Programming contests tutorial

or

how to improve when you're already decent

Przemek Uznański

June 28, 2017

About myself

from University of Wrocław, Poland

Programming competitions: 2003-2013

- ACM ICPC,
- Google Code Jam,
- Facebook Hacker Cup,
- TopCoder

Coach of highschoolers (IOI) in Wrocław:

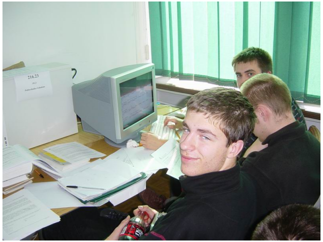
About myself

from University of Wrocław, Poland

Uniwersytet Wrocławski

Programming competitions: 2003-2013

- ACM ICPC,
- Google Code Jam,
- Facebook Hacker Cup,
- TopCoder



(2004, my first onsite)

Coach of highschoolers (IOI) in Wrocław:

End Goal:

End Goal:


The Fortieth Annual
ACM-ICPC
World Finals
May 19, 2016
Phuket Thailand


TCO16
topcoder
cisco
docker
rng_58
MAKOTO SOEJIMA

How to get better?

I asked a few friends: “Any tips/tricks for getting better?”

- Just go to `codeforces.com` and start solving problems.
- Solve 100 problems/year..
- After few years..

How to get better?

I asked a few friends: "Any tips/tricks for getting better?"

- Just go to `codeforces.com` and start solving problems.
- Solve 100 problems/year..
- After few years..

Internet:

Practice, practice, practice. A lot.

It's one thing that you know how to solve stuff in theory. It's a completely different thing that you can code it, it compiles and runs, gives no errors, you made sure all variables are big enough so they will fit, it is fast enough and doesn't use too much memory.

Learning to code is all about practicing. **Participate regularly** in the programming contests. Solve the ones that you cannot solve in the

Competitive Programming requires a lot, A LOT of practice. Like, years of practice if you want to excel. You should not lose confidence on failing to solve problems, since this will happen a lot. Even after you're good.

How to get better?

I asked a few friends: "Any tips/tricks for getting better?"

- Just go to `codeforces.com` and start solving problems.
- Solve 100 problems/year..
- After few years..

Internet:

Practice, practice, practice. A lot.

It's one thing that you know how to solve stuff in theory. It's a completely different thing that you can code it, it compiles and runs, gives no errors, you made sure all variables are big enough so they will fit, it is fast enough and doesn't use too much memory.

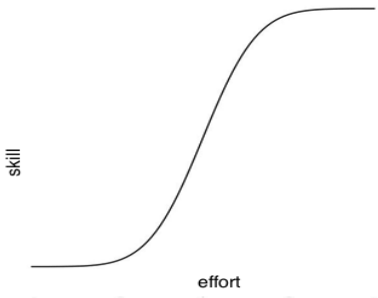
Learning to code is all about practicing. **Participate regularly** in the programming contests. Solve the ones that you cannot solve in the

Competitive Programming requires a lot, A LOT of practice. Like, years of practice if you want to excel. You should not lose confidence on failing to solve problems, since this will happen a lot. Even after you're good.

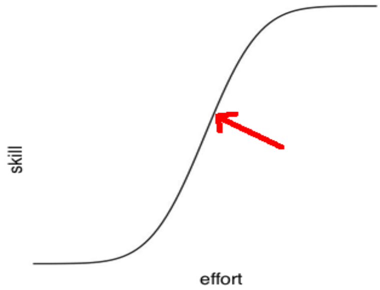
Plan for today:

- how to train effectively
- advanced topics:
  - number theory
  - hashing
- tips&tricks

How to train effectively?



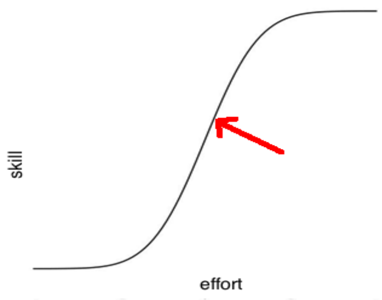
How to train effectively?



Diminishing returns:

lot of effort, little improvement

How to train effectively?



How to solve problems.

