you have 2 options
  1. let the strand continue
  2. stop the strand and start again

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$
- For gap between colours you have 2 options
  1. let the strand continue
  2. stop the strand and start again
- For every colour, we have to calculate for every gap:

$$\min(c_{stop} + c_{start}, gap_{size} \cdot c_{unused})$$

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$
- For gap between colours you have 2 options
  1. let the strand continue
  2. stop the strand and start again
- For every colour, we have to calculate for every gap:

$$\min(c_{stop} + c_{start}, gap_{size} \cdot c_{unused})$$

- Calculate the total cost for each colour and sum it together

Knitting Pattern

- Observation: $|p| \leq 10^6$, so we are aiming for a $\mathcal{O}(|p| \log |p|)$
- For gap between colours you have 2 options
  1. let the strand continue
  2. stop the strand and start again
- For every colour, we have to calculate for every gap:

$$\min(c_{stop} + c_{start}, gap_{size} \cdot c_{unused})$$

- Calculate the total cost for each colour and sum it together
- The complexity is $\mathcal{O}(|w| \cdot |p|)$, or in a single pass over p with creative bookkeeping

Knitting Pattern

```
a, b, c = map(int, input().split())
w = input()
s = input()
for x in w:
  off = 0
  on = 10**9
  for y in s:
    if x == y:
      on = min(on, off + c) + b
      off = on + c
    else:
      on = on + a
  print(off)
```

Kiosk Construction

- Source BAPC 2022
- Time limit: 8s
- Find the optimal kiosk position for a given camping layout.

Kiosk Construction

- The shortest path for every kiosk position to every other plot can be found by using DFS/BFS

Kiosk Construction

- The shortest path for every kiosk position to every other plot can be found by using DFS/BFS
- Then find the kiosk(s) that can reach all plot and minimize the maximum distance

Kiosk Construction

- The shortest path for every kiosk position to every other plot can be found by using DFS/BFS
- Then find the kiosk(s) that can reach all plot and minimize the maximum distance
- Doing n^2 DFS/BFS for every kiosk results in a $\mathcal{O}(n^3)$ solution and receives a TLE

Kiosk Construction

- The shortest path for every kiosk position to every other plot can be found by using DFS/BFS
- Then find the kiosk(s) that can reach all plot and minimize the maximum distance
- Doing n^2 DFS/BFS for every kiosk results in a $\mathcal{O}(n^3)$ solution and receives a TLE
- You an optimize do to some preprocessing, calculate the distance from every plot to every kiosk position, storing intermediate results

Kiosk Construction

- The shortest path for every kiosk position to every other plot can be found by using DFS/BFS
- Then find the kiosk(s) that can reach all plot and minimize the maximum distance
- Doing n^2 DFS/BFS for every kiosk results in a $\mathcal{O}(n^3)$ solution and receives a TLE
- You an optimize do to some preprocessing, calculate the distance from every plot to every kiosk position, storing intermediate results
- This optimization results in a $\mathcal{O}(n^2)$ solution

Kiosk Construction

```python
from collections import deque

h, w = map(int, input().split())
plots = [list(map(int, input().split())) for _ in range(h)]
neighs = [[[min((abs(neigh - dest), abs(neigh - plots[y][x]), xx, yy) for xx, yy, neigh in
                ((xx, yy, plots[yy][xx]) for xx, yy in ((x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1))
                 if 0 <= xx < w and 0 <= yy < h))[2:]
            for x in range(w)] for y in range(h)] for dest in range(1, w * h + 1)]


def find_paths(dest_x, dest_y, dest):
  seen, queue = [[0] * w for _ in range(h)], deque([(dest_x, dest_y, 0)])
  while queue:
    x, y, dist = queue.popleft()
    for xx, yy in ((x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)):
      if 0 <= xx < w and 0 <= yy < h and neighs[dest - 1][yy][xx] == (x, y) and not seen[yy][xx]:
        queue.append((xx, yy, dist + 1))
        seen[yy][xx] = dist + 1
  return seen


paths = [[find_paths(x, y, plots[y][x]) for x in range(w)] for y in range(h)]
best = min((max(paths[dest_y][dest_x][kiosk_y][kiosk_x] or 1e9 for dest_y in range(h) for dest_x in range(w)),
            plots[kiosk_y][kiosk_x]) for kiosk_y in range(h) for kiosk_x in range(w))
print("impossible") if best[0] == 1e9 else print(*best[::-1])
```

Solving interactive problems

What are Interactive Problems?

