dynamic programming

1 Recursive Techniques in Efficient Algorithms

- divide and conquer: get solution for problem size n from solutions of *disjoint* subproblems of size $< n$

$$T(n) = n + 2T(n/2)$$

- telescoping: get solution for problem size n from solutions of *one* subproblem of size $< n$

$$C(n) = n + C(n/2)$$

- *dynamic programming*: get solution for problem size n from solutions of many em not necessarily disjoint subproblems of size $< n$. Reasonably efficient if total number of subproblems is not too large.

2 Multiplication of many matrices

$$M = M_1 \times \ldots M_n$$

- dimension of matrices: each M_i is $r_{i-1} \times r_i$ matrix.
- unit of cost: multiply and accumulate.
- scalar product of length n costs n
- multiplying $a \times b$ matrix with $b \times c$ matrix: $b \cdot (a \cdot c)$.

2 Multiplication of many matrices

$$M = M_1 \times \ldots M_n$$

- dimension of matrices: each M_i is $r_{i-1} \times r_i$ matrix.
- unit of cost: multiply and accumulate.
- scalar product of length n costs n
- multiplying $a \times b$ matrix with $b \times c$ matrix: $b \cdot (a \cdot c)$.

Assiciativity: many orders if evaluation posssible.

Example

$$n = 3 \ , r_0 = 10 \ , \ r_1 = 20 \ , \ r_= 50 \ , \ r_3 = 2$$

- $(M_1 \times M_2) \times M_3$:

$$10 \cdot 20 \cdot 50 + 10 \cdot 50 \cdot 2 = 10000 + 1000 = 11000$$

- $M_1 \times (M_2 \times M_3)$:

$$20 \cdot 50 \cdot 2 + 10 \cdot 20 \cdot 2 = 2000 + 400 = 2400$$

2 Multiplication of many matrices

$$M = M_1 \times \ldots M_n$$

- dimension of matrices: each M_i is $r_{i-1} \times r_i$ matrix.
- unit of cost: multiply and accumulate.
- scalar product of length n costs n
- multiplying $a \times b$ matrix with $b \times c$ matrix: $b \cdot (a \cdot c)$.

Assiciativity: many orders if evaluation posssible.

Example

$$n = 3\ , r_0 = 10\ ,\ r_1 = 20\ ,\ r_= 50\ ,\ r_3 = 2$$

- $(M_1 \times M_2) \times M_3$:

$$10 \cdot 20 \cdot 50 + 10 \cdot 50 \cdot 2 = 10000 + 1000 = 11000$$

- $M_1 \times (M_2 \times M_3)$:

$$20 \cdot 50 \cdot 2 + 10 \cdot 20 \cdot 2 = 2000 + 400 = 2400$$

Promlem:

- input: $n, r_0, \ldots r_n$
- find optimal order of evaluation! (there are terribly many)

2 Multiplication of many matrices

$$M = M_1 \times \dots M_n$$

- dimension of matrices: each M_i is $r_{i-1} \times r_i$ matrix.
- unit of cost: multiply and accumulate.
- scalar product of length n costs n
- multiplying $a \times b$ matrix with $b \times c$ matrix: $b \cdot (a \cdot c)$.

Assiciativity: many orders if evaluation posssible.

Example

$$n = 3 \ , r_0 = 10 \ , \ r_1 = 20 \ , \ r_= 50 \ , \ r_3 = 2$$

- $(M_1 \times M_2) \times M_3$:

$$10 \cdot 20 \cdot 50 + 10 \cdot 50 \cdot 2 = 10000 + 1000 = 11000$$

- $M_1 \times (M_2 \times M_3)$:

$$20 \cdot 50 \cdot 2 + 10 \cdot 20 \cdot 2 = 2000 + 400 = 2400$$

Promlem:

- input: $n, r_0, \dots r_n$
- find optimal order of evaluation! (there are terribly many)

crucial observation:
Let m_{ij} be the minimal cost for computing $M_i \times \dots \times M_j$. Then

$$m_{ij} = \begin{cases} 0 : i = j \\ \min\{m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j \ : \ i \leq k \leq j\} \end{cases}$$

minimizing over orders

$$(M_i \times \dots M_k) \times (M_{k+1} \times \dots \times M_j)$$

2 Multiplication of many matrices

$$M = M_1 \times \dots M_n$$

- dimension of matrices: each M_i is $r_{i-1} \times r_i$ matrix.
- unit of cost: multiply and accumulate.
- scalar product of length n costs n
- multiplying $a \times b$ matrix with $b \times c$ matrix: $b \cdot (a \cdot c)$.

Assiciativity: many orders if evaluation posssible.