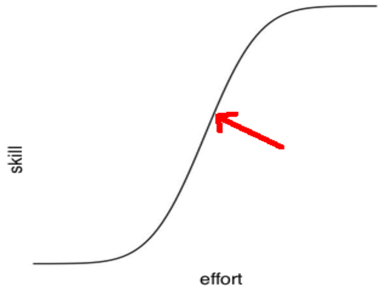
vs.

How to approach whole classes of problems.

Diminishing returns:

lot of effort, little improvement

How to train effectively?



How to solve problems.
vs.
How to approach whole classes of problems.

Diminishing returns:
lot of effort, little improvement

Do not try to rediscover the wheel!

Number theory

Number theory

Typical tasks:

Number theory

Typical tasks:

- primality testing
- functions: $\varphi(n)$ (Euler's totient function), $\sigma(n)$ (sum of divisors), $\tau(n)$ (number of divisors), $\omega(n)$ (distinct prime divisors)
- factorization of n: factorizing single number, precomputing all numbers factorization

Number theory

Typical tasks:

- primality testing
- functions: $\varphi(n)$ (Euler's totient function), $\sigma(n)$ (sum of divisors), $\tau(n)$ (number of divisors), $\omega(n)$ (distinct prime divisors)
- factorization of n: factorizing single number, precomputing all numbers factorization

Building block: sieve of Eratosthenes

Claim:

Sieve of Eratosthenes to rule them all.

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Sieve of Eratosthenes - vanilla

```
vector<bool> isprime(n,true);
for(int i=2;i<n;i++)
    if( isprime[i] )
        for(int j=2*i;j<n;j+=i)
            isprime[j] = false;
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

How to extract factorization of numbers?

Factorization

Idea:

For every number, store ONE of prime divisors.

$$\text{factor}(n) = \text{divisor}(n) \cdot \text{factor}\left(\frac{n}{\text{divisor}(n)}\right)$$

Factorization

Idea:

For every number, store ONE of prime divisors.

$$\text{factor}(n) = \text{divisor}(n) \cdot \text{factor}\left(\frac{n}{\text{divisor}(n)}\right)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	2		2		2		2		2		2		2		2		2		2		2		2	

Factorization

Idea:

For every number, store ONE of prime divisors.

$$\text{factor}(n) = \text{divisor}(n) \cdot \text{factor}\left(\frac{n}{\text{divisor}(n)}\right)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	2	3	2		3		2	3	2		3		2	3	2		3		2	3	2		3	

Factorization

Idea:

For every number, store ONE of prime divisors.

$$\text{factor}(n) = \text{divisor}(n) \cdot \text{factor}\left(\frac{n}{\text{divisor}(n)}\right)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	2	3	2	5	3		2	3	5		3		2	5	2		3		5	3	2		3	5

Factorization

Idea:

For every number, store ONE of prime divisors.

$$\text{factor}(n) = \text{divisor}(n) \cdot \text{factor}\left(\frac{n}{\text{divisor}(n)}\right)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	2	3	2	5	3	7	2	3	5	11	3	13	2	5	2	17	3	19	5	3	2	23	3	5

Factorization

Idea:

For every number, store ONE of prime divisors.

$$\text{factor}(n) = \text{divisor}(n) \cdot \text{factor}\left(\frac{n}{\text{divisor}(n)}\right)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	2	3	2	5	3	7	2	3	5	11	3	13	2	5	2	17	3	19	5	3	2	23	3	5

```
vector<int> divisor(n);
for(int i=2;i<n;i++)
    if( divisor[i]==0 )
        for(int j=i;j<n;j+=i)
            divisor[j] = i;
```

Observe:

We are actually computing THE LARGEST prime factor.

Extending factorization

Let's precompute $\tau(n)$ (number of divisors).

$$n = \prod_p p^{\nu_n(p)}$$

Extending factorization

Let's precompute $\tau(n)$ (number of divisors).

$$n = \prod_p p^{\nu_n(p)} \qquad \tau(n) = \prod_p (\nu_n(p) + 1)$$

- keep count on number of repetitions of factor
- $\tau(n)$ follows from $\tau(n/\mathrm{divisor}(n))$

Extending factorization

Let's precompute $\tau(n)$ (number of divisors).

$$n = \prod_p p^{\nu_n(p)} \qquad\qquad \tau(n) = \prod_p (\nu_n(p) + 1)$$

- keep count on number of repetitions of factor
- $\tau(n)$ follows from $\tau(n/\mathrm{divisor}(n))$