- Traditional problems give all the input at once, you solve and print all the output at once
- Interactive problems give input, you do work, print output, and you receive new input
- This process continues until you find the final answer
- The problem defines an interaction protocol
- The problem may have an interaction limit
- If an interactive problem may be in the set, an simple interactive problem will be included in the test session

Type of problems for Interactive Problems

- Search in a finite space
- Explore a maze
- Matching games
- Double interaction problem (very, very rare)
  - Program has 2 modes
  - the first mode, input transforms input to output following certain rules
  - The second mode, the output of mode 1 is given and you have tranform it back to the input of mode 1

Common pitfalls for Interactive problems

- Flush the output after every write
  - Only the output, not the input
  - Not flushing the output results in Time Limit Exceeded
- Verdict of a solution is not deterministic, but the following is guaranteed:
  - Wrong Answer means you printed something wrong
  - Runtime Error means you returned an 0 error code
  - If both occur, you will get either
- ICPC style contests don't have "Idleness Limit Exceeded", but a total runtime limit.

Flushing the output

C++ : end your output with `std::endl` or `std::flush`

Python : use the flush parameter, like `print("abc", flush=True)`

Java/Kotlin : use a `java.io.BufferedWriter` and after each write use the `.flush()` method.

Interactive problems testing tool

- Most contests provide a testing tool to test the interaction with a testing tool
- This is usually called `testing_tool.py` in our region
- The header file tells you how to run run the testing tool, for example
  `$ python3 testing_tool.py -f 1.in python3 ./solution.py`
- Pitfall for Java/Kotlin: You should run the testing tool in the directory which contains the compiled class file
- **Wrong**:
  `~/$ python3 testing_tool.py -f 1.in java ./code/ProblemA`
- **Right**:
  `~/code/$ python3 testing_tool.py -f 1.in java ProblemA`

Example Interactive Problem

You are asked to guess a number between 0 and n.

Example Interactive Problem: Interaction

This is an interactive problem. Your submission will be run against an interactor, which reads from the standard output of your submission and writes to the standard input of your submission. This interaction needs to follow a specific protocol:

The interactor first sends one line with an integer n ($3 \leq n \leq 1000$), the upper bound of the guessing game.

You can then send a guess g ($0 \leq g \leq n$).

The interactor will respond with the strings `lower`, `higher`, or `correct`. This represents is if the number to guess is lower, higher, or correct, respectively. After you have guessed the correct number, you should exit the program.

The interactor is not adaptive, i.e. the secret number is fixed during a round. Using more than 12 guesses will result in a wrong answer.

Example Interactive Problem: Example interaction

Sample Input 1	Sample Output 1
1000	
	67
higher	
	967
lower	
	500
correct	

Example Interactive Problem: Python Solution

```python
low = 0
high = int(input())
state = "initial"
while state != "correct":
    mid = (high+low)//2
    _, state = print(mid, flush=True), input().strip()
    if state == "higher":
        low = mid+1
    if state == "lower":
        high = mid - 1
```

Example Interactive Problem: C++ Solution

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
  int high,low;
  string state;
  low = 0;
  cin >> high;
  do {
    int mid;
    mid = (low+high) / 2;
    cout << mid << endl << flush;
    cin >> state;
    if(state.compare("higher") == 0) {
      low = mid +1;
    } else if(state.compare("lower") ==0 ){
      high = mid - 1;
    }
  }while (state.compare("correct") != 0);
  return 0;
}
```

Example Interactive Problem: Java Solution

```java
import java.io.*;

public class InteractiveExample {
  public static void main(String... args) throws IOException {
    var output = new BufferedWriter(new OutputStreamWriter(System.out));
    var input = new BufferedReader(new InputStreamReader(System.in));
    var low = 0;
    var high = Integer.parseInt(input.readLine());
    String state;
    do {
      var mid = (low + high) >> 1;
      output.write(mid + "\n");
      output.flush();
      state = input.readLine();
      if (state.equals("higher")) {
        low = mid + 1;
      } else if (state.equals("lower")) {
        high = mid - 1;
      }
    } while (!state.equals("correct"));
  }
}
```

Example Interactive Problem: Kotlin Solution

```kotlin
fun main() {
  val output = System.out.bufferedWriter()
  var low = 0
  var high = readln().toInt()
  var state: String
  do {
    val mid = (low + high) shr 1
    output.write("$mid\n")
    output.flush()
    state = readln()
    when (state) {
      "higher" -> low = mid + 1
      "lower" -> high = mid - 1
    }
  } while (state != "correct")
}
```

Solving Dynamic Programming Problems

Dynamic Programming

- Dynamic Programming (DP) is a techinque of solving problem by solving an problem by solving it in a recursive simpler sub-problem
- DP requires an overlap to occur, else its considered a Divide and Conquer algorithm

Dynamic Programming: Fibonacci Numbers

- The formula is $\mathcal{F}_i = \mathcal{F}_{i-1} + \mathcal{F}_{i-2}$
- It depends clearly on previous calculations
- It can be solved by $\mathcal{F}_1$ and $\mathcal{F}_2$ which are both 1

