Online Context-Free Recognition in OMv Time

Bartłomiej Dudek[1] Paweł Gawrychowski[1]

[1] University of Wrocław, Poland

Context-free language

Context-free grammar is a tuple: $G = (V_N, V_T, P, S)$ where:

- V_N - set of non-terminals
- V_T - set of terminals (alphabet)
- P - production rules
- S - starting non-terminal

Example: productions P for correct bracketing (e.g. [[][]]):

$$S \to \varepsilon$$
$$S \to SS$$
$$S \to [S]$$

Chomsky normal form: all productions are either $S \to AB$ or $S \to c$.

Context-free language

Context-free grammar is a tuple: $G = (V_N, V_T, P, S)$ where:

- V_N - set of non-terminals $(V_N = \{S\})$
- V_T - set of terminals (alphabet) $(V_T = \{[,]\})$
- P - production rules
- S - starting non-terminal

Example: productions P for correct bracketing (e.g. [[][]]):

$$S \to \varepsilon$$
$$S \to SS$$
$$S \to [S]$$

Chomsky normal form: all productions are either $S \to AB$ or $S \to c$.

Context-free language

Context-free grammar is a tuple: $G = (V_N, V_T, P, S)$ where:

- V_N - set of non-terminals
- V_T - set of terminals (alphabet)
- P - production rules
- S - starting non-terminal

Example: productions P for correct bracketing (e.g. [[][]]):

$$S \to \varepsilon$$
$$S \to SS$$
$$S \to [S]$$

Chomsky normal form: all productions are either $S \to AB$ or $S \to c$.

Context-free language

Context-free grammar is a tuple: $G = (V_N, V_T, P, S)$ where:

- V_N - set of non-terminals $(V_N = \{S, O, T, C\})$
- V_T - set of terminals (alphabet) $(V_T = \{[,]\})$
- P - production rules
- S - starting non-terminal

Example: productions P for correct bracketing (e.g. [[][]]):

$S \to \varepsilon$	$O \to [$
$S \to SS$	$T \to SC$
$S \to OT$	$C \to]$

Chomsky normal form: all productions are either $S \to AB$ or $S \to c$.

Context-free recognition problem

Input: CFG G (of constant size).

Offline

Given a string w, determine if $w \in L(G)$.

Online

String w is revealed one character at a time.
For every $t = 1, \ldots,$ after seing $w[t]$, determine if $w[1..t] \in L(G)$.

Context-free recognition problem

Input: CFG G (of constant size).

Offline

Given a string w, determine if $w \in L(G)$.

Online

String w is revealed one character at a time.
For every $t = 1, \ldots,$ after seing $w[t]$, determine if $w[1..t] \in L(G)$.

Context-free recognition problem

Input: CFG G (of constant size).

Offline

Given a string w, determine if $w \in L(G)$.

Online

String w is revealed one character at a time.
For every $t = 1, \ldots$, after seing $w[t]$, determine if $w[1..t] \in L(G)$.

History of online context-free recognition

Year	Authors	Runtime
1961	Cocke,Younger and Kasami (CYK)	$\mathcal{O}(n^3)$
1980	Graham, Harrison and Ruzzo	$\mathcal{O}(n^3/\log n)$
1995	Rytter	$\mathcal{O}(n^3/\log^2 n)$
2002	Lee	no comb. $\mathcal{O}(gn^{3-\varepsilon})$ *
2015	Abboud, Backurs and V. Williams	no comb. $\mathcal{O}(n^{3-\varepsilon})$ *
2024	this work	$n^3/2^{\Omega(\sqrt{\log n})}$

Valiant 1975, Rytter 1995

Offline context-free recognition in $\mathcal{O}(n^\omega)$ time.

* - holds also for the offline variant

History of online context-free recognition

Year	Authors	Runtime
1961	Cocke,Younger and Kasami (CYK)	$\mathcal{O}(n^3)$
1980	Graham, Harrison and Ruzzo	$\mathcal{O}(n^3/\log n)$
1995	Rytter	$\mathcal{O}(n^3/\log^2 n)$
2002	Lee	no comb. $\mathcal{O}(gn^{3-\varepsilon})$ *
2015	Abboud, Backurs and V. Williams	no comb. $\mathcal{O}(n^{3-\varepsilon})$ *
2024	this work	$n^3/2^{\Omega(\sqrt{\log n})}$

Valiant 1975, Rytter 1995

Offline context-free recognition in $\mathcal{O}(n^\omega)$ time.