```
vector<int> cnt(n);
for(int i=2;i<n;i++)
    if( divisor[i]==i )
        cnt[i] = 1;
    else {
        int j = i/divisor[i];
        if( divisor[i]==divisor[j] )
            cnt[i] = cnt[j]+1;
        else
            cnt[i] = 1;
    }
```

Extending factorization

Let's precompute $\tau(n)$ (number of divisors).

$$n = \prod_p p^{\nu_n(p)} \qquad \tau(n) = \prod_p (\nu_n(p) + 1)$$

- keep count on number of repetitions of factor
- $\tau(n)$ follows from $\tau(n/\mathrm{divisor}(n))$

```
vector<int> tau(n);
for(int i=2;i<n;i++)
    if( divisor[i]==i ){
        tau[i] = 2;
    } else {
        int j = i/divisor[i];
        tau[i] = tau[j]/cnt[i] * (cnt[i]+1);
    }
```

Exercise session

Hashing

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

$A =$

b	a	n	a	n	a	n	a

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

$A =$

b	a	n	a	n	a	n	a

$A[2..5] = \texttt{anan}$ $A[4..7] = \texttt{anan}$

YES

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

$A =$

b	a	n	a	n	a	n	a

$A[1..4] = \texttt{bana}$ $A[4..7] = \texttt{anan}$

NO

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

$A =$

b	a	n	a	n	a	n	a

$A[1..4] = \texttt{bana}$ $A[4..7] = \texttt{anan}$

NO

Given two unrooted trees decide if they are isomorphic.

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

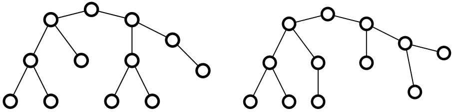
$A =$

b	a	n	a	n	a	n	a

$A[1..4] = \texttt{bana}$ $A[4..7] = \texttt{anan}$

NO

Given two unrooted trees decide if they are isomorphic.



YES

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

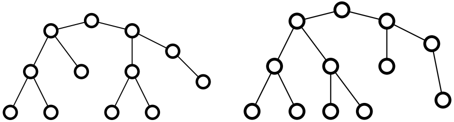
$A =$

b	a	n	a	n	a	n	a

$A[1..4] = \texttt{bana}$ $\quad A[4..7] = \texttt{anan}$

NO

Given two unrooted trees decide if they are isomorphic.



NO

Hashing

String A, queries:
is $A[i..i+\ell] = A[j..j+\ell]$?

example:

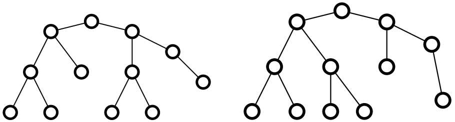
$A =$

b	a	n	a	n	a	n	a

$A[1..4] = \texttt{bana}$ $\quad$ $A[4..7] = \texttt{anan}$

NO

Given two unrooted trees decide if they are isomorphic.



NO

IDEA:

hash objects, use recursive structure
objects = substrings, subtrees

Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

b

a

n

a

n

a

n

a

f

Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

b	a
a	a
n	a
a	a
n	b
a	f
n	n
a	n
f	n

Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

b	a	a = 1
a	a	b = 2
n	a	f = 3
a	a	n = 4
n	b	
a	f	
n	n	
a	n	
f	n	

Subwords - Karp Miller Rosenberg algorithm

$A =$ | b | a | n | a | n | a | n | a | f |

$\texttt{ba} = (\texttt{b}, \texttt{a})$

$\texttt{an} = (\texttt{a}, \texttt{n})$

$\texttt{na} = (\texttt{n}, \texttt{a})$

$\texttt{an} = (\texttt{a}, \texttt{n})$

$\texttt{na} = (\texttt{n}, \texttt{a})$

$\texttt{an} = (\texttt{a}, \texttt{n})$