```python
def fibonacci(n):
  if n == 1 or n == 2:
    return 1
  else:
    return fibonacci(n - 1) + fibonacci(n - 2)
```

```kotlin
fun fibonacci(i: Long): Long = when(i) {
  1L, 2L -> 1
  else -> fibonacci(i - 1) + fibonacci(i - 2)
}
```

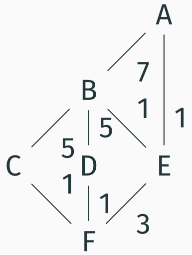
Dynamic Programming: Fibonacci Numbers: Caching

```python
@lru_cache(None)
def fibonacci(n):
  if n == 1 or n == 2:
    return 1
  else:
    return fibonacci(n - 1) + fibonacci(n - 2)
```

```kotlin
val cache = mutableMapOf<Long, Long>()
fun fibonacci(i: Long): Long = when(i) {
  1L, 2L -> 1
  else -> cache.getOrPut(i - 1) { fibonacci(i - 1) }
        + cache.getOrPut(i - 2) { fibonacci(i - 2) }
}
```

Dynamic Programming: Using States

Consider a weighted graph with a adjacency matrix w



$$w = \begin{bmatrix} 0 & 7 & \infty & \infty & 1 & \infty \\ 7 & 0 & 5 & 5 & 1 & \infty \\ \infty & 5 & 0 & \infty & \infty & 1 \\ 1 & 5 & \infty & 0 & \infty & 1 \\ \infty & 1 & \infty & \infty & 0 & 3 \\ \infty & \infty & 1 & 1 & 3 & 0 \end{bmatrix}$$

Calculate a matrix giving the shortest path from and to all nodes (All Pair Shortest Path (APSP)).

Dynamic Programming: Using States (APSP)

- Calculating Dijkstra for every node is very inefficient
- Create the sub-problem calculate APSP with a subset of the connections
- Define k as the number of nodes to use
- Then do the DP by using the following formula:

$$f(i,j,k) = \begin{cases} w(i,j) & \text{if } k = 0 \\ \min(f(i,j,k-1), f(i,k,k-1) + f(k,j,k-1)) \end{cases}$$

- The shortest path between i and j with using only the first k nodes is the min of:
  - the shortest path when using $k-1$
  - the shortest path from i to k plus the shortest path from k to j
- This is Floyd-Warshall's APSP with a complexity $\mathcal{O}(n^3)$

Cookbook Composition

- Source BAPC Preliminaries 2022
- Time limit: 2s
- Given a list of recipes, print the order the recipes by accessibility (lowest $\frac{\textit{beginner time}}{\textit{expert time}}$ first).

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$
- Observation: Steps are in order, dependencies are always declared first

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$
- Observation: Steps are in order, dependencies are always declared first
- The beginner time is trivial to calculate, the sum of the time of all steps

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$
- Observation: Steps are in order, dependencies are always declared first
- The beginner time is trivial to calculate, the sum of the time of all steps
- The expert time can be defined as a DP relation

$$\text{time}_s = \begin{cases} t & \text{if no dependencies are given} \\ t + \max\limits_{steps}^{i=0} time_i & \end{cases}$$

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$
- Observation: Steps are in order, dependencies are always declared first
- The beginner time is trivial to calculate, the sum of the time of all steps
- The expert time can be defined as a DP relation

$$\text{time}_s = \begin{cases} t & \text{if no dependencies are given} \\ t + \max\limits_{steps}^{i=0} time_i & \end{cases}$$

- This can be calculated in linear time by processing the steps one by one, where the expert time is the time of the last step.

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$
- Observation: Steps are in order, dependencies are always declared first
- The beginner time is trivial to calculate, the sum of the time of all steps
- The expert time can be defined as a DP relation

$$\text{time}_s = \begin{cases} t & \text{if no dependencies are given} \\ t + \max\limits_{steps}^{i=0} time_i & \end{cases}$$

- This can be calculated in linear time by processing the steps one by one, where the expert time is the time of the last step.
- Then sort the recipes based on $\frac{\textit{beginner time}}{\textit{expert time}}$ and print out the result

Cookbook Composition

- Observation: $n \cdot s \leq 2.5 \cdot 10^4$, so we are aiming for a $\mathcal{O}(n \cdot s \log n \cdot s)$
- Observation: Steps are in order, dependencies are always declared first
- The beginner time is trivial to calculate, the sum of the time of all steps
- The expert time can be defined as a DP relation

$$\text{time}_s = \begin{cases} t & \text{if no dependencies are given} \\ t + \max\limits_{steps}^{i=0} time_i & \end{cases}$$

- This can be calculated in linear time by processing the steps one by one, where the expert time is the time of the last step.
- Then sort the recipes based on $\frac{\textit{beginner time}}{\textit{expert time}}$ and print out the result
- This results in a $\mathcal{O}(n \cdot s + n \log n)$ solution

Cookbook Composition

