

Algorithm Engineering

„Teilmengen-Suche“

Stefan Edelkamp

Motivation

- ▶ Speicherplatzsparende Wörterbücher
 - Programm/Modellprüfung (Model Checking)
- ▶ Subzeichenketten-Erkennung:
 - Editierdistanz-Problem, Approximative Zeichenkettensuche
 - MSA (Multiple Sequence Alignment) Problem
- ▶ Teilwort-Erkennung:
 - Kreuzwort-Problem: Eine Anfrage wie B*T**R im Kreuzworträtselproble mit BETTER, BITTER, BUTLER, oder BUTTER beantworten
 - Spieleprogrammierung

Übersicht

- ▶ Speicherplatzsparende Wörterbücher
 - Bit-State Hashing
 - Hash-Compaction
- ▶ Subzeichenketten-Erkennung:
 - Dynamische Programmierung
- ▶ Teilwort-Erkennung:
 - Listen und Array
 - Hashing und Tries
 - Unlimited Branching Trees

Übersicht

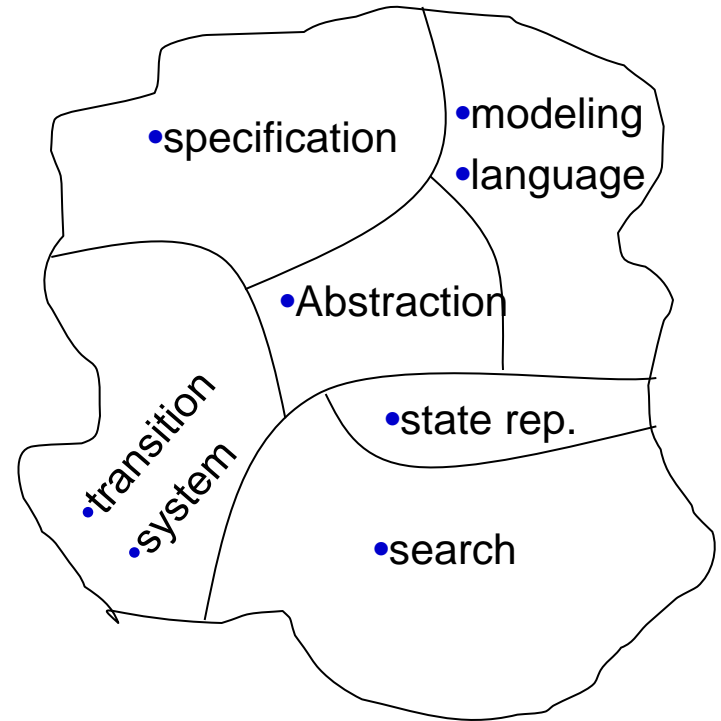
- ▶ Speicherplatzsparende Wörterbücher
 - Bit-State Hashing
 - Kollaps-Kompression
- ▶ Subzeichenketten-Erkennung:
 - Dynamische Programmierung
- ▶ Teilwort-Erkennung:
 - Listen und Array
 - Hashing und Tries
 - Unlimited Branching Trees

Model Checking

▶ Gegeben

- Eine Modell eines Systems.
- Die Spezifikation einer Eigenschaft

▶ **Problem** Erfüllt das System die Eigenschaft?