$\texttt{na} = (\texttt{n}, \texttt{a})$

$\texttt{af} = (\texttt{a}, \texttt{f})$

$\texttt{a} = 1$

$\texttt{b} = 2$

$\texttt{f} = 3$

$\texttt{n} = 4$

Subwords - Karp Miller Rosenberg algorithm

$A =$ | b | a | n | a | n | a | n | a | f |

b	a	n	a	n	a	n	a	f

$\texttt{ba} = (\texttt{b}, \texttt{a}) = (2, 1)$

$\texttt{an} = (\texttt{a}, \texttt{n}) = (1, 4)$

$\texttt{na} = (\texttt{n}, \texttt{a}) = (4, 1)$

$\texttt{an} = (\texttt{a}, \texttt{n}) = (1, 4)$

$\texttt{na} = (\texttt{n}, \texttt{a}) = (2, 1)$

$\texttt{an} = (\texttt{a}, \texttt{n}) = (1, 4)$

$\texttt{na} = (\texttt{n}, \texttt{a}) = (4, 1)$

$\texttt{af} = (\texttt{a}, \texttt{f}) = (1, 3)$

$\texttt{a} = 1$

$\texttt{b} = 2$

$\texttt{f} = 3$

$\texttt{n} = 4$

Subwords - Karp Miller Rosenberg algorithm

$A =$ | b | a | n | a | n | a | n | a | f |

$\mathtt{ba} = (\mathtt{b}, \mathtt{a}) = (2, 1)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (2, 1)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{af} = (\mathtt{a}, \mathtt{f}) = (1, 3)$

$\mathtt{af} = (\mathtt{a}, \mathtt{f}) = (1, 3)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{ba} = (\mathtt{b}, \mathtt{a}) = (2, 1)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$

$\mathtt{a} = 1$
$\mathtt{b} = 2$
$\mathtt{f} = 3$
$\mathtt{n} = 4$

Subwords - Karp Miller Rosenberg algorithm

$A =$	b	a	n	a	n	a	n	a	f

$\mathtt{ba} = (\mathtt{b}, \mathtt{a}) = (2, 1)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (2, 1)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{af} = (\mathtt{a}, \mathtt{f}) = (1, 3)$

$\mathtt{af} = (\mathtt{a}, \mathtt{f}) = (1, 3)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{an} = (\mathtt{a}, \mathtt{n}) = (1, 4)$
$\mathtt{ba} = (\mathtt{b}, \mathtt{a}) = (2, 1)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$
$\mathtt{na} = (\mathtt{n}, \mathtt{a}) = (4, 1)$

$\mathtt{a} = 1$
$\mathtt{b} = 2$
$\mathtt{f} = 3$
$\mathtt{n} = 4$
$\mathtt{af} = 5$
$\mathtt{an} = 6$
$\mathtt{ba} = 7$
$\mathtt{na} = 8$

Subwords - Karp Miller Rosenberg algorithm

$A =$ | b | a | n | a | n | a | n | a | f |

$\texttt{bana} = (\texttt{ba}, \texttt{na})$	$\texttt{a} = 1$
$\texttt{anan} = (\texttt{an}, \texttt{an})$	$\texttt{b} = 2$
$\texttt{nana} = (\texttt{na}, \texttt{na})$	$\texttt{f} = 3$
$\texttt{anan} = (\texttt{an}, \texttt{an})$	$\texttt{n} = 4$
$\texttt{nana} = (\texttt{na}, \texttt{na})$	$\texttt{af} = 5$
$\texttt{anaf} = (\texttt{an}, \texttt{af})$	$\texttt{an} = 6$
	$\texttt{ba} = 7$
	$\texttt{na} = 8$

Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

$\texttt{bana} = (\texttt{ba}, \texttt{na}) = (7, 8)$

$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$

$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$