* - holds also for the offline variant

History of online context-free recognition

Year	Authors	Runtime
1961	Cocke,Younger and Kasami (CYK)	$\mathcal{O}(n^3)$
1980	Graham, Harrison and Ruzzo	$\mathcal{O}(n^3/\log n)$
1995	Rytter	$\mathcal{O}(n^3/\log^2 n)$
2002	Lee	no comb. $\mathcal{O}(gn^{3-\varepsilon})$ *
2015	Abboud, Backurs and V. Williams	no comb. $\mathcal{O}(n^{3-\varepsilon})$ *
2024	this work	$n^3/2^{\Omega(\sqrt{\log n})}$

Valiant 1975, Rytter 1995

Offline context-free recognition in $\mathcal{O}(n^\omega)$ time.

* - holds also for the offline variant

History of online context-free recognition

Year	Authors	Runtime
1961	Cocke,Younger and Kasami (CYK)	$\mathcal{O}(n^3)$
1980	Graham, Harrison and Ruzzo	$\mathcal{O}(n^3/\log n)$
1995	Rytter	$\mathcal{O}(n^3/\log^2 n)$
2002	Lee	no comb. $\mathcal{O}(gn^{3-\varepsilon})$ *
2015	Abboud, Backurs and V. Williams	no comb. $\mathcal{O}(n^{3-\varepsilon})$ *
2024	this work	$n^3/2^{\Omega(\sqrt{\log n})}$

Valiant 1975, Rytter 1995

Offline context-free recognition in $\mathcal{O}(n^\omega)$ time.

* - holds also for the offline variant

CYK algorithm

$\mathcal{O}(n^3 g)$ dynamic approach based on:

$$\begin{cases} A \xrightarrow{\star} w[i..k] \\ B \xrightarrow{\star} w[k+1..j] \\ (C \to AB) \in P \end{cases} \implies C \xrightarrow{\star} w[i..j]$$

```
for j = 1.. do
    for (C → w[j]) ∈ P do
        D^C[j, j] := true
    for i = (j − 1)..1 do
        for k = i..(j − 1) do
            for (C → AB) ∈ P do
                if D^A[i, k] ∧ D^B[k + 1, j] then
                    D^C[i, j] := true
```

Works also for the online case!

CYK algorithm

$\mathcal{O}(n^3 g)$ dynamic approach based on:

$$\begin{cases} A \xrightarrow{\star} w[i..k] \\ B \xrightarrow{\star} w[k+1..j] \\ (C \to AB) \in P \end{cases} \implies C \xrightarrow{\star} w[i..j]$$

```
for j = 1.. do
    for (C → w[j]) ∈ P do
        D^C[j, j] := true
    for i = (j − 1)..1 do
        for k = i..(j − 1) do
            for (C → AB) ∈ P do
                if D^A[i, k] ∧ D^B[k + 1, j] then
                    D^C[i, j] := true
```

Works also for the online case!

CYK algorithm

$\mathcal{O}(n^3 g)$ dynamic approach based on:

$$\begin{cases} A \xrightarrow{\star} w[i..k] \\ B \xrightarrow{\star} w[k+1..j] \\ (C \to AB) \in P \end{cases} \implies C \xrightarrow{\star} w[i..j]$$

---

for $j = 1..$ **do**
  for $(C \to w[j]) \in P$ **do**
    $D^C[j,j] :=$ *true*
  for $i = (j-1)..1$ **do**
    for $k = i..(j-1)$ **do**
      for $(C \to AB) \in P$ **do**
        if $D^A[i,k] \wedge D^B[k+1,j]$ **then**
          $D^C[i,j] :=$ *true*

Works also for the online case!

Valiant's approach

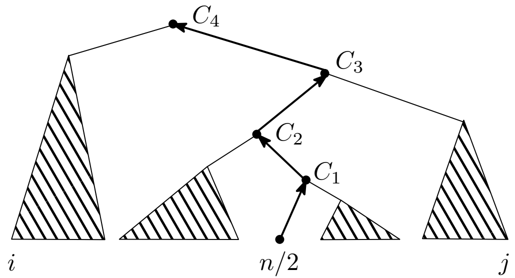
Calculate DP recursively for $w[1..n/2]$ and $w[(n/2+1)..n]$ and merge the results: need to process all substrings that contain $w[n/2]$.