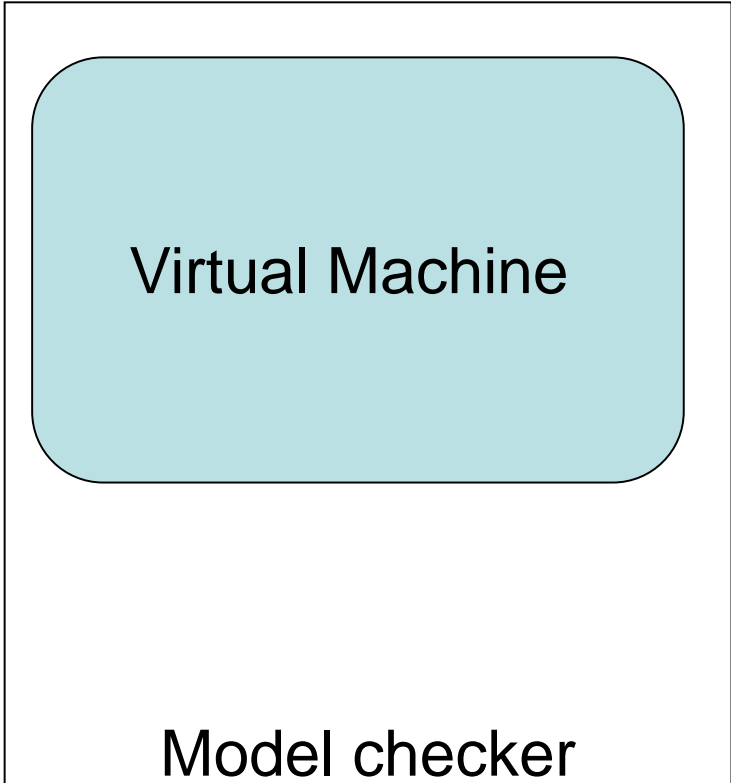


Beispiel: Prüfen eines C++ Programms

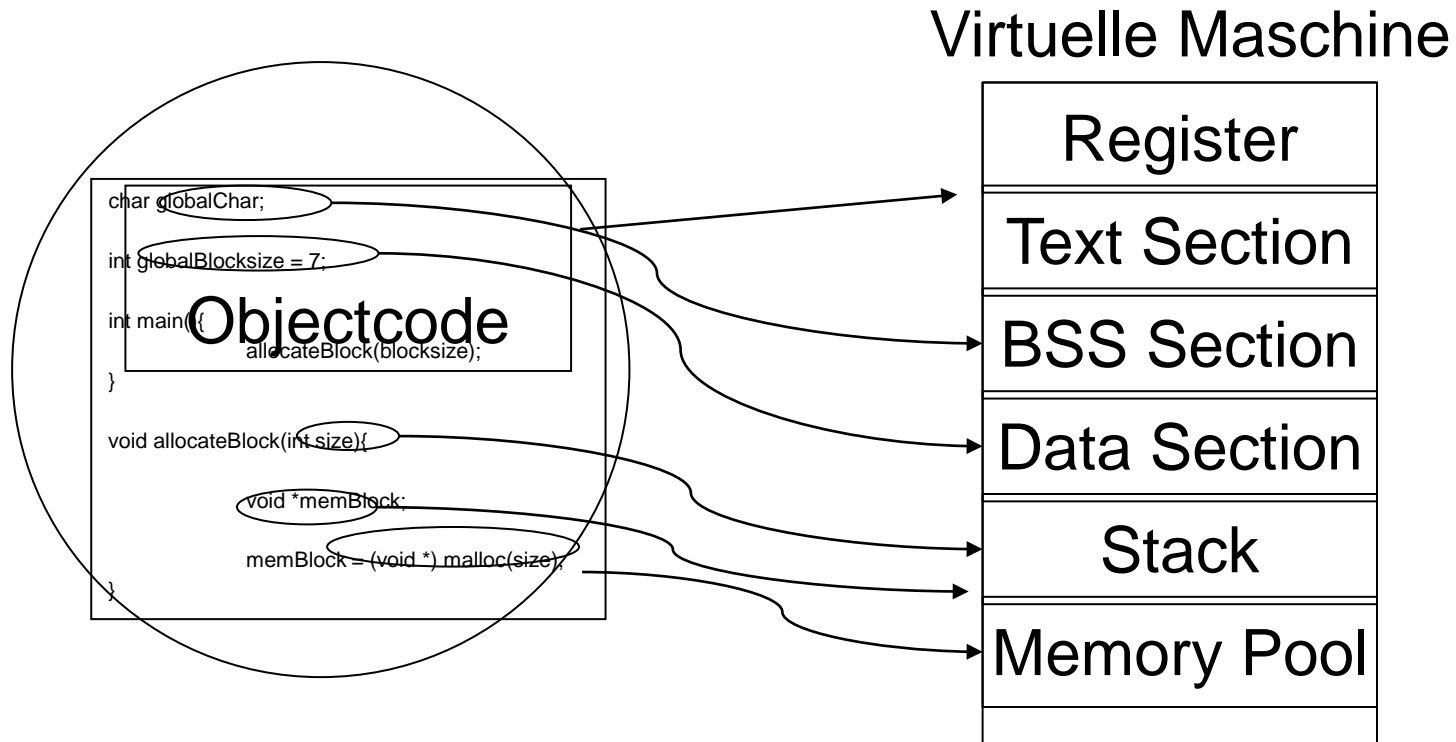
```
char globalChar;  
  
int globalBlocksize = 7;  
  
int main(){  
    allocateBlock(blocksize);  
}  
  
void allocateBlock(int size){  
    void *memBlock;  
    memBlock = (void *) malloc(size);  
}
```