$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{anaf} = (\texttt{an}, \texttt{af}) = (6, 5)$

$\texttt{a} = 1$

$\texttt{b} = 2$

$\texttt{f} = 3$

$\texttt{n} = 4$

$\texttt{af} = 5$

$\texttt{an} = 6$

$\texttt{ba} = 7$

$\texttt{na} = 8$

Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

$\texttt{bana} = (\texttt{ba}, \texttt{na}) = (7, 8)$

$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$

$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$

$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{anaf} = (\texttt{an}, \texttt{af}) = (6, 5)$

$\texttt{anaf} = (\texttt{an}, \texttt{af}) = (6, 5)$

$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$

$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$

$\texttt{bana} = (\texttt{ba}, \texttt{na}) = (7, 8)$

$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{a} = 1$

$\texttt{b} = 2$

$\texttt{f} = 3$

$\texttt{n} = 4$

$\texttt{af} = 5$

$\texttt{an} = 6$

$\texttt{ba} = 7$

$\texttt{na} = 8$

Subwords - Karp Miller Rosenberg algorithm

$$A = \boxed{\texttt{b}}\boxed{\texttt{a}}\boxed{\texttt{n}}\boxed{\texttt{a}}\boxed{\texttt{n}}\boxed{\texttt{a}}\boxed{\texttt{n}}\boxed{\texttt{a}}\boxed{\texttt{f}}$$

$\texttt{bana} = (\texttt{ba}, \texttt{na}) = (7, 8)$
$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$
$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$
$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$
$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$
$\texttt{anaf} = (\texttt{an}, \texttt{af}) = (6, 5)$

$\texttt{anaf} = (\texttt{an}, \texttt{af}) = (6, 5)$
$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$
$\texttt{anan} = (\texttt{an}, \texttt{an}) = (6, 6)$
$\texttt{bana} = (\texttt{ba}, \texttt{na}) = (7, 8)$
$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$
$\texttt{nana} = (\texttt{na}, \texttt{na}) = (8, 8)$

$\texttt{a} = 1$
$\texttt{b} = 2$
$\texttt{f} = 3$
$\texttt{n} = 4$
$\texttt{af} = 5$
$\texttt{an} = 6$
$\texttt{ba} = 7$
$\texttt{na} = 8$

$\texttt{anaf} = 9$
$\texttt{anan} = 10$
$\texttt{bana} = 11$
$\texttt{nana} = 12$

Subwords - Karp Miller Rosenberg algorithm

$A =$ | b | a | n | a | n | a | n | a | f |

$\texttt{anananaf} = (\texttt{anan}, \texttt{anaf}) = (10, 9)$

$\texttt{bananana} = (\texttt{bana}, \texttt{nana}) = (11, 12)$

a	= 1	anaf	= 9
b	= 2	anan	= 10
f	= 3	bana	= 11
n	= 4	nana	= 12
af	= 5	anananaf	= 13
an	= 6	bananana	= 14
ba	= 7		
na	= 8		

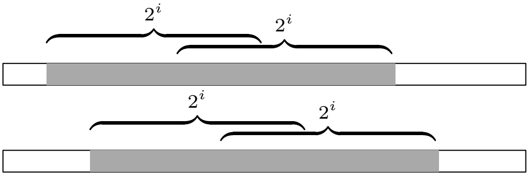
Subwords - Karp Miller Rosenberg algorithm

$A =$

b	a	n	a	n	a	n	a	f

KMR =

2	1	4	1	4	1	4	1	3
7	6	8	6	8	6	8	5	
11	10	12	10	12	9			
14	13							



a = 1
b = 2
f = 3
n = 4
af = 5
an = 6
ba = 7
na = 8

anaf = 9
anan = 10
bana = 11
nana = 12
anananaf = 13
bananana = 14

Pseudocode