Example

$$n = 3 \ , r_0 = 10 \ , \ r_1 = 20 \ , \ r_{=} 50 \ , \ r_3 = 2$$

- $(M_1 \times M_2) \times M_3$:

$$10 \cdot 20 \cdot 50 + 10 \cdot 50 \cdot 2 = 10000 + 1000 = 11000$$

- $M_1 \times (M_2 \times M_3)$:

$$20 \cdot 50 \cdot 2 + 10 \cdot 20 \cdot 2 = 2000 + 400 = 2400$$

Promlem:

- input: $n, r_0, \dots r_n$
- find optimal order of evaluation! (there are terribly many)

crucial observation:
Let m_{ij} be the minimal cost for computing $M_i \times \dots \times M_j$. Then

$$m_{ij} = \begin{cases} 0 : i = j \\ \min\{m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j \ : \ i \le k \le j\} \end{cases}$$

minimizing over orders

$$(M_i \times \dots M_k) \times (M_{k+1} \times \dots \times M_j)$$

algorithm

- compute $m_{i,j}$ in order of $j - i$. Record for each m_{ij} the optimal $k(i,j)$.
- table of n rows $x \in [0 : n-1]$. In row x store $m_{1,1+x}$, $m_{2,2+x}, \dots$
- computation for an entry in row $x+1$ accesses at most $2x$ entries in rows $y < x$. Time $O(n)$
- $O(n^2)$ table entries; total time $O(n^3)$

2 Multiplication of many matrices

Promlem:

- input: $n, r_0, \dots r_n$
- find optimal order of evaluation! (there are terribly many)

crucial observation:
Let m_{ij} be the minimal cost for computing $M_i \times \dots \times M_j$. Then

$$m_{ij} = \begin{cases} 0 : i = j \\ \min\{m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j \ : \ i \leq k \leq j\} \end{cases}$$

minimizing over orders

$$(M_i \times \dots M_k) \times (M_{k+1} \times \dots \times M_j)$$

algorithm

- compute $m_{i,j}$ in order of $j - i$. Record for each m_{ij} the optimal $k(i,j)$.
- table of n rows $x \in [0 : n-1]$. In row x store $m_{1,1+x}$, $m_{2,2+x}, \dots$
- computation for an entry in row $x+1$ accesses at most $2x$ entries in rows $y < x$. Time $O(n)$
- $O(n^2)$ table entries; total time $O(n^3)$

2 Multiplication of many matrices

Promlem:

- input: $n, r_0, \ldots r_n$
- find optimal order of evaluation! (there are terribly many)

crucial observation:
Let m_{ij} be the minimal cost for computing $M_i \times \ldots \times M_j$. Then

$$m_{ij} = \begin{cases} 0 : i = j \\ \min\{m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j \ : \ i \leq k \leq j\} \end{cases}$$

minimizing over orders

$$(M_i \times \ldots M_k) \times (M_{k+1} \times \ldots \times M_j)$$

algorithm

- compute $m_{i,j}$ in order of $j - i$. Record for each m_{ij} the optimal $k(i,j)$.
- table of n rows $x \in [0 : n-1]$. In row x store $m_{1,1+x}$, $m_{2,2+x}, \ldots$
- computation for an entry in row $x+1$ accesses at most $2x$ entries in rows $y < x$. Time $O(n)$
- $O(n^2)$ table entries; total time $O(n^3)$

example [AHU: Aho, Hopcroft, Ullman]

$$n = 4 \ , \ r_0, \ldots, r_4 = 10, 20, 50, 1, 100$$

$m_{11} = 0$	$m_{22} = 0$	$m_{33} = 0$	$m_{44} = 0$
$m_{12} = 10000$	$m_{23} = 1000$	$m_{34} = 5000$	
$m_{13} = 1200$	$m_{24} = 3000$		
$m_{14} = 2200$			

Table 1: Values m_{ij} are computed row by row

3 Younger's algorithm for context free language recognition

The dynamic programming algorithm, but amazingly enough missing both in [AHU] and [CLRS].

from I2OS slide set 2

A context-free grammar G consists of the following components:

- a finite set of symbols $G.T$, called the alphabet of *terminal* symbols,
- a finite set of symbols $G.N$, called the alphabet of *nonterminal* symbols,
- a *start symbol* $G.S \in G.N$, and
- a finite set $G.P \subset G.N \times (G.N \cup G.T)^*$ of *productions*.

Symbols are either terminal or nonterminal, never both:

$$G.T \cap G.N = \emptyset.$$

If (n, w) is a production of grammar G, i.e., $(n, w) \in G.P$, we say that the string w is *directly derived* in G from the nonterminal symbol n. As a shorthand, we write

$$n \to_G w.$$

If there are several strings $w^1, \ldots, w^s$ that are directly derived in G from n, we write

$$n \to_G w^1 \mid \ldots \mid w^s.$$

$$n \to_G \varepsilon,$$

allowed, but we avoid it

$$\begin{aligned} T &= G.T, \\ N &= G.N, \\ S &= G.S, \\ P &= G.P, \\ \to &= \to_G . \end{aligned}$$

if G is clear

$N \cap T = \emptyset$ *nonterminals, terminals*

$S \in N$ *start symbol*

$P \subset N \times (N \cup T)^*$ *productions*

$$
\begin{aligned}
T &= \{0, 1, X\}, \\
N &= \{V, B, \langle BS \rangle\}, \\
S &= V, \\
B &\rightarrow 0 \mid 1, \\
\langle BS \rangle &\rightarrow B \mid B\langle BS \rangle, \\
V &\rightarrow X\langle BS \rangle \mid 0 \mid 1.
\end{aligned}
$$

Definition

$$n \rightarrow_G^* y$$

Nonterminal n derives $y \in (T \cup N)^*$

- $$n \rightarrow_G^* n$$

- If
$$n \rightarrow_G^* umv \ , \ m \rightarrow w \in P$$
then
$$n \rightarrow^* uwv$$