Objectcode

igcc
Compiler

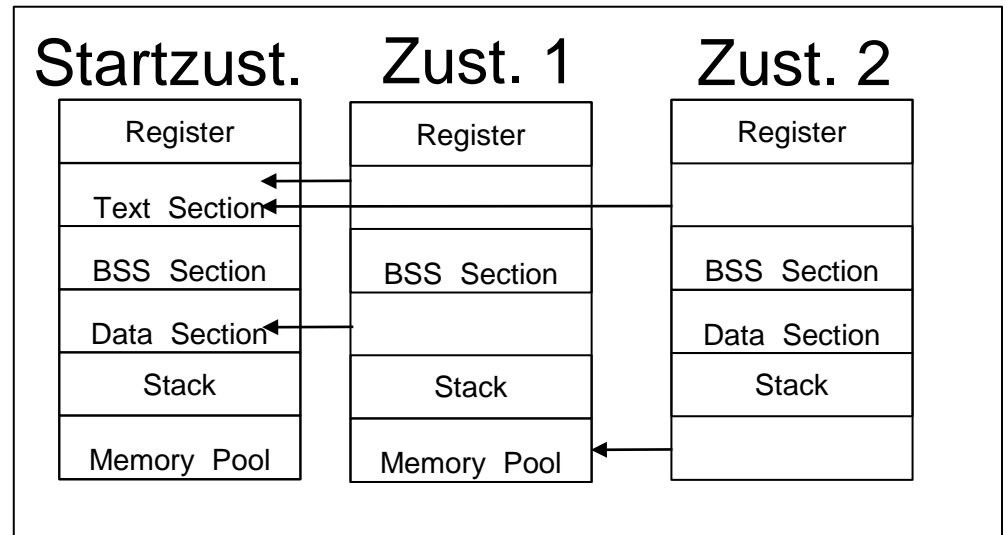
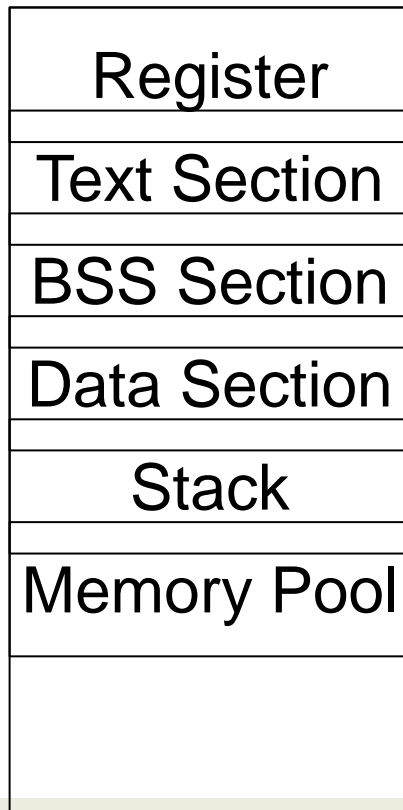