```
cnt = 0;
vector<pair<char,int> > letter_position(n);
for(int i=0;i<n;i++)
    letter_position[i] = make_pair(A[i],i);
sort(letter_position.begin(),letter_position.end());
for(int i=0;i<n;i++)
{
    if(i==0 || letter_position[i].first != letter_position[i-1].first)
        cnt++;
    KMR[letter_position[i]][0] = cnt;
}

for(j=1; (1<<j) <= n; j++)
{
    vector<pair<pair<int,int>,int> > pair_position(n);
    for(int i=0;i<n;i++)
        pair_position[i] = make_pair(make_pair(KMR[i][j-1],KMR[i+(1<<j-1)][j]),i);
    sort(pair_position.begin(),pair_position.end());
    for(int i=0;i+(1<<j)<=n;i++)
    {
        if(i==0 || pair_position[i].first != pair_position[i-1].first)
            cnt++;
        KMR[pair_position[i]][j] = cnt;
    }
}
```

Hashing

KMR:

- time complexity $\mathcal{O}(n \log^2 n)$ or $\mathcal{O}(n \log n)$ - too slow
- memory usage $\mathcal{O}(n \log n)$ - too large
- computes "too much"

Testing identity of objects: find hash function h, that:

- $h(X) = h(Y)$ when $X \sim Y$
- $h(X) \neq h(Y)$ when $X \nsim Y$ (hopefully)
- $h(X)$ can be computed using recursive structure of X

Hashing

Required:

- $h(X) = h(Y)$ when $X \sim Y$
- $h(X) \neq h(Y)$ when $X \nsim Y$ (hopefully)
- $h(X)$ can be computed using recursive structure of X

Hashing

Required:

- $h(X) = h(Y)$ when $X \sim Y$
- $h(X) \neq h(Y)$ when $X \nsim Y$ (hopefully)
- $h(X)$ can be computed using recursive structure of X

- Fix large prime p
- $h(A[1..2^i]) = \left(h(A[1..2^{i-1}]) + x_i \cdot h(A[2^{i-1}+1..2^i])\right) \bmod p$,
- where x_i is picked randomly and depends only on i

Subwords - hashing

$A =$

b	a	n	a	n	a	n	a	f

$p = 1009$

Subwords - hashing

$A =$ | b | a | n | a | n | a | n | a | f | $\quad p = 1009$

b	a	n	a	n	a	n	a	f

$$h(\texttt{b}) = 2$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{n}) = 14$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{n}) = 14$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{n}) = 14$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{f}) = 6$$

Subwords - hashing

$A =$ | b | a | n | a | n | a | n | a | f | $\quad p = 1009$

$h(\texttt{b}) = 2$	$h(\texttt{ba}) = 2 + 10 * 1 = 12$
$h(\texttt{a}) = 1$	$h(\texttt{an}) = 1 + 10 * 14 = 141$
$h(\texttt{n}) = 14$	$h(\texttt{na}) = 14 + 10 * 1 = 24$
$h(\texttt{a}) = 1$	$h(\texttt{an}) = 1 + 10 * 14 = 141$
$h(\texttt{n}) = 14$	$h(\texttt{na}) = 14 + 10 * 1 = 24$
$h(\texttt{a}) = 1$	$h(\texttt{an}) = 1 + 10 * 14 = 141$
$h(\texttt{n}) = 14$	$h(\texttt{na}) = 14 + 10 * 1 = 24$
$h(\texttt{a}) = 1$	$h(\texttt{af}) = 1 + 10 * 6 = 61$
$h(\texttt{f}) = 6$	

Subwords - hashing

$A =$	b	a	n	a	n	a	n	a	f

$p = 1009$

$$h(\texttt{b}) = 2$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{n}) = 14$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{n}) = 14$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{n}) = 14$$
$$h(\texttt{a}) = 1$$
$$h(\texttt{f}) = 6$$

$$h(\texttt{ba}) = 2 + {\color{red}10} * 1 = 12$$
$$h(\texttt{an}) = 1 + {\color{red}10} * 14 = 141$$
$$h(\texttt{na}) = 14 + {\color{red}10} * 1 = 24$$
$$h(\texttt{an}) = 1 + {\color{red}10} * 14 = 141$$
$$h(\texttt{na}) = 14 + {\color{red}10} * 1 = 24$$
$$h(\texttt{an}) = 1 + {\color{red}10} * 14 = 141$$
$$h(\texttt{na}) = 14 + {\color{red}10} * 1 = 24$$
$$h(\texttt{af}) = 1 + {\color{red}10} * 6 = 61$$

$$h(\texttt{bana}) = 12 + {\color{red}28} * 24 = 684$$
$$h(\texttt{anan}) = 141 + {\color{red}28} * 141 = 53$$
$$h(\texttt{nana}) = 24 + {\color{red}28} * 24 = 696$$
$$h(\texttt{anan}) = 141 + {\color{red}28} * 141 = 53$$
$$h(\texttt{nana}) = 24 + {\color{red}28} * 24 = 696$$
$$h(\texttt{anaf}) = 141 + {\color{red}28} * 61 = 840$$

Subwords - hashing

$A =$ | b | a | n | a | n | a | n | a | f | $p = 1009$

$h(\texttt{b}) = 2$
$h(\texttt{a}) = 1$
$h(\texttt{n}) = 14$
$h(\texttt{a}) = 1$
$h(\texttt{n}) = 14$
$h(\texttt{a}) = 1$
$h(\texttt{n}) = 14$
$h(\texttt{a}) = 1$
$h(\texttt{f}) = 6$

$h(\texttt{ba}) = 2+10*1 = 12$
$h(\texttt{an}) = 1+10*14 = 141$
$h(\texttt{na}) = 14+10*1 = 24$
$h(\texttt{an}) = 1+10*14 = 141$
$h(\texttt{na}) = 14+10*1 = 24$
$h(\texttt{an}) = 1+10*14 = 141$
$h(\texttt{na}) = 14+10*1 = 24$
$h(\texttt{af}) = 1+10*6 = 61$

$h(\texttt{bana}) = 12+28*24 = 684$
$h(\texttt{anan}) = 141+28*141 = 53$
$h(\texttt{nana}) = 24+28*24 = 696$
$h(\texttt{anan}) = 141+28*141 = 53$
$h(\texttt{nana}) = 24+28*24 = 696$
$h(\texttt{anaf}) = 141+28*61 = 840$

$h(\texttt{ananaf}) = 53+190*840 = 231$
$h(\texttt{bananana}) = 684+190*696 = 745$

Pseudocode