```python
n, recipes = int(input()), []

for _ in range(n):
  (recipe, s), end = input().split(), {}
  steps = [(sn, int(t), ds) for sn, t, d, *ds in (input().split() for _ in range(int(s)))]
  for step, t, ds in steps:
    end[step] = t + max((end[d] for d in ds), default=0)
  recipes.append((sum(t for _, t, _ in steps) / max(end.values()), recipe))

print("\n".join(name for _, name in sorted(recipes)))
```

Guest Speaker: Maarten Sijm

Guest speaker

Maarten Sijm

- Head of jury for BAPC since 2022
  - FPC jury member since 2018
  - BAPC jury member since 2020
  - NWERC jury member since 2022
- BAPC/NWERC particpant (best result: 24th)
  - 2016: “Tie Limit Exceeded”
  - 2017: “class RubberDuck extends Throwable {}”
  - 2018: “$\Omega($🦆$^n)$”
- BSc+MSc Computer Science @ TU Delft
- Second-oldest member of CHipCie

Recipe for a Contest

- About a dozen jury members
- Tens of problem ideas
- A few months of time
  - Meeting every two weeks

Problem Selection

- Label problems submitted to Call for Problems
  - How much we like the problem
  - Difficulty rating
  - Categories (math, geometry, graph, ...)
- Select problems that we like best,
  with spread in difficulty and categories

Problem Naming

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**
2	**Function optimising**	Algorithm ...				Efficiency Enhanc	Function Optimizing						
3	**List of Cubes**		Brewing ...	Chemistry ...	Dangerous Chemi	Explosions?		Gas usage					**Liquid Mixing** Liquid Processing
4	**k-Bubble Sort**		**Bubblebubblesort**							Interval Sort		k-Bubble Sort	
5	**Cookbook Composition**	Accessible Cookb	Book of Recipes Beginner Cookbo	**Cookbook Composition**		Expert Cookbook							
6	**Knitting Patterns**							Granny's Knitting				**Knitting Patterns**	
7	**Shortest Unique Prefix**	**Abbreviated Aliases**		Cutting Short									
8	**Showerhead**				Drilling Holes								
9	**Extended Braille**		Braille ...			**Extended Braille**							
10	**Primel (interactive)**						Factoring is Futile	Great Guessing Game					
11	**Slow Memory**	Array Adjustment	Book ...	Copying ...						**Inked Inscriptions**			
12	**Testing**			Comparing Algorit Correctness ...	Dimensional Data								
13	**Moving blocks**	Arranging Boxes	Box Arrangement	Cargo Shipping					Heavy Boxes Heavy lifting				

Problem naming sheet of DAPC 2022

Problem Naming

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		A	B	C	D	E	F	G	H	I	J	K	L
2	Shortest Unique Prefix	**Abbreviated Aliases**		Cutting Short									
3	k-Bubble Sort		**Bubble Bubble Sort**							Interval Sort		k-Bubble Sort	
4	Cookbook Composition	Accessible Cookb	Book of Recipes Beginner Cookboo	**Cookbook Composition**		Expert Cookbook							
5	Testing			Comparing Algorit Correctness ...	**Dimensional Debugging**								
6	Extended Braille		Braille ...		Deaf ...	**Extended Braille**							
7	Function optimising	Algorithm ...			Data manipulation	Efficiency Enhanc Early optimisation	Function Optimising Fascinating function **Fastestest Function**		Horrible code		justifying changes justifying junk justifying optimisation junk code		
8	Primel (interactive)						Factoring is Futile	**Guessing Game**		Integer inquiries			
9	Moving blocks	Arranging Boxes	Box Arrangement	Cargo Shipping					Heavy Boxes Heavy lifting **Heavy Hauling**		Junk... Justified Jetsam		
10	Slow Memory	Array Adjustment	Book ...	Copying ...						**Inked Inscription**	Jumbling Pages		