Interpretieren des Objectcodes

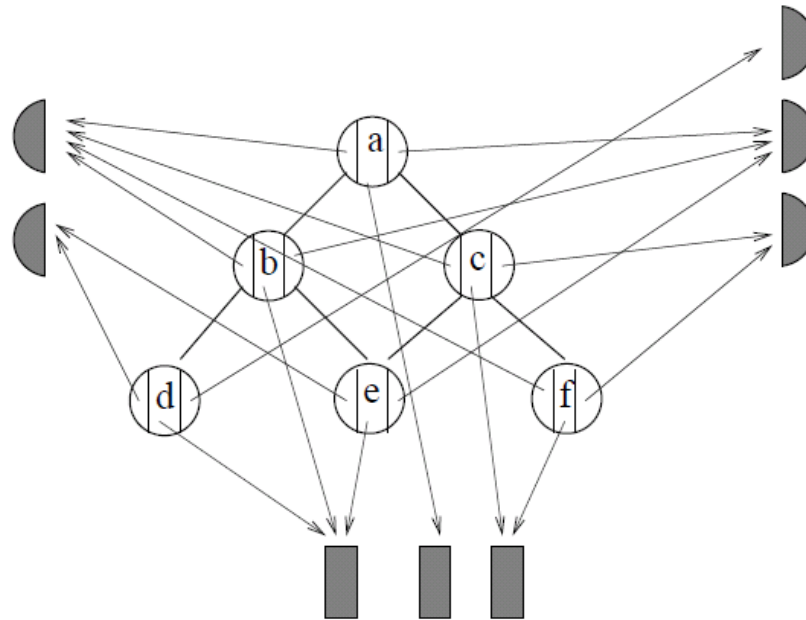


Generierung von Zuständen

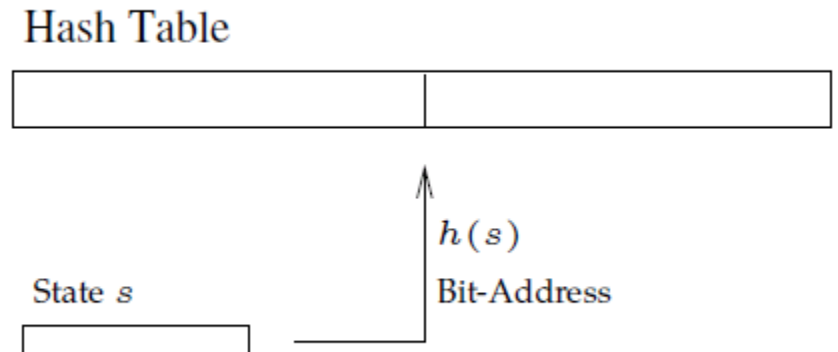
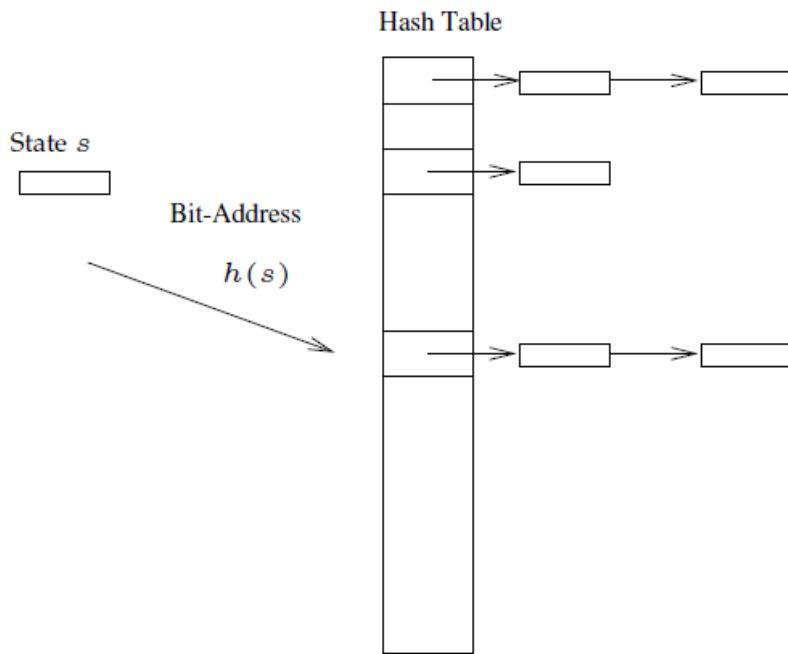
Virtuelle Maschine