- this is all

Example grammar

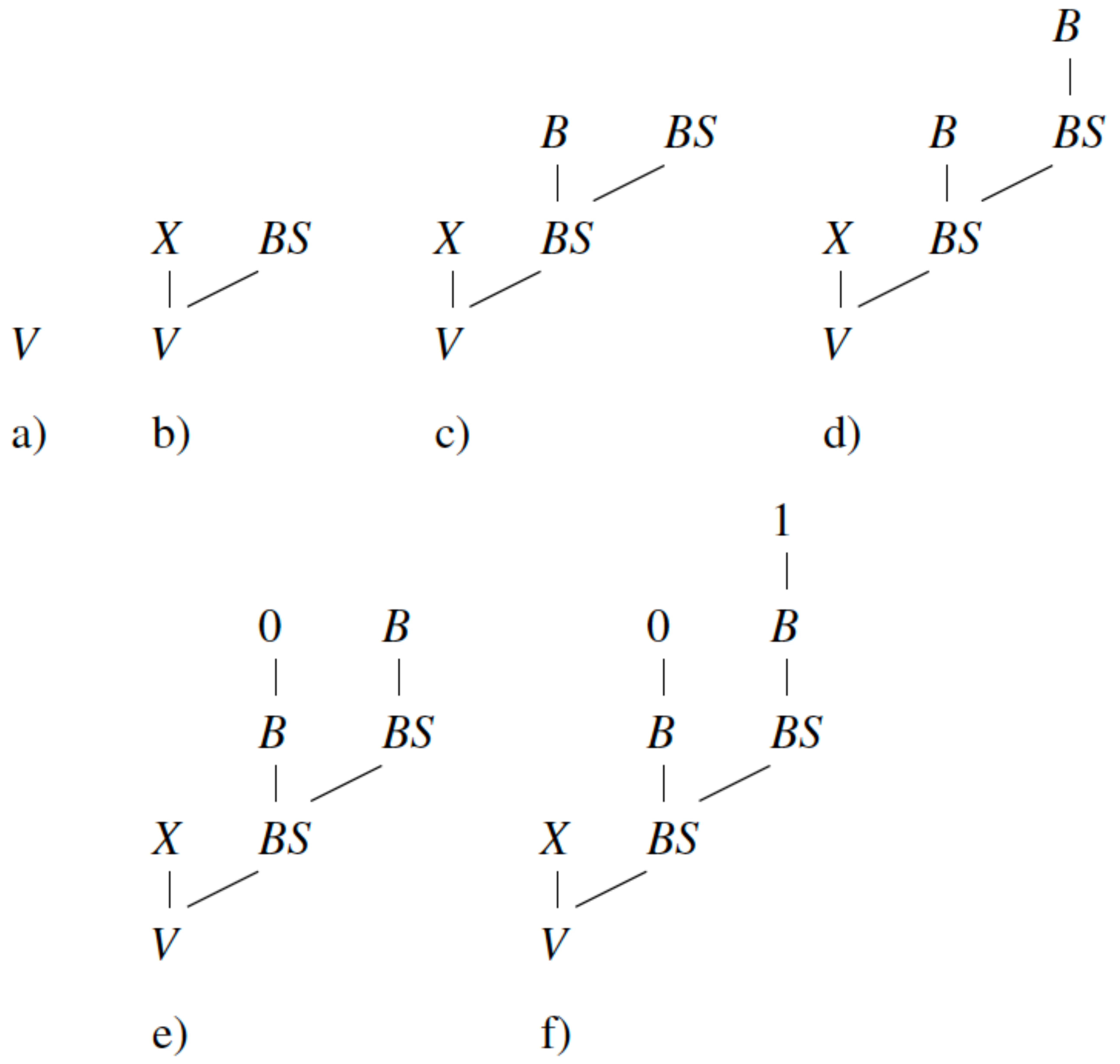
$$V \rightarrow^* X\langle BS\rangle \rightarrow^* XB\langle BS\rangle \rightarrow^* XBB \rightarrow^* X0B \rightarrow^* X01$$

$$
\begin{aligned}
T &= \{0, 1, X\}, \\
N &= \{V, B, \langle BS \rangle\}, \\
S &= V, \\
B &\to 0 \mid 1, \\
\langle BS \rangle &\to B \mid B\langle BS \rangle, \\
V &\to X\langle BS \rangle \mid 0 \mid 1.
\end{aligned}
$$

Example grammar

$$V \to^* X\langle BS \rangle \to^* XB\langle BS \rangle \to^* XBB \to^* X0B \to^* X01$$