Difficulty: we can extend the infix in both directions.

Valiant's approach

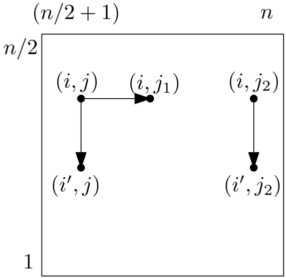
Calculate DP recursively for $w[1..n/2]$ and $w[(n/2+1)..n]$ and merge the results: need to process all substrings that contain $w[n/2]$.

Difficulty: we can extend the infix in both directions.



Valiant's idea (Rytter's presentation)

Create a graph in which a node (i, j) stores all non-terminals producing $w[i..j]$, for $i \leq n/2 < j$ and e.g. edges "down" correspond to extending a word at the beginning (to the left).



Observation: moving from (i, j) to (i', j) does not depend on j!

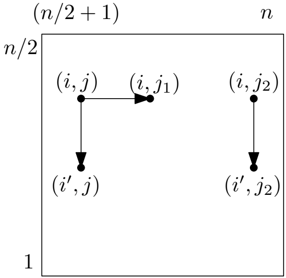
We only need to know the non-terminal producing $w[i..j]$.

Valiant 1975

We can calculate the transitive closure of the graph in $\mathcal{O}(n^{\omega})$ time.

Valiant's idea (Rytter's presentation)

Create a graph in which a node (i, j) stores all non-terminals producing $w[i..j]$, for $i \leq n/2 < j$ and e.g. edges "down" correspond to extending a word at the beginning (to the left).



Observation: moving from (i, j) to (i', j) does not depend on j!

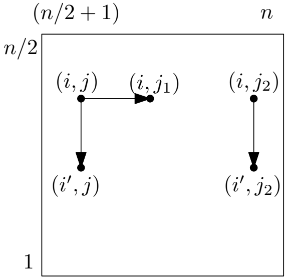
We only need to know the non-terminal producing $w[i..j]$.

Valiant 1975

We can calculate the transitive closure of the graph in $\mathcal{O}(n^{\omega})$ time.

Valiant's idea (Rytter's presentation)

Create a graph in which a node (i, j) stores all non-terminals producing $w[i..j]$, for $i \leq n/2 < j$ and e.g. edges "down" correspond to extending a word at the beginning (to the left).



Observation: moving from (i, j) to (i', j) does not depend on j!

We only need to know the non-terminal producing $w[i..j]$.

Valiant 1975

We can calculate the transitive closure of the graph in $\mathcal{O}(n^\omega)$ time.

Why matrix multiplication?

Observation

Test if we can extend (i, j) to (i', j) based only on the non-terminal producing $w[i..j]$, independently on j.

Jumps "to the left":

$$V[i', i]^{X,Y} = 1 \iff \exists_{Z \in V_N}\Big((X \to ZY) \in P \land Z \xrightarrow{\star} w[i'..i-1]\Big)$$

(*the infix starting at i and produced by Y can be extended to an infix starting at i′ and produced by X*)

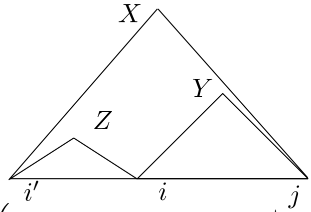
Multiple extensions in one direction: matrix multiplications!

Why matrix multiplication?

Observation

Test if we can extend (i, j) to (i', j) based only on the non-terminal producing $w[i..j]$, independently on j.

Jumps "to the left":



$$V[i', i]^{X,Y} = 1 \iff \exists_{Z \in V_N}\Big((X \to ZY) \in P \wedge Z \xrightarrow{\star} w[i'..i-1]\Big)$$

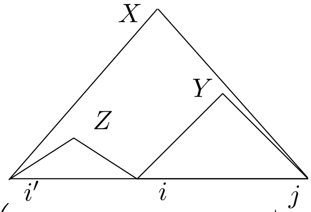
(*the infix starting at i and produced by Y can be extended to an infix starting at i′ and produced by X*)

Multiple extensions in one direction: matrix multiplications!

Why matrix multiplication?

Observation

Test if we can extend (i, j) to (i', j) based only on the non-terminal producing $w[i..j]$, independently on j.