„Kollaps“-Kompression für große Zustände



Fehlerhafte Wörterbücher sparen Platz



Fehlerwahrscheinlichkeit in Bit-State Hashing (Bloom Filter)

- ▶ n : Anzahl der Elemente
- ▶ m : Anzahl der Bits (Tabellengröße)
- ▶ i tes Element kollidiert mit $1 \dots i-1$
mit W'keit von $(i - 1)/m, 1 \leq i \leq n$.

- ▶ Wahrscheinlichkeit für einen fehlerhafte Suche

$$P_1 \leq \frac{1}{n} \sum_{i=0}^{n-1} \frac{i}{m} \leq n/2m.$$

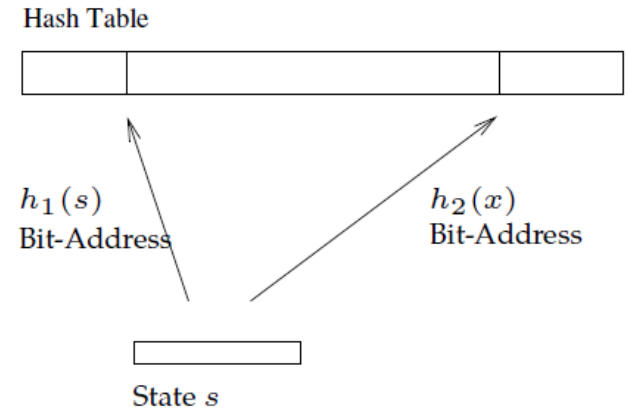
Senken der Fehlerrate

- ▶ h unabhängige Hashfunktionen, $hn \leq n$

$$P_h \leq \frac{1}{n} \sum_{i=0}^{n-1} \left(h \cdot \frac{i}{m} \right)^h$$

- ▶ Mit zwei Bits (Double Bit-State Hashing)

$$P_2 \leq \frac{1}{n} \left(\frac{2}{m} \right)^2 \sum_{i=0}^{n-1} i^2 = 2(n-1)(2n-1)/3m^2 \leq 4n^2/3m^2.$$



Übersicht

- ▶ Speicherplatzsparende Wörterbücher
 - Bit-State Hashing
 - Kollaps-Kompression
- ▶ **Subzeichenketten-Erkennung:**
 - **Dynamische Programmierung**
- ▶ Teilwort-Erkennung:
 - Listen und Array
 - Hashing und Tries
 - Unlimited Branching Trees

Editierdistanz-Problem

Berechne für zwei gegebene Zeichenfolgen A und B möglichst effizient die Editier-Distanz $D(A,B)$ und eine minimale Folge von Editieroperationen, die A in B überführt.