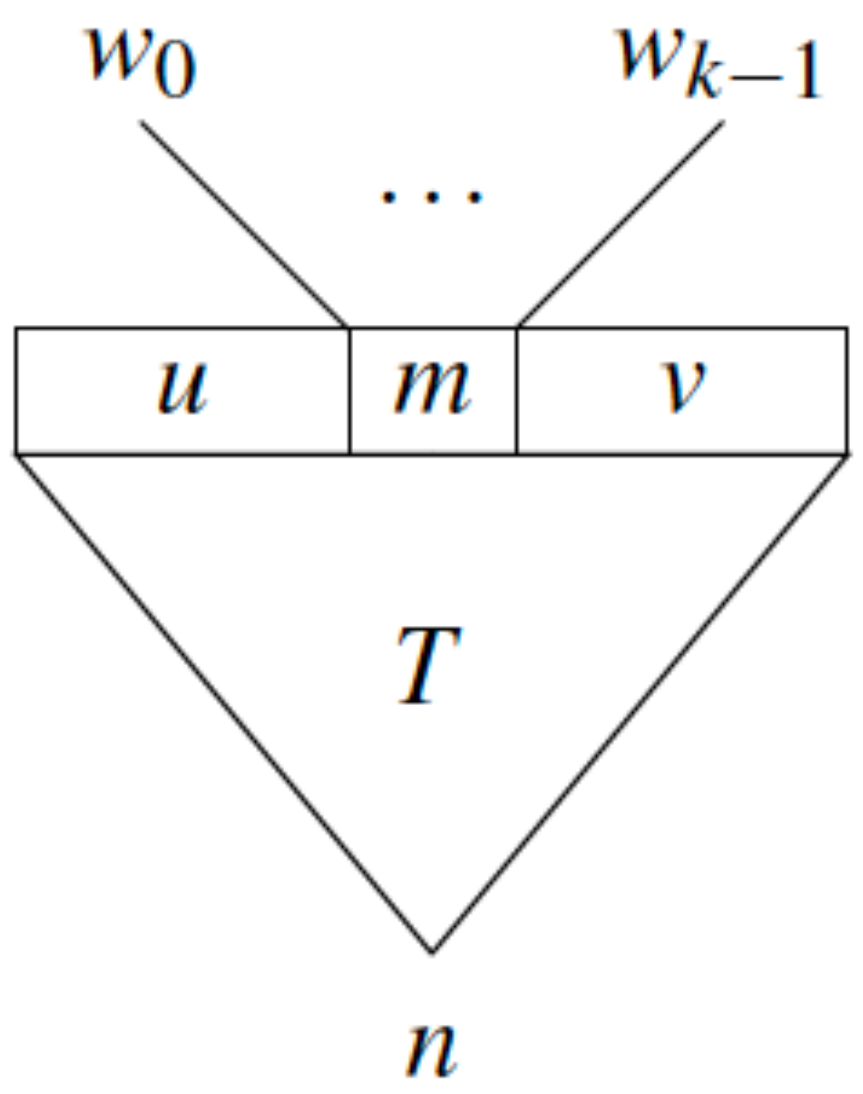
$$\begin{aligned} T &= \{0, 1, X\}, \\ N &= \{V, B, \langle BS \rangle\}, \\ S &= V, \\ B &\rightarrow 0 \mid 1, \\ \langle BS \rangle &\rightarrow B \mid B\langle BS \rangle, \\ V &\rightarrow X\langle BS \rangle \mid 0 \mid 1. \end{aligned}$$



derivation trees: example

def: derivation trees, quick and dirty

1. a single nonterminal n is a derivation tree with root n and border word n.
2. if T is a derivation tree with root n and border word umv and $m \to w \in P$ is a production with $w = w_0 \dots w_{k-1}$, then the tree illustrated in Fig. 71 is a derivation tree with root n and border word uwv.[1]
3. all derivation trees can be generated by finitely many applications of the above two rules.



3 Younger's algorithm for context free language recognition

The dynamic programming algorithm, but amazingly enough missing both in [AHU] and [CLRS].

reminder: context free grammars G

Components

- alphabet of terminals T
- alphabet of non termnals N
- startsymbos $s \in N$
- system of productions $P \subseteq N \times (N \cup T)^+$
- for $(n,w) \in P$ one writes $n \rightarrow w$.
- note: we excluded productions of the form $N \rightarrow \varepsilon$ (empty word). In standard definition allowed. Consequences only mildly interesting IMHO (in my humble opinion)

3 Younger's algorithm for context free language recognition

The dynamic programming algorithm, but amazingly enough missing both in [AHU] and [CLRS].

reminder: context free grammars G

Components

- alphabet of terminals T
- alphabet of non termnals N
- startsymbos $s \in N$
- system of productions $P \subseteq N \times (N \cup T)^+$
- for $(n, w) \in P$ one writes $n \rightarrow w$.
- note: we excluded productions of the form $N \rightarrow \varepsilon$ (empty word). In standard definition allowed. Consequences only mildly interesting IMHO (in my humble opinion)

For G we defined

- $r \rightarrow^* w$ iff there is a derivation trees for G with root r and border word w.

3 Younger's algorithm for context free language recognition

The dynamic programming algorithm, but amazingly enough missing both in [AHU] and [CLRS].

reminder: context free grammars G

Components

- alphabet of terminals T
- alphabet of non termnals N
- startsymbos $s \in N$
- system of productions $P \subseteq N \times (N \cup T)^+$
- for $(n,w) \in P$ one writes $n \rightarrow w$.
- note: we excluded productions of the form $N \rightarrow \varepsilon$ (empty word). In standard definition allowed. Consequences only mildly interesting IMHO (in my humble opinion)

For G we defined

- $r \rightarrow^* w$ iff there is a derivation trees for G with root r and border word w.
- language generated by G:

$$L(G) = \{w \in T^+ \ : S \rightarrow^* w\}$$

3 Younger's algorithm for context free language recognition

The dynamic programming algorithm, but amazingly enough missing both in [AHU] and [CLRS].

reminder: context free grammars G

Components

- alphabet of terminals T
- alphabet of non termnals N
- startsymbos $s \in N$
- system of productions $P \subseteq N \times (N \cup T)^+$
- for $(n,w) \in P$ one writes $n \rightarrow w$.
- note: we excluded productions of the form $N \rightarrow \varepsilon$ (empty word). In standard definition allowed. Consequences only mildly interesting IMHO (in my humble opinion)

For G we defined

- $r \rightarrow^* w$ iff there is a derivation trees for G with root r and border word w.