```
for(int i=0;i<n;i++)
     hash[i][0] = A[i]%p;
for(j=1; (1<<j) <= n; j++)
{
     x = 1+rand()%(p-1);
     for(int i=0;i+(1<<j)<=n;i++)
          hash[i][j] = (hash[i][j-1] + x*hash[i+(1<<j-1)][j-1])%p;
}
```

Pseudocode

```
for(int i=0;i<n;i++)
    hash[i][0] = A[i]%p;
for(j=1; (1<<j) <= n; j++)
{
    x = 1+rand()%(p-1);
    for(int i=0;i+(1<<j)<=n;i++)
        hash[i][j] = (hash[i][j-1] + x*hash[i+(1<<j-1)][j-1])%p;
}
```

hashing:

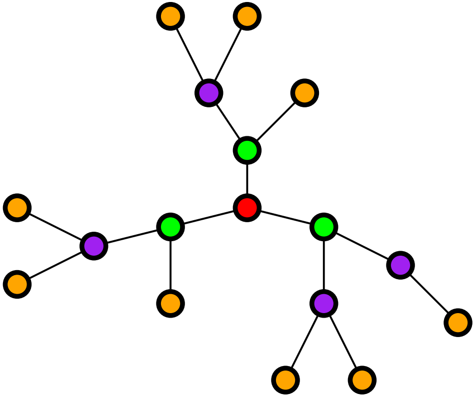
- time complexity $\mathcal{O}(n \log n)$
- much "nicer" memory access – much faster
- memory usage $\mathcal{O}(n \log n)$ - too large
- randomized (small probability of collision)

There is even simpler way of hashing, with $\mathcal{O}(n)$ time and memory, and $\mathcal{O}(1)$ queries. See: Karp-Rabin fingerprints.

Tree isomorphism - labeling

Idea:

- go bottom-up level by level
- label subtree roots

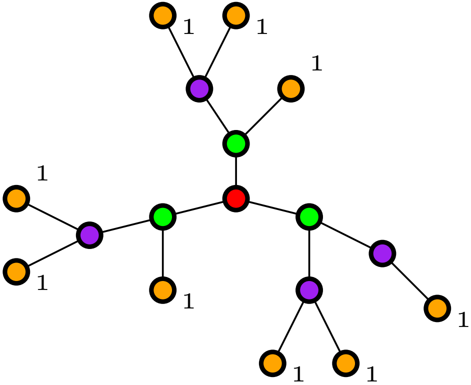


Tree isomorphism - labeling

Idea:

- go bottom-up level by level
- label subtree roots

$$1 = \{\}$$

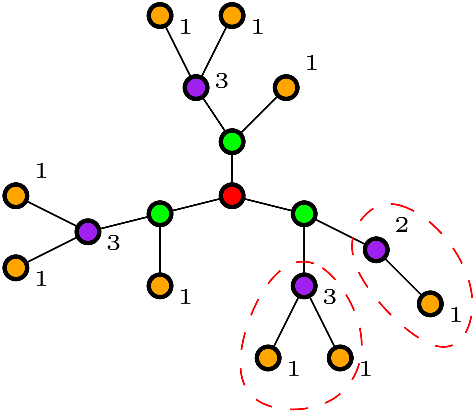


Tree isomorphism - labeling

Idea:

- go bottom-up level by level
- label subtree roots

$$1 = \{\}$$
$$2 = \{1\}$$
$$3 = \{1, 1\}$$

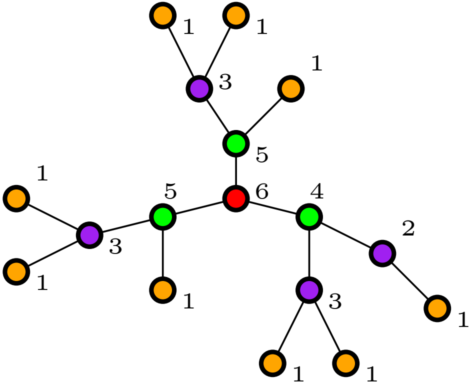


Tree isomorphism - labeling

Idea:

- go bottom-up level by level
- label subtree roots

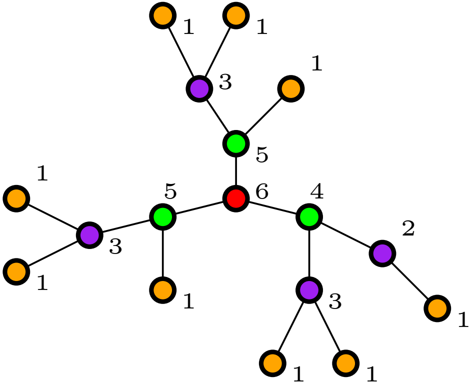
$$1 = \{\}$$
$$2 = \{1\}$$
$$3 = \{1, 1\}$$
$$4 = \{2, 3\}$$
$$5 = \{1, 3\}$$
$$6 = \{4, 5, 5\}$$



Tree isomorphism - labeling

Idea:

- go bottom-up level by level
- label subtree roots



$$1 = \{\}$$
$$2 = \{1\}$$
$$3 = \{1, 1\}$$
$$4 = \{2, 3\}$$
$$5 = \{1, 3\}$$
$$6 = \{4, 5, 5\}$$

Hint:

for each node, sort its children list

Tree isomorphism - hashing

Idea:

- go bottom-up level by level
- hash subtree roots

Tree isomorphism - hashing

Idea:

- go bottom-up level by level
- hash subtree roots

Solution:

sort $t_1, t_2, ..$, for node at level i:

$h(\{t_1, t_2, ..\}) =$

$h(t_1) + h(t_2) \cdot X_i + h(t_3) \cdot X_i^2 + ..$

Tree isomorphism - hashing

Idea:

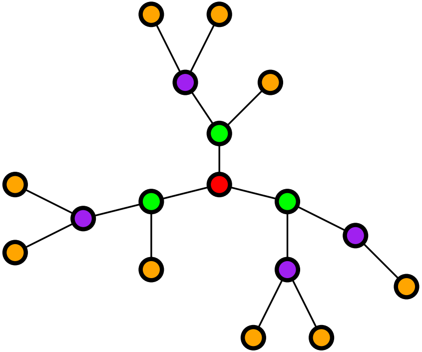
- go bottom-up level by level
- hash subtree roots

Solution:

sort $t_1, t_2, ..,$ for node at level i:
$h(\{t_1, t_2, ..\}) =$
$h(t_1) + h(t_2) \cdot X_i + h(t_3) \cdot X_i^2 + ..$

Careful:

Different node symmetry (mirror-invariant, rotation-invariant, reordering-invariant) require different $h()$.



Tips&Tricks

Tips&Tricks

Tip 1

Learn your tools.

- `bash` syntax
- `grep`
- `head`, `tail`
- `wc`
- `bc`
- `factor`

Tips&Tricks

Tip 1

Learn your tools.

Try:

