Mergjað mál

- Í þessu dæmi erum við beðin um að ákvarða hvort inntakið innihaldi aðra hvora töluna 69 eða 420.
- Gera má þetta með flestum innbyggðum strengjaleitaföllum.

Lausn - Bash

```
inp=`cat`
if [[ $inp == *"69"* ]] || [[ $inp == *"420"* ]] then
    echo "Mergjad!"
else
    echo "Leim!"
fi
```

COM Bestun

- Við þurfum einfaldlega að reikna hversu margar sekúndur vinnan sparar.
- Reiknum svo út kostnaðinn á verkefninu í sekúndum og berum saman.
- Þurfum að margfalda með 60 fyrir mínútur, 3600 fyrir klukkustundur, o.s.frv.

Lausn - Go, hluti 1

```
package main
import (
        "bufio"
        "os"
        "fmt"
        "strings"
        "strconv"
)
func read_time (s string) int64 {
        n, s, _ := strings.Cut(s, " ")
        var x, _ = strconv.ParseInt(n, 10, 64)
        if(strings.HasPrefix(s, "min")) {
                x *= 60
        } else if(strings.HasPrefix(s, "klu")) {
                x *= 60 * 60
        } else if(strings.HasPrefix(s, "dag")) {
                x *= 60 * 60 * 8
        } else if(strings.HasPrefix(s, "vik")) {
                x *= 60 * 60 * 8 * 5
        } else if(strings.HasPrefix(s, "ar")) {
                x *= 60 * 60 * 8 * 5 * 52
        }
        return x
}
```

Lausn - Go, hluti 2

```
func main() {
        reader := bufio.NewReader(os.Stdin)
        inp, _ := reader.ReadString('\n')
        n, leftover, _ := strings.Cut(inp, " ")
        _, freq, _ := strings.Cut(leftover, " ")
        var t1, _ = strconv.ParseInt(n, 10, 64)
        t1 *= 5
        if (!strings.HasPrefix(freq, "ar")) {
                t1 *= 52
        }
        if (strings.HasPrefix(freq, "dag")) {
                t1 *= 5
        }
        l1, _ := reader.ReadString('\n')
        l2, _ := reader.ReadString('\n')
        t1 *= read_time(l1)
        t2 := read_time(l2)
        if (t2 > t1) {
                fmt.Println("Borgar sig ekki!")
        } else {
                fmt.Println(strconv.FormatInt(t1 - t2, 10))
        }
}
```

Innvolsarinnihaldslýsing

- Inntak er nógu lítið til að hægt sé að einfaldlega skoða sérhvern hlutstreng.
- Með tvöfaldri/þrefaldri for-lykkju má bæta hlutstrengjunum í hakkatöflu.
- Loks má prenta niðurstöðurnar eftir að raða þeim í rétta röð.

Lausn - Ruby

```ruby
dna = STDIN.read.strip
count = Hash.new(0)
for i in 0..dna.length-1
  for j in i..dna.length-1
    count[dna[i..j]] += 1
  end
end
count.each_pair.sort_by {
  |k, v| [-v, k]
}.each {
  |k, v| puts "#{v} #{k}"
}
```

Deildadrottnun

- Það er aðeins ein leið til að gera þetta.
- Auðveldast er að byrja með dreifinguna $1, 2, \ldots, k$ þannig að summan sé $\leq n$.
- Byrja má þá með $k = 1$ og stækka þar til summan yrði $> n$ í næsta skrefi.
- Loks má hækka r öftustu tölurnar ef r er eftir af summunni.

Lausn - PHP

```php
<?php
fscanf(STDIN, "%d\n", $n);
$ans = array();
$cur = 1;
while ($n >= $cur) {
    $ans[] = $cur;
    $n -= $cur;
    $cur++;
}
for($i = sizeof($ans) - $n; $i < sizeof($ans); $i++) {
    $ans[$i]++;
}
echo sizeof($ans);
echo "\n";
for($i = 0; $i < sizeof($ans); $i++) {
    echo $ans[$i];
    echo " ";
}
echo "\n";
```

Bjagað Beðaltal

- Eftir eina ítrun er listinn af lengd 3.
- Því tekur hver ítrun eftir það mjög stutta stund, svo við getum reiknað þetta einfaldlega með því að framkvæma nógu margar ítranir.
- Passa þarf að margfalda ekki allar 10^5 tölur saman og svo taka rót, því þá missist mikil nákvæmni.
- Taka má rætur fyrst, eða nota logra.