- language generated by G:

$$L(G) = \{w \in T^+ : S \rightarrow^* w\}$$

Fix G:

problem

- input $w \in T^+$
- decide if $w \in L(G)$
- trying all trees would be terribly slow.

3.1 Transform grammar to (almost) Chomsky normal form

Grammar has *normal form* if all productions have the form

- $n \to AB$ with $A, B \in N \cup T$
  i.e. right hand sides of productions have length 2
- except $S \to x$ with $a \in T$
  i.e. signle terminals $x \in L(G)$ are derived directly from S.

3.1 Transform grammar to (almost) Chomsky normal form

Grammar has *normal form* if all productions have the form

- $n \rightarrow AB$ with $A, B \in N \cup T$
  i.e. right hand sides of productions have length 2
- except $S \rightarrow x$ with $a \in T$
  i.e. signle terminals $x \in L(G)$ are derived directly from S.

Transforming G to normal form:

- eliminate right hand sides with more than 2 symbols: for each production

$$P: \quad n \rightarrow a_1 \ldots a_s \; s \geq 2$$

  introduce a new non terminal x and and replace p by

$$p \rightarrow a_1 \ldots a_{s-2}x \;,\; x \rightarrow a_{s-1}a_s$$

  Repeat until all right hand sides have length 2 or 1.

3.1 Transform grammar to (almost) Chomsky normal form

Grammar has *normal form* if all productions have the form

- $n \rightarrow AB$ with $A, B \in N \cup T$
  i.e. right hand sides of productions have length 2
- except $S \rightarrow x$ with $a \in T$
  i.e. signle terminals $x \in L(G)$ are derived directly from S.

Transforming G to normal form:

- eliminate right hand sides with more than 2 symbols: for each production

  $$P: \quad n \rightarrow a_1 \dots a_s \; s \geq 2$$

  introduce a new non terminal x and and replace p by

  $$p \rightarrow a_1 \dots a_{s-2}x \; , \; x \rightarrow a_{s-1}a_s$$

  Repeat until all right hand sides have length 2 or 1.

- for non terminals $A \in N$ define *chain*(A) as the set of $a \in N \cup T$ which can be derived from A ba a chain of productions

  $$A = N_0 \rightarrow N_1 \rightarrow \dots \rightarrow N_k = a \text{ with } N_i \in N \cup T \text{ for all } i$$

  Compute *chain*(A) for all $a \in N$ of the original grammar.

3.1 Transform grammar to (almost) Chomsky normal form

Grammar has *normal form* if all productions have the form

- $n \to AB$ with $A, B \in N \cup T$
  i.e. right hand sides of productions have length 2
- except $S \to x$ with $a \in T$
  i.e. signle terminals $x \in L(G)$ are derived directly from S.

Transforming G to normal form:

- eliminate right hand sides with more than 2 symbols: for each production

  $$P: \quad n \to a_1 \dots a_s \; s \geq 2$$

  introduce a new non terminal x and and replace p by

  $$p \to a_1 \dots a_{s-2}x \;,\; x \to a_{s-1}a_s$$

  Repeat until all right hand sides have length 2 or 1.

- for non terminals $A \in N$ define $chain(A)$ as the set of $a \in N \cup T$ which can be derived from A ba a chain of productions

  $$A = N_0 \to N_1 \to \dots \to N_k = a \text{ with } N_i \in N \cup T \text{ for all } i$$

  Compute $chain(A)$ for all $a \in N$ of the original grammar.

- for each $A \in N$ with $chain(A) \neq \emptyset$ add to productions of the form

  $$N \to AB$$

  the productions

  $$N \to XB \;,\; X \in chain(a)$$

  and to productions of the form

  $$N \to BA$$

  the productions

  $$N \to BX \;,\; X \in chain(A)$$

3.1 Transform grammar to (almost) Chomsky normal form

Grammar has *normal form* if all productions have the form

- $n \to AB$ with $A, B \in N \cup T$
  i.e. right hand sides of productions have length 2
- except $S \to x$ with $a \in T$
  i.e. signle terminals $x \in L(G)$ are derived directly from S.

Transforming G to normal form:

- eliminate right hand sides with more than 2 symbols: for each production

$$P: \quad n \to a_1 \ldots a_s \; s \geq 2$$

  introduce a new non terminal x and and replace p by

$$p \to a_1 \ldots a_{s-2} x \; , \; x \to a_{s-1} a_s$$

  Repeat until all right hand sides have length 2 or 1.

- for non terminals $A \in N$ define $chain(A)$ as the set of $a \in N \cup T$ which can be derived from A ba a chain of productions

$$A = N_0 \to N_1 \to \ldots \to N_k = a \text{ with } N_i \in N \cup T \text{ for all } i$$

  Compute $chain(A)$ for all $a \in N$ of the original grammar.