```
> bc -l
scale = 1000
a(1)*4
e(1)
```

- bash syntax
- `grep`
- `head`, `tail`
- `wc`
- `bc`
- `factor`

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?
- What does `while(cin >> x)` do?

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?
- What does `while(cin >> x)` do?
- `for(auto& i : v)` construction.

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?
- What does `while(cin >> x)` do?
- `for(auto& i : v)` construction.
- `stringstream`

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?
- What does `while(cin >> x)` do?
- `for(auto& i : v)` construction.
- `stringstream`
- `vector<>` and `g++ -O2` vs. `g++ -O0`

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?
- What does `while(cin >> x)` do?
- `for(auto& i : v)` construction.
- `stringstream`
- `vector<>` and `g++ -O2` vs. `g++ -O0`
- What's wrong with: `for(int i = 0; i < x.size()-1; i++)`?

Tips&Tricks

Tip 2

Learn language and library.

- What is the memory usage of empty `vector<>`?
- What does `while(cin >> x)` do?
- `for(auto& i : v)` construction.
- `stringstream`
- `vector<>` and `g++ -O2` vs. `g++ -O0`
- What's wrong with: `for(int i = 0; i < x.size()-1; i++)`?
- `g++ -ftrapv`

Tips&Tricks

Tip 3

Keep having fun!

Exercise session