11	Showerhead				Drilling Holes		Fancy Showerhead		Hole drilling		jets jet propulsion jet stream **Jabbing Jets**		
12	Knitting Patterns			Crochet ...				Granny's Knitting				**Knitting Patterns**	
13	List of Cubes		Brewing ...	Chemistry ...	Dangerous Chemi	Explosions?		Gas usage			justified jetsam		Liquid Mixing Liquid Processing **Lots of Liquid**

Problem naming sheet of DAPC 2022

Problem Implementation

- Problem statement (LaTeX)
- Generating test data (YAML spec + C++/Python scripts)
- Validating input (C++/Python scripts)
- Validating output (C++/Python scripts)
- Submissions in all supported languages
- Solution slides (LaTeX)

- Tooling: `github.com/RagnarGrootKoerkamp/BAPCtools`

Trying to Break Stuff

- Constraints checking
    - Minimal/maximal input
- Fuzzing
    - Generate more random test data from existing scripts
- Write submissions that should be wrong/too slow
- Invite proof readers/solvers

Check That Everything Works

- Continuous Integration
- Upload problems to DOMjudge
    - Local machine
    - Test in Drebbelweg with Maarten (systems Maarten)
- Check time limits on contest hardware

Start of the Contest

- Waiting for the first submissions to come in
- Taking guesses
  - Which problem would be solved first, and after how many minutes?
  - What will be the order of most-solved to least-solved?

During the Contest

- Check incoming submissions
  - Are they correctly marked as AC/TLE/WA/...?
  - Are they using clever solutions that we didn't think of?
- Answer incoming clarification requests
  - The dreadful "*No comment.*" and
    "*Please read the problem statement carefully.*"
- Add common mistakes to solution slides

During the Contest

Submissions

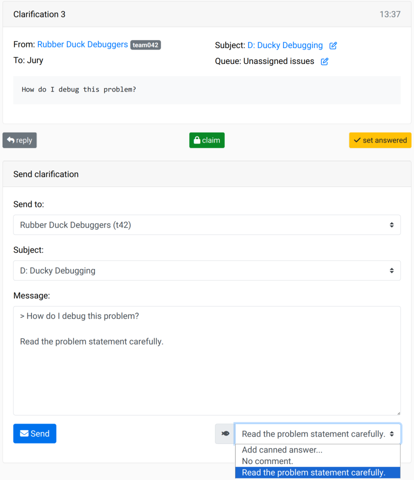
Show: newest | unverified | unjudged | all

Filter

233 submitted | 151 correct | 2 unverified

ID	time	team		problem	lang	result	verified	by	test results
s233	10:10	team055	'); DROP TABLE teams; --	L	KT	CORRECT	no	claim	✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓ ✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓ ✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
s232	09:52	team001	Not The Worst	G	PY3	CORRECT	no	claim	✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓ ✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓✓
s231	09:25	team037	Phoenix	A	JAVA	WRONG-ANSWER	yes	maarten	✓✓✓✓✓✓✓✓✓✓w✕✕✕✕✕✕✕✕✕✕✕✕✕✕✕✕ ✕✕✕✕✕✕✕
s230	09:25	team042	Rubber Duck Debuggers	A	CPP	TIMELIMIT	yes	ducky	✓✓✓✓✓✓✓✓✓✓✓t✕✕✕✕✕✕✕✕✕✕✕✕✕✕✕ ✕✕✕✕✕✕✕

During the Contest



During the Contest

Clarification 3 13:37

From: Rubber Duck Debuggers team042

To: Jury

Subject: D: Ducky Debugging

Queue: Unassigned issues

```
How do I debug this problem?
```

reply claim set answered

Send clarification

Send to:

ALL

Subject:

D: Ducky Debugging

Message:

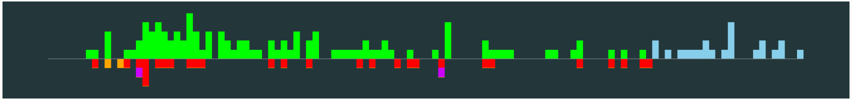
> How do I debug this problem?

Step 1: Talk to the duck.
Step 2: Wait for the duck to reply.
Step 3: ???
Step 4: Profit!

Send Add canned answer...

After the Contest

- Generate solve stats



- Present solutions

And, next year...

- Do it all over again!

Conclusion

Next Session

Next session is on Thursday the 21st of September.
Guest Speaker: Jeroen op de Beek and Leon van der Waal from Segment goes BRRRR about geometry problems.

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