- for each $A \in N$ with $chain(A) \neq \emptyset$ add to productions of the form

$$N \to AB$$

  the productions

$$N \to XB \; , \; X \in chain(a)$$

  and to productions of the form

$$N \to BA$$

  the productions

$$N \to BX \; , \; X \in chain(A)$$

- for all $x \in chain(S) \cap T$ add

$$S \to x$$

3.2 Youngers algorithm

- assume G is in normal form.
- input $w = w_1 \dots w_n \in T^+$
- **crucial definition:** For $1 \leq i \leq n$ define the sets ot terminals

$$n_{i,i} = \{w_i\}$$

and for $1 \leq i < j \leq n$ define the *finite* sets of non terminals

$$n_{i,j} = \{A \in N \; : \; A \rightarrow^* w_i \dots w_j\}$$

3.2 Youngers algorithm

- assume G is in normal form.
- input $w = w_1 \dots w_n \in T^+$
- **crucial definition:** For $1 \leq i \leq n$ define the sets ot terminals

$$n_{i,i} = \{w_i\}$$

and for $1 \leq i < j \leq n$ define the *finite* sets of non terminals

$$n_{i,j} = \{A \in N \; : \; A \rightarrow^* w_i \dots w_j\}$$

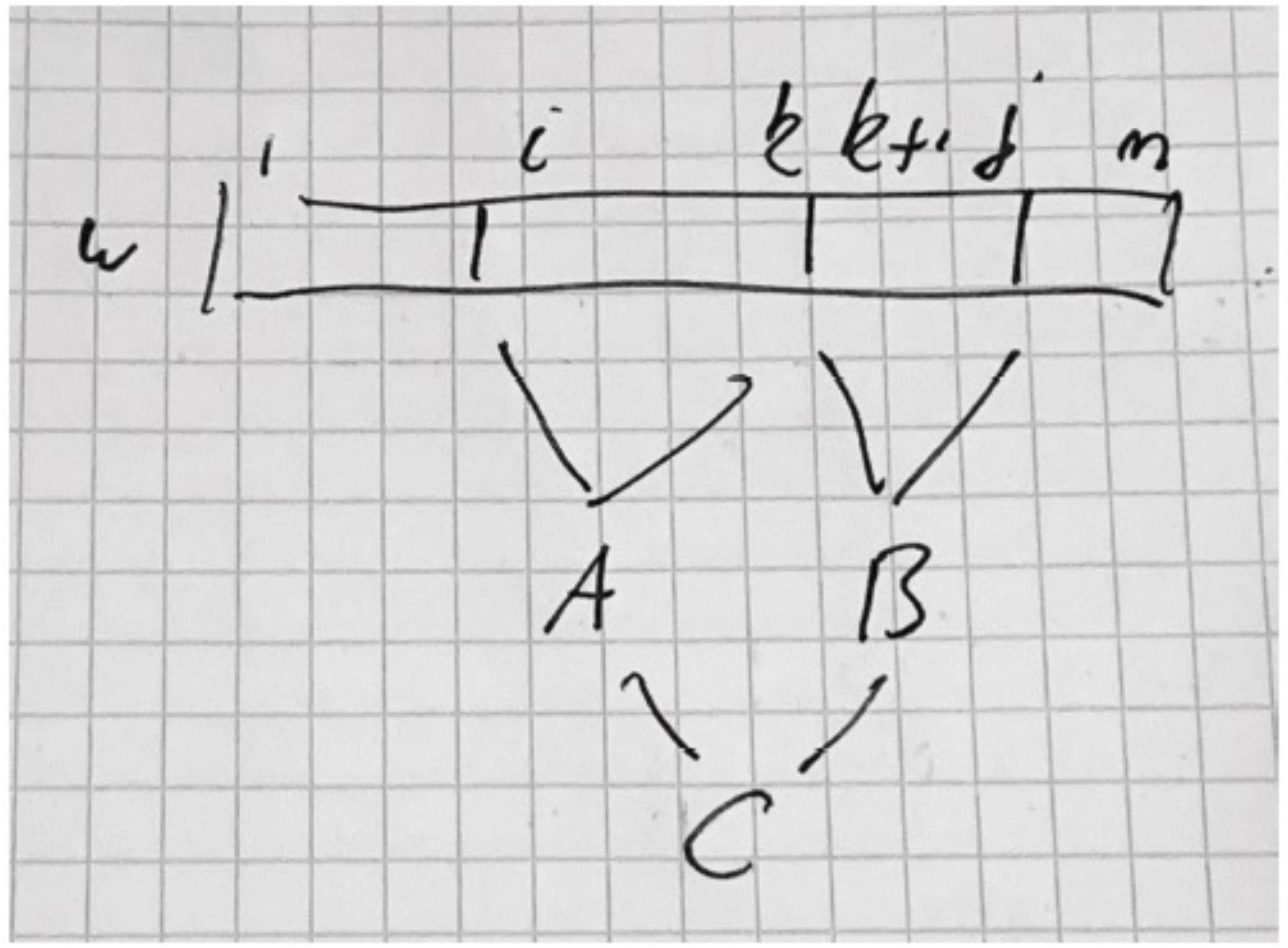


Figure 1: To subtree with border word $w[i : j]$ has a root C and the sons A and B of C are roots of subtrees with border words $w[i,k]$ and $w[k+1 : j]$ for some k.