Lausn - Javascript - I/O

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});
let inp = [];
rl.on('line', (line) => {
    inp.push(line);
});
rl.on('close', () => {
    // Main code
});
```

Lausn - Javascript - Main

```
let n = parseInt(inp[0]);
let xs = inp[1].split(' ').map(x => parseFloat(x, 10));
for(let i = 0; i < 10000; ++i) {
    xs = [
        [...xs]
            .reduce((s, x) => s + x, 0.0) / xs.length,
        [...xs]
            .map(x => Math.pow(x, 1.0 / xs.length))
            .reduce((m, x) => m * x, 1.0),
        [...xs]
            .sort((a, b) => a-b)[Math.floor(xs.length / 2)]
    ]
}
console.log(xs[0]);
```

Höskuldarháska

- Engin trikk, bara gera það sem beðið er um.
- Hægt er að gera þetta með vigur af vísum og hækka alltaf afasta, með carry.
- Hægt að gera með innbyggðu eins og `itertools.product`.
- Einnig hægt að gera þetta endurkvæmt.

Lausn - Haskell

```haskell
import Data.List

main = interact ioparse

ioparse :: String -> String
ioparse s = unlines . sort . stringprod $
    map (tail . words) (tail $ lines s)

stringprod :: [[String]] -> [String]
stringprod [] = [""]
stringprod (x:xs) = concat $ map (prepend (stringprod xs)) x

prepend :: [String] -> String -> [String]
prepend [] y = []
prepend (x:xs) y = (y ++ " " ++ x) : prepend xs y
```

Austan Atlantshafs

- Hægt að gera á marga vegu
- Má gera DFS úr rót til að finna hæsta gildi undir hverri nóðu
- Eftir það er amortized hratt að labba upp í hágildið í hvert sinn
- Einnig má leysa þetta með binary jumping
- Þá má finna svarið í hverri nóðu í logratíma

Lausn - Rust - DFS

```rust
use std::collections::HashMap;
use std::io::BufRead;

fn dfs(adj: &Vec<Vec<usize>>, s: usize,
    high: &mut Vec<(i32, usize)>, xs: &Vec<i32>) {
    high[s] = (xs[s], 1);
    for y in &adj[s] {
        dfs(adj, *y, high, xs);
        if high[*y].0 > high[s].0 {
            high[s] = high[*y];
        } else if high[*y].0 == high[s].0 {
            high[s].1 += high[*y].1;
        }
    }
}
```

Lausn - Rust - I/O

```rust
fn main() {
    let mut buffer = String::new();
    let stdin = std::io::stdin();
    stdin.lock().read_line(&mut buffer).unwrap();
    let n: usize = buffer.trim().parse().unwrap();
    let mut g = vec![vec![]; n];
    let mut xs = vec![0; n];
    let mut names = vec![String::new(); n];
    let mut par = vec![String::new(); n];
    let mut ind: HashMap<String,usize> = HashMap::new();
    for i in 0..n {
        buffer.clear();
        stdin.lock().read_line(&mut buffer).unwrap();
        let mut tokens = buffer.trim().split_whitespace();
        names[i] = String::from(tokens.next().unwrap());
        par[i] = String::from(tokens.next().unwrap());
        xs[i] = tokens.next().unwrap().parse::<i32>().unwrap();
        ind.insert(names[i].clone(), i);
    }
```

Lausn - Rust - Main