Jumps "to the left":

$$V[i', i]^{X,Y} = 1 \iff \exists_{Z \in V_N} \Big((X \to ZY) \in P \land Z \xrightarrow{\star} w[i'..i-1] \Big)$$

(*the infix starting at i and produced by Y can be extended to an infix starting at i' and produced by X*)

Multiple extensions in one direction: matrix multiplications!

Online variant

New character $\approx$ new column in the considered matrix/graph, so:

Online Matrix-Vector Multiplication (OMv)

Given a matrix $M \in \{0,1\}^{n\times n}$, and a sequence of vectors $v_1, \ldots, v_n \in \{0,1\}^n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \ldots, n-1$.

Larsen, Williams [SODA 2017]

OMv can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time (w.h.p.).

OMv Hypothesis by Henzinger et al. [STOC 2015]

Every (randomized) algorithm solving OMv must take total time $n^{3-o(1)}$.

Online variant

New character $\approx$ new column in the considered matrix/graph, so:

Online Matrix-Vector Multiplication (OMv)

Given a matrix $M \in \{0,1\}^{n \times n}$, and a sequence of vectors $v_1, \ldots, v_n \in \{0,1\}^n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \ldots, n-1$.

Larsen, Williams [SODA 2017]

OMv can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time (w.h.p.).

OMv Hypothesis by Henzinger et al. [STOC 2015]

Every (randomized) algorithm solving OMv must take total time $n^{3-o(1)}$.

Online variant

New character $\approx$ new column in the considered matrix/graph, so:

Online Matrix-Vector Multiplication (OMv)

Given a matrix $M \in \{0,1\}^{n \times n}$, and a sequence of vectors $v_1, \ldots, v_n \in \{0,1\}^n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \ldots, n-1$.

Larsen, Williams [SODA 2017]

OMv can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time (w.h.p.).

OMv Hypothesis by Henzinger et al. [STOC 2015]

Every (randomized) algorithm solving OMv must take total time $n^{3-o(1)}$.

Online variant

New character $\approx$ new column in the considered matrix/graph, so:

Online Matrix-Vector Multiplication (OMv)

Given a matrix $M \in \{0,1\}^{n \times n}$, and a sequence of vectors $v_1, \ldots, v_n \in \{0,1\}^n$, the task is to output Mv_i before seeing v_{i+1}, for all $i = 1, \ldots, n-1$.

Larsen, Williams [SODA 2017]

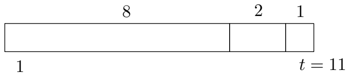
OMv can be solved in $n^3/2^{\Omega(\sqrt{\log n})}$ time (w.h.p.).

OMv Hypothesis by Henzinger et al. [STOC 2015]

Every (randomized) algorithm solving OMv must take total time $n^{3-o(1)}$.

Our approach

Maintain a division of the current prefix into powers of 2:



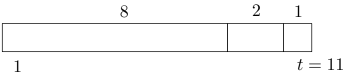
For every interval $\mathcal{I} = [p, p+s)$ create a process that:

- runs for s queries $Q_{p+s}, Q_{p+s+1}, \ldots, Q_{p+2s-1}$ where $Q_t = \{(i, E) : E \overset{\star}{\to} w[i..t], i \in [p+s..t]\}$
- processes all infixes of w that start in $\mathcal{I}$ and end in t

  (formally it computes: $A_t = \{(i, E) : E \overset{\star}{\to} w[i..t], i \in \mathcal{I}\}$)
- is updated after every query ($w[t]$ becomes part of the string)
- takes total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time.

Our approach

Maintain a division of the current prefix into powers of 2:



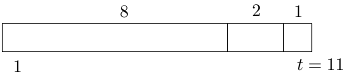
For every interval $\mathcal{I} = [p, p+s)$ create a process that:

- runs for s queries $Q_{p+s}, Q_{p+s+1}, \ldots, Q_{p+2s-1}$ where $Q_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [p+s..t]\}$
- processes all infixes of w that start in $\mathcal{I}$ and end in t

  (formally it computes: $A_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in \mathcal{I}\}$)
- is updated after every query ($w[t]$ becomes part of the string)
- takes total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time.