i n f - - - o r m a t i k -
 i n t e r p o l - a t i o n

	S	O	R	T
S	0	1	2	3
P	1	0	1	2
O	2	1	1	2
R	3	2	1	2
T	4	3	2	1

	S	O	R	T
S				
P				
O				
R				
T				

S _ O R T
 S P O R T

Sequenzalignierung

A B C - B
 - B C D -
 - - - D B

	A	B	C	D	-
A	0	2	4	2	3
B		1	3	3	3
C			2	2	3
D				1	3
-					0

sum-of-pair cost: $6+7+8+7+7 = 35$,

Größere Protein/DNA Sequenzen

```
1thx    _aeqpvlvyfwaswgcgpcqlmsplinlaantysdrkvvkleidpnpttvkky...
1grx    __mqtvi__fgrsgcpysvrakdlaeklsnerdd_fqyqyvdiraegitkedl...
1erv    agdklvvdfsatwgcgpckmikpffhslsekysn_viflevdvddcqvasec...
2trcP   _kvttivvniyedgvrgcdalnssleclaaeypm_vkfckira_sntgagdrf...
```

```
1thx    ...k_____vegvpalrlvkgeqildstegvis__kdkllsf_ldthln_____
1grx    ...qqkagkpvvetvp__qifvdqqhiggytdfaawvken_____lda_____
1erv    ...e_____vksmptfqffkkgqkvgefsgan__kek_____leatine__lv____
2trcP   ...s_____sdvlptllvykggelisnfisvaeqfaedffaadvesflneygllper_
```

Rekursionsgleichung

$$m_1 = m'_1 x_1$$

$$m_2 = m'_2 y_2$$

$$\delta(m_1, m_2) = \begin{cases} \delta(m_1, m'_2) + w('-', x_2) & \text{if } |m_1| = 0 \\ \delta(m'_1, m_2) + w(x_1, '-') & \text{if } |m_2| = 0 \\ \min\{ \delta(m_1, m'_2) + w('-', x_2), & \text{(insertion of } x_2) \\ \delta(m'_1, m_2) + w(x_1, '-'), & \text{(insertion of } x_1) \\ \delta(m'_1, m'_2) + w(x_1, x_2) \} & \text{otherwise} \end{cases}$$

Programm

Procedure Align-Pairs

Input: Substitution costs w , Strings m, m'