- Compute these sets in order of increasing $j - i$.

$$n_{i,i+1} = \{A \in N \; : \; A \rightarrow w_i w_{i+1}\}$$

$$n_{i,j} = \bigcup_{i \leq k \leq j} \{C \in N \; : \; C \rightarrow AB \; , \; A \in n_{i,k} \; , B \in n_{k+1,j}\}$$

3.2 Youngers algorithm

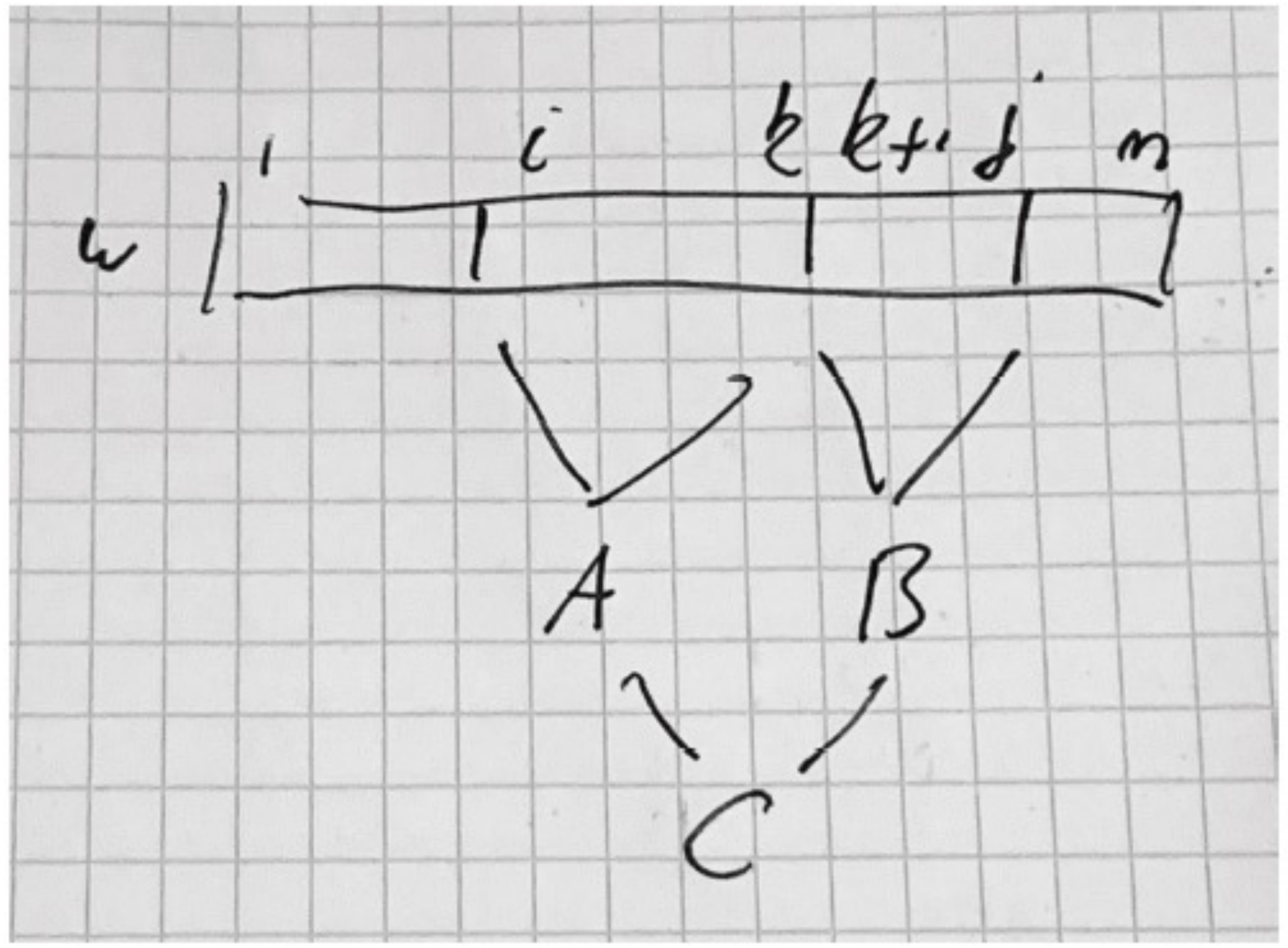


Figure 1: To subtree with border word $w[i:j]$ has a root C and the sons A and B of C are roots of subtrees with border words $w[i,k]$ and $w[k+1:j]$ for some k.