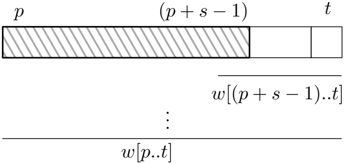
Our approach

Maintain a division of the current prefix into powers of 2:



For every interval $\mathcal{I} = [p, p+s)$ create a process that:

- runs for s queries $Q_{p+s}, Q_{p+s+1}, \ldots, Q_{p+2s-1}$ where $Q_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [p+s..t]\}$
- processes all infixes of w that start in $\mathcal{I}$ and end in t

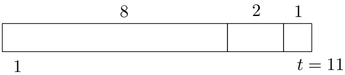


(formally it computes: $A_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in \mathcal{I}\}$)

- is updated after every query ($w[t]$ becomes part of the string)
- takes total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time.

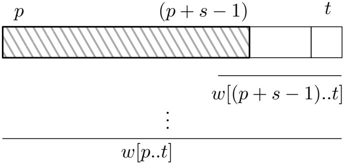
Our approach

Maintain a division of the current prefix into powers of 2:



For every interval $\mathcal{I} = [p, p+s)$ create a process that:

- runs for s queries $Q_{p+s}, Q_{p+s+1}, \ldots, Q_{p+2s-1}$ where $Q_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [p+s..t]\}$
- processes all infixes of w that start in $\mathcal{I}$ and end in t

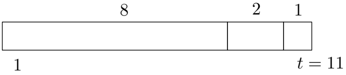


(formally it computes: $A_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in \mathcal{I}\}$)

- is updated after every query ($w[t]$ becomes part of the string)
- takes total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time.

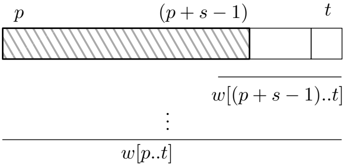
Our approach

Maintain a division of the current prefix into powers of 2:



For every interval $\mathcal{I} = [p, p+s)$ create a process that:

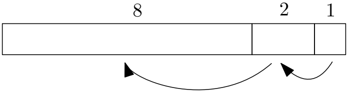
- runs for s queries $Q_{p+s}, Q_{p+s+1}, \ldots, Q_{p+2s-1}$ where $Q_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [p+s..t]\}$
- processes all infixes of w that start in $\mathcal{I}$ and end in t



(formally it computes: $A_t = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in \mathcal{I}\}$)

- is updated after every query ($w[t]$ becomes part of the string)
- takes total $s^3/2^{\Omega(\sqrt{\log s})}$ randomized time.

Algorithm



While processing the t-th character, the j-th process (for $[b_j, e_j)$):

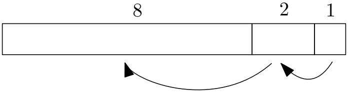
- gets: $Q_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [e_j..t]\}$
- computes: $A_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [b_j..e_j)\}$

Note: $Q_t^j = A_t^{j+1} \cup A_t^{j+2} \ldots$

Process of length 2^k is created every $n/2^k$ queries and runs for at most 2^k queries, so the total running time is :

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \cdot \left(2^k\right)^3 / 2^{\Omega(\sqrt{k})} = n \cdot \sum_{k=0}^{\log n} 2^{2k - \Omega(\sqrt{k})} = n^3 / 2^{\Omega(\sqrt{\log n})}.$$

Algorithm



While processing the t-th character, the j-th process (for $[b_j, e_j)$):

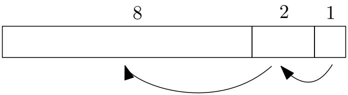
- gets: $Q_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [e_j..t]\}$
- computes: $A_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [b_j..e_j)\}$

Note: $Q_t^j = A_t^{j+1} \cup A_t^{j+2} \ldots$

Process of length 2^k is created every $n/2^k$ queries and runs for at most 2^k queries, so the total running time is :

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \cdot \left(2^k\right)^3 / 2^{\Omega(\sqrt{k})} = n \cdot \sum_{k=0}^{\log n} 2^{2k - \Omega(\sqrt{k})} = n^3 / 2^{\Omega(\sqrt{\log n})}.$$

Algorithm



While processing the t-th character, the j-th process (for $[b_j, e_j)$):

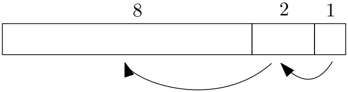
- gets: $Q_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [e_j..t]\}$
- computes: $A_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [b_j..e_j)\}$