Output: Matrix D containing shortest distances

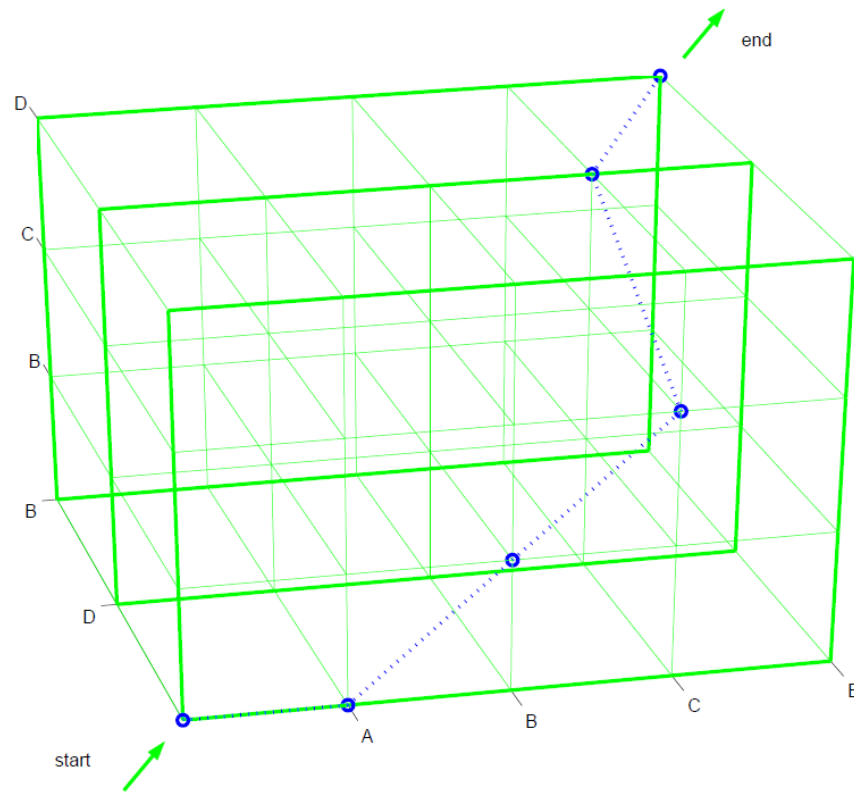
```

for each  $i$  in  $\{0, \dots, |m|\}$   $D_{i,0} \leftarrow w('-', m_i)$ 
for each  $i \leftarrow \{1, \dots, |m'|\}$   $D_{0,i} = w('-', m'_i)$ 
for each  $i$  in  $\{1, \dots, |m|\}$ 
    for each  $j$  in  $\{1, \dots, |m'|\}$ 
         $D_{i,j} \leftarrow \min\{D_{i,j-1} + w('-', m'_j),$ 
             $D_{i-1,j} + w(m_i, '-'),$ 
             $D_{i-1,j-1} + w(m_i, m'_j)\}$ 
return  $D$ 

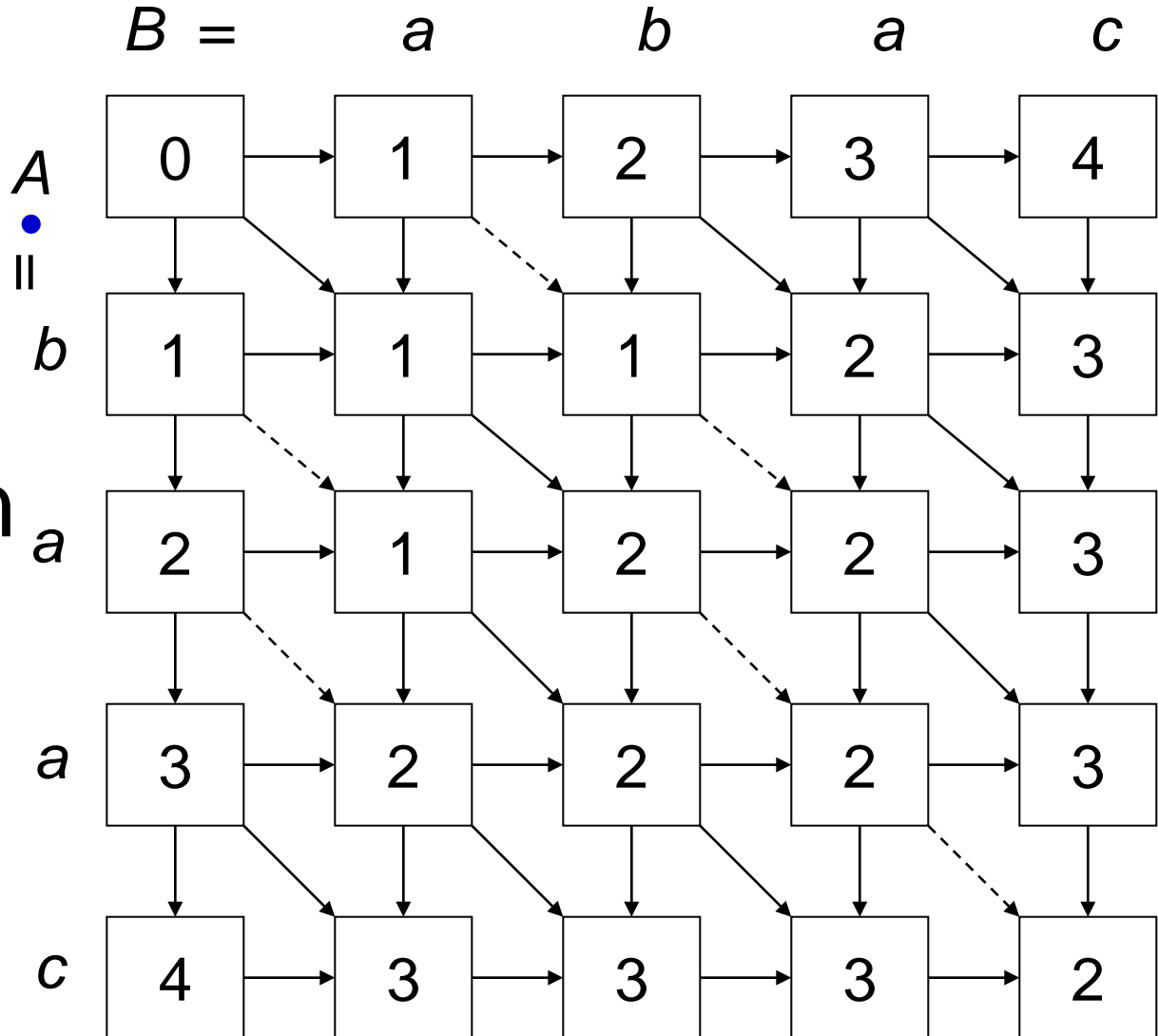
```