```rust
    let mut high = vec![(0, 0); n];
    dfs(&g, 0, &mut high, &xs);
    for i in 0..n {
        if xs[i] == -1 { continue; }
        let mut j = i;
        loop {
            let k = ind[&par[j]];
            if k == j { break; }
            if high[k].1 > 1 { break; }
            if high[k].0 > high[i].0 { break; }
            j = k;
        }
        println!("{}", names[j]);
    }
}
```

Jaðarjuð

- Þetta er Union-Find!
- Hins vegar þarf að bjóða upp á undo.
- Best gert með því að halda utan um allt sem er gert í uppfærslu, skrifa það í stack.
- Þegar á að gera undo, er bara lagað allt sem er í fyrsta undo pakka á stack

Jaðarjuð

- Svo þarf einnig að halda utan um hver er í hverjum hóp.
- Þar kemur XKCD-ið inn, notum linked list.
- Geymum bara linked list í hverjum representative og sameinum þegar við gerum union.
- Muna að setja þær breytingar á undo stack líka!

Lausn - Zig - Unionfind init

```zig
const std = @import("std");

const UGUF = struct {
    const This = @This();
    fst: []usize, lst: []usize, nxt: []i32,
    op1: std.ArrayList(i32), op2: std.ArrayList(i32),

    fn init(n: usize, a: std.mem.Allocator) !This {
        var res = This {
            .fst = try a.alloc(usize, n),
            .lst = try a.alloc(usize, n),
            .nxt = try a.alloc(i32, n),
            .op1 = std.ArrayList(i32).init(a),
            .op2 = std.ArrayList(i32).init(a),
        };
        var i: usize = 0;
        while (i < n) : (i += 1) {
            res.fst[i] = i;
            res.lst[i] = i;
            res.nxt[i] = -1;
        }
        return res;
    }
```

Lausn - Zig - Unionfind functions

```
fn deinit(self: *This, a: std.mem.Allocator) void {
    a.free(self.fst);
    a.free(self.lst);
    a.free(self.nxt);
    self.op1.deinit();
    self.op2.deinit();
}
fn print_group(self: *This, a: usize, w: anytype) !void {
    var x = self.find(a);
    try w.print("{} ", .{ x + 1 });
    while(self.nxt[x] >= 0) {
        x = @intCast(self.nxt[x]);
        try w.print("{} ", .{ x + 1 });
    }
    try w.print("\n", .{});
}
fn find(self: *This, a: usize) usize {
    var x = a;
    while(self.fst[x] != x) {
        x = self.fst[x];
    }
    return x;
}
```

Lausn - Zig - Unionfind undo

```
fn undo(self: *This) void {
    if(self.op1.items.len == 0) {
        return;
    }
    if(self.op1.getLast() != -1) {
        self.nxt[@intCast(self.op1.getLast())] =
            @intCast(self.op2.getLast());
        _ = self.op1.pop(); _ = self.op2.pop();
        self.nxt[@intCast(self.op1.getLast())] =
            @intCast(self.op2.getLast());
        _ = self.op1.pop(); _ = self.op2.pop();
        self.fst[@intCast(self.op1.getLast())] =
            @intCast(self.op2.getLast());
        _ = self.op1.pop(); _ = self.op2.pop();
        self.lst[@intCast(self.op1.getLast())] =
            @intCast(self.op2.getLast());
    }
    _ = self.op1.pop(); _ = self.op2.pop();
}
```

Lausn - Zig - Unionfind unite

```
    fn unite(self: *This, a: usize, b: usize) !void {
        var x = self.find(a); var y = self.find(b);
        if(x == y) {
            try self.op1.append(-1);
            try self.op2.append(-1);
            return; }
        if(self.nxt[self.lst[x]] > self.nxt[self.lst[y]]) {
            const tmp = x; x = y; y = tmp; }
        try self.op1.append(@intCast(x));
        try self.op2.append(@intCast(self.lst[x]));
        try self.op1.append(@intCast(y));
        try self.op2.append(@intCast(self.fst[y]));
        try self.op1.append(@intCast(self.lst[x]));
        try self.op2.append(@intCast(self.nxt[self.lst[x]]));
        try self.op1.append(@intCast(self.lst[y]));
        try self.op2.append(@intCast(self.nxt[self.lst[y]]));
        self.nxt[self.lst[y]] += self.nxt[self.lst[x]];
        self.nxt[self.lst[x]] = @intCast(self.fst[y]);
        self.fst[y] = x;
        self.lst[x] = self.lst[y];
    }
};