Note: $Q_t^j = A_t^{j+1} \cup A_t^{j+2} ...$

Process of length 2^k is created every $n/2^k$ queries and runs for at most 2^k queries, so the total running time is :

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \cdot \left(2^k\right)^3 / 2^{\Omega(\sqrt{k})} = n \cdot \sum_{k=0}^{\log n} 2^{2k-\Omega(\sqrt{k})} = n^3 / 2^{\Omega(\sqrt{\log n})}.$$

Algorithm



While processing the t-th character, the j-th process (for $[b_j, e_j)$):

- gets: $Q_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [e_j..t]\}$
- computes: $A_t^j = \{(i, E) : E \xrightarrow{\star} w[i..t], i \in [b_j..e_j)\}$

Note: $Q_t^j = A_t^{j+1} \cup A_t^{j+2} \ldots$

Process of length 2^k is created every $n/2^k$ queries and runs for at most 2^k queries, so the total running time is :

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \cdot \left(2^k\right)^3 / 2^{\Omega(\sqrt{k})} = n \cdot \sum_{k=0}^{\log n} 2^{2k-\Omega(\sqrt{k})} = n^3 / 2^{\Omega(\sqrt{\log n})}.$$

Additional tool

Doubly-dynamic OMv: in every query we get next column of the matrix and the vector to multiply with.

Theorem

We can process in $s^3/2^{\Omega(\sqrt{\log s})}$ time a sequence of s vectors $v_1, q_1, v_2, q_2, \ldots$ where $|v_i| = s, |q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}.

Sketch of the proof:

- divide the processed matrix of size $s \times i$ into submatrices of sizes that are (decreasing) powers of 2
- divide every $s \times 2^k$ submatrix into $s/2^k$ matrices of size $2^k \times 2^k$ and build OMv data structure on each of them.

Thank you!

Additional tool

Doubly-dynamic OMv: in every query we get next column of the matrix and the vector to multiply with.

Theorem

We can process in $s^3/2^{\Omega(\sqrt{\log s})}$ time a sequence of s vectors $v_1, q_1, v_2, q_2, \ldots$ where $|v_i| = s, |q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}.

Sketch of the proof:

- divide the processed matrix of size $s \times i$ into submatrices of sizes that are (decreasing) powers of 2
- divide every $s \times 2^k$ submatrix into $s/2^k$ matrices of size $2^k \times 2^k$ and build OMv data structure on each of them.

Thank you!

Additional tool

Doubly-dynamic OMv: in every query we get next column of the matrix and the vector to multiply with.

Theorem

We can process in $s^3/2^{\Omega(\sqrt{\log s})}$ time a sequence of s vectors $v_1, q_1, v_2, q_2, \ldots$ where $|v_i| = s, |q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}.

Sketch of the proof:

- divide the processed matrix of size $s \times i$ into submatrices of sizes that are (decreasing) powers of 2
- divide every $s \times 2^k$ submatrix into $s/2^k$ matrices of size $2^k \times 2^k$ and build OMv data structure on each of them.

Thank you!

Additional tool

Doubly-dynamic OMv: in every query we get next column of the matrix and the vector to multiply with.

Theorem

We can process in $s^3/2^{\Omega(\sqrt{\log s})}$ time a sequence of s vectors $v_1, q_1, v_2, q_2, \ldots$ where $|v_i| = s, |q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}.

Sketch of the proof:

- divide the processed matrix of size $s \times i$ into submatrices of sizes that are (decreasing) powers of 2
- divide every $s \times 2^k$ submatrix into $s/2^k$ matrices of size $2^k \times 2^k$ and build OMv data structure on each of them.

Thank you!

Additional tool

Doubly-dynamic OMv: in every query we get next column of the matrix and the vector to multiply with.

Theorem

We can process in $s^3/2^{\Omega(\sqrt{\log s})}$ time a sequence of s vectors $v_1, q_1, v_2, q_2, \ldots$ where $|v_i| = s, |q_i| = i$ in which we need to calculate $(v_1, \ldots, v_i) \times q_i$ online, before seeing v_{i+1}.

Sketch of the proof:

- divide the processed matrix of size $s \times i$ into submatrices of sizes that are (decreasing) powers of 2
- divide every $s \times 2^k$ submatrix into $s/2^k$ matrices of size $2^k \times 2^k$ and build OMv data structure on each of them.

Thank you!