- assume G is in normal form.
- input $w = w_1 \dots w_n \in T^+$
- **crucial definition:** For $1 \leq i \leq n$ define the sets ot terminals

$$n_{i,i} = \{w_i\}$$

  and for $1 \leq i < j \leq n$ define the *finite* sets of non terminals

$$n_{i,j} = \{A \in N \ : \ A \rightarrow^* w_i \dots w_j\}$$

- Compute these sets in order of increasing $j-i$.

$$n_{i,i+1} = \{A \in N \ : \ A \rightarrow w_i w_{i+1}\}$$

$$n_{i,j} = \bigcup_{i \leq k \leq j} \{C \in N \ : \ C \rightarrow AB \ , \ A \in n_{i,k} \ , B \in n_{k+1,j}\}$$

For implementation in rows x with $n_{1,1+x}, n_{2,2+x}, \dots$

3.2 Youngers algorithm

- assume G is in normal form.
- input $w = w_1 \dots w_n \in T^+$
- **crucial definition:** For $1 \leq i \leq n$ define the sets ot terminals

$$n_{i,i} = \{w_i\}$$

and for $1 \leq i < j \leq n$ define the *finite* sets of non terminals

$$n_{i,j} = \{A \in N \;:\; A \rightarrow^* w_i \dots w_j\}$$

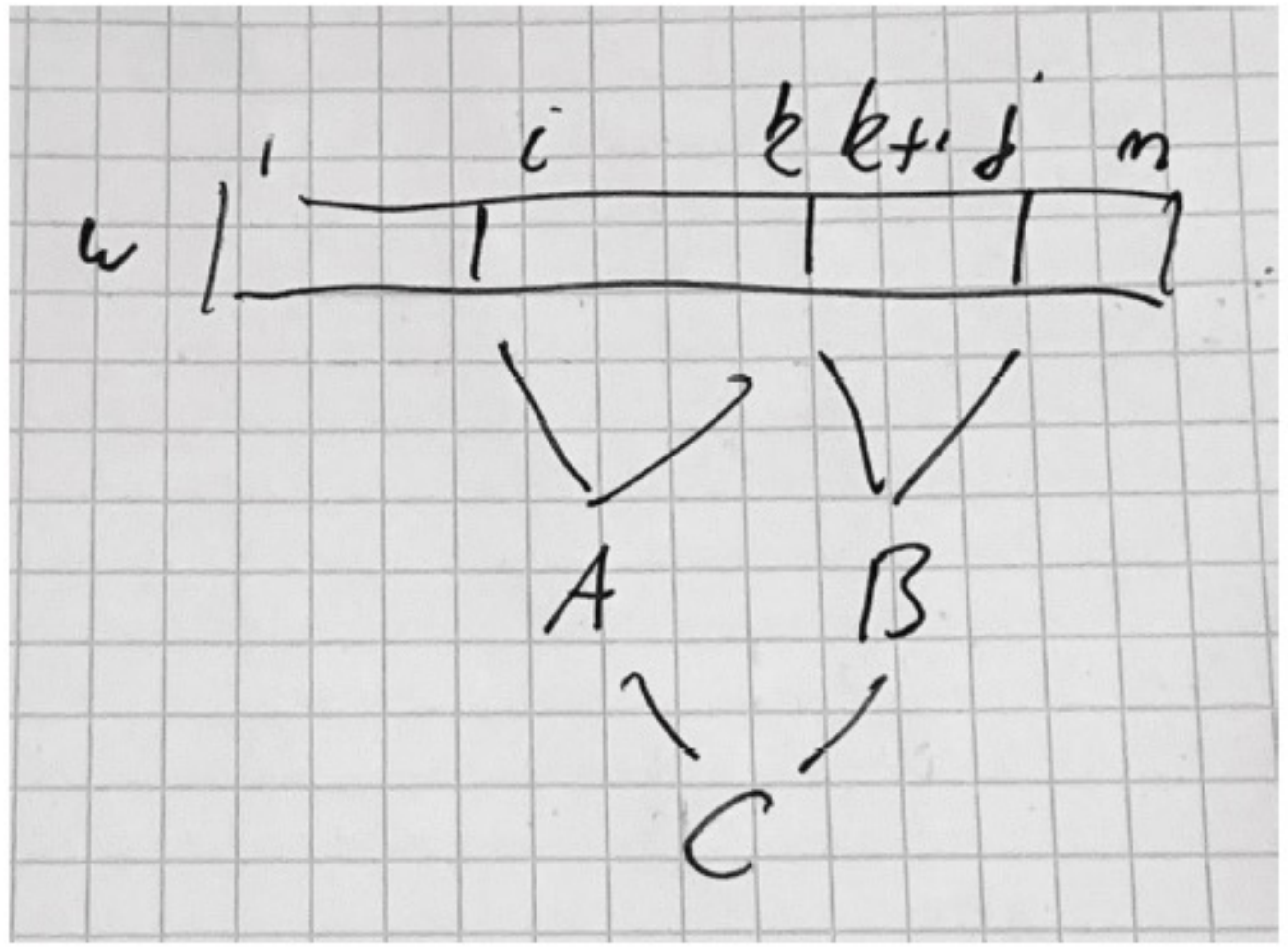


Figure 1: To subtree with border word $w[i:j]$ has a root C and the sons A and B of C are roots of subtrees with border words $w[i,k]$ and $w[k+1:j]$ for some k.

- Compute these sets in order of increasing $j - i$.

$$n_{i,i+1} = \{A \in N \;:\; A \rightarrow w_i w_{i+1}\}$$

$$n_{i,j} = \bigcup_{i \leq k \leq j} \{C \in N \;:\; C \rightarrow AB \;,\; A \in n_{i,k} \;, B \in n_{k+1,j}\}$$

For implementation in rows x with $n_{1,1+x}, n_{2,2+x}, \dots$

- time: for each i, j, k time $O(1)$ because sets $n_{i,j}$ are finite. There are $O(n^3)$ such triples.