fn read_num(r: anytype, del: u8) !i32 {
```

Lausn - Zig - Main part 1

```zig
fn read_num(r: anytype, del: u8) !i32 {
    var line_buf: [20]u8 = undefined;
    const amt = (try r.readUntilDelimiterOrEof(&line_buf, del)).?;
    const inp = try std.fmt.parseInt(i32, amt, 10);
    return inp;
}

pub fn main() !void {
    const in = std.io.getStdIn();
    const out = std.io.getStdOut();
    var bufout = std.io.bufferedWriter(out.writer());
    var bufin = std.io.bufferedReader(in.reader());
    const w = bufout.writer();
    const r = bufin.reader();
    const n = try read_num(r, ' ');
    var q = try read_num(r, '\n');
    var uguf = try UGUF.init(@intCast(n), std.heap.page_allocator);
    defer uguf.deinit(std.heap.page_allocator);
```

Lausn - Zig - Main part 2

```
    while (q >= 0) : (q -= 1) {
        var tiny_buf: [2]u8 = undefined;
        _ = try r.read(&tiny_buf);
        if(tiny_buf[0] == 'j') {
            const a = (try read_num(r, ' ')) - 1;
            const b = (try read_num(r, '\n')) - 1;
            try uguf.unite(@intCast(a), @intCast(b));
        } else if(tiny_buf[0] == 'u') {
            uguf.undo();
        } else if(tiny_buf[0] == 'p') {
            const a = (try read_num(r, '\n')) - 1;
            try uguf.print_group(@intCast(a), w);
        }
    }
    try bufout.flush();
}
```

Lafhræddir Læknar

- Athugum fyrst að ef við getum enst $k > 0$ daga, þá getum við enst $k - 1$ dag
- Helmingunarleitum því hvað við getum enst marga daga
- Þá er spurningin, að k gefnu, getum við enst k daga?

Lafhræddir Læknar

- Þá vitum við hvað þarf að gefa hverri læknategund marga skammta samtals
- Þá má leysa þetta sem flæðiverkefni!

Lafhræddir Læknar

- Setjum legg frá uppsprettu í öll epli með getu jafna fjölda þeirra epla
- Segjum legg frá hverri læknategund í niðurfall með getu jafna hvað sú tegund þarf að fá mörg epli samtals
- Setjum svo legg frá epli í læknategund með óendanlega getu ef sú eplategund virkar á þá læknategund
- Ef hámarksflæðið í gegn mettar alla leggi frá læknategundum í niðurfall, þá er þetta hægt fyrir þennan fjölda daga

Kjördæmi Königsbergs

- Beðin um að búa til net með n hnúta.
- Fáum skorður á nákvæmar gráður hnútanna d_i.
- Havel-Hakimi reikniritið leysir verkefnið upp að þessu.
- Fáum líka skorðuna að netið þurfi að vera samhangandi.

- Byrjum með Havel-Hakimi reikniritið.
- Veljum einhvern hnút u með $d_u > 0$.
- Tengjum legg milli u og allra hnúta v sem hafa hæstu d_u gráðurnar.
- Endurtökum þetta þangað til við getum ekki valið hnút.
- Mögulega náum ekki að tengja allt og þá er ómögulegt.
- Nú höfum við net, en það er ekki endilega samhangandi.

- Hættan er að við tengjum lága gráðu við lága gráðu og sköpum einangraðan þátt.
- Einföld lausn hér því við ráðum hvaða hnút við veljum.
- Gráðugt veljum u með lægsta d_u.
- Ef tekst að búa til tengt net að lokum þá er það svar.
- Annars ómögulegt.

Glötuð gildi

- Fyrst þarf að ákvarða hvaða runur eru ógildar, eins og t.d. $0, 0, 3, 3$.
- Til er setning, Landau's theorem, sem er hægt að stinga beint inn í

Glötuð gildi

- Ekki svo margar raðaðar runur af heiltölum ≤ 16 sem mynda gildar stigatöflur
- Getum framkvæmt kvika bestun á gildu rununum
- Mikilvægt að raða alltaf áður en flett er upp í minnistöflu!

Glötuð gildi

- Ef stærsta gildi rununnar er x þá verður síðasti að sigra x aðra
- Prófum allar leiðir til að draga 1 frá x öðrum gildum og fjarlægja x úr runu
- Summa svaranna fyrir allar þessar nýjar runur er þá svarið okkar
- Þetta dugar, það má gera aðeins betur, en það er óþarfi í þessu dæmi

Einfasa Eindahraðall

- Þetta er einfaldlega prefix sum dæmi!
- En því miður er það í allt að 10 víddum!
- Í einni er summuformúlan 2 liðir, í tveimur 4 liðir og í þremur 8 liðir
- Því þarf einhverja leið til að halda utan um summu sem getur verið af stærð allt að 2^{10}

Einfasa Eindahraðall

- Fyrir hverja vídd i erum við að skoða eitthvað bil $[a_i, b_i]$, af n víddum
- Í n víddum summum við yfir öll 2^n hlutmengi $\{1, 2, \ldots, n\}$
- Ef x er í menginu bætu takmörkum við okkur við gildin $[0, a_x[$ í þeirri vídd, en $[a_x, b_x]$ annars
- Ef við mengið er af odda stærð drögum við fjölda punkta sem við finnum frá summunni, annars bætum við við

Einfasa Eindahraðall

- Allt gott og blessað, fáum $\mathcal{O}(n2^n)$ lausn
- En það er of hægt

Einfasa Eindahraðall

- Þurfum að spara tíma
- Þegar við skoðum öll hlutmengin gerum við það í gray code röð
- Þá getum við haldið utan um mengið og uppfært það í $\mathcal{O}(1)$ per ítrun með því að bæta við eða fjarlægja eitt stak
- Þetta sparar nægan tíma til að fá $\mathcal{O}(2^n)$ lausn

Fljúgandi furðuhlutir forðast fókus

- Athugum að ef hægt er að komast á leiðarenda eftir $k > 0$ uppfærslur, þá er einnig hægt að komast eftir $k - 1$ uppfærslu.
- Getum því notað helmingunarleit.
- Breytum spurningunni í að gá hvort hægt sé að komast eftir k uppfærslur, fyrir gefið k.

Fljúgandi furðuhlutir forðast fókus

- Hér þyngist málið, svo vægast sé sagt.
- Snið kúlnanna við planið sem við fljúgum í eru skífur.
- Allt sem við kemur að finna öll pör skífa sem skerast er of hægt, því það er kvaðratískt.

Fljúgandi furðuhlutir forðast fókus

- Viljum eitthvað eins og Delaunay triangulation, en það gefur ranga niðurstöðu hér.
- Til er fyrirbæri sem kallast vigtað Delaunay triangulation, einnig þekkt sem power diagram.
- En að reikna það beint er hrottalegt.

Fljúgandi furðuhlutir forðast fókus

- Reikna má Delaunay triangulation á punktum (x_i, y_i) útfrá 3D-hull á punktunum $(x_i, y_i, x_i^2 + y_i^2)$
- Eins má reikna power diagram útfrá $(x_i, y_i, x_i^2 + y_i^2 - r^2)$ þar sem r er radíus samsvarandi hrings

Fljúgandi furðuhlutir forðast fókus

- Fáum þá triangulation þar sem allir leggir sem þarf til að leysa dæmið koma fyrir.
- Þurfum hins vegar að skoða alla leggi og sjá hvort þeir samsvari raunverulega skurði skífa.
- Notum union-find til að sameina þríhyrninga sem eru ekki raunverulega aðskildir.

Fljúgandi furðuhlutir forðast fókus

- Loks getum við þá gáð hvort upphafs og endapunktur séu í sama svæði að öllu þessu loknu.
- Endurtökum svo bara þetta allt $\log(n)$ sinnum!
- Nota má mörg reiknirit til að gera 3D-hull, við notuðum Clarkson-Shor.