Beispiel: ABCD,BCD, DB

A B C _ B
 _ B C D _
 _ _ _ D B.



Spurgraph der Editier- operationen



Subgraph der Editieroperationen

Spurgraph: Übersicht über alle möglichen Spuren zur Transformation von A in B, gerichtete Kanten von Knoten (i, j) zu $(i + 1, j)$, $(i, j + 1)$ und $(i + 1, j + 1)$.

Gewichtung der Kanten entsprechen den Editierkosten.

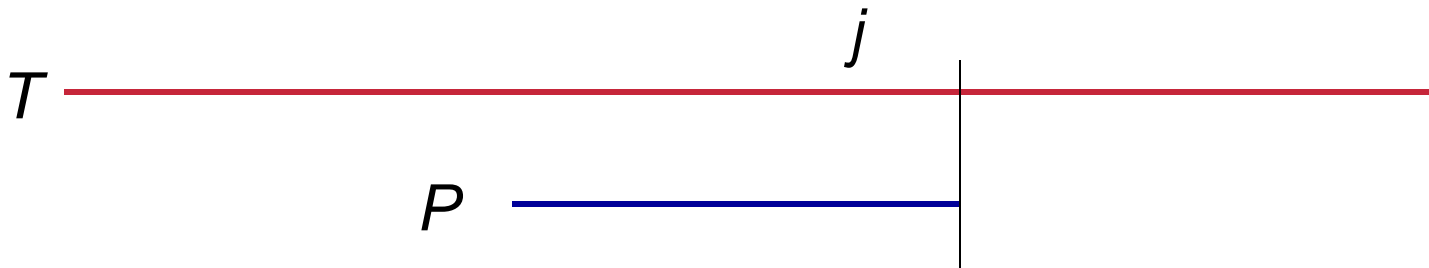
Kosten nehmen entlang eines optimalen Weges monoton zu.

Jeder Weg mit monoton wachsenden Kosten von der linken oberen Ecke zu rechten unteren Ecke entspricht einer optimalen Spur.

Approximative Zeichenkettensuche

Gegeben: zwei Zeichenketten $P = p_1 p_2 \dots p_m$ (Muster) und $T = t_1 t_2 \dots t_n$ (Text)

Gesucht: Ein Intervall $[j', j]$, $1 \leq j' \leq j \leq n$, so dass das Teilwort $T_{j', j} = t_{j'} \dots t_j$ das dem Muster P ähnlichste Teilwort von T ist, d.h. für alle anderen Intervalle $[k', k]$, $1 \leq k' \leq k \leq n$, gilt: $D(P, T_{j', j}) \leq D(P, T_{k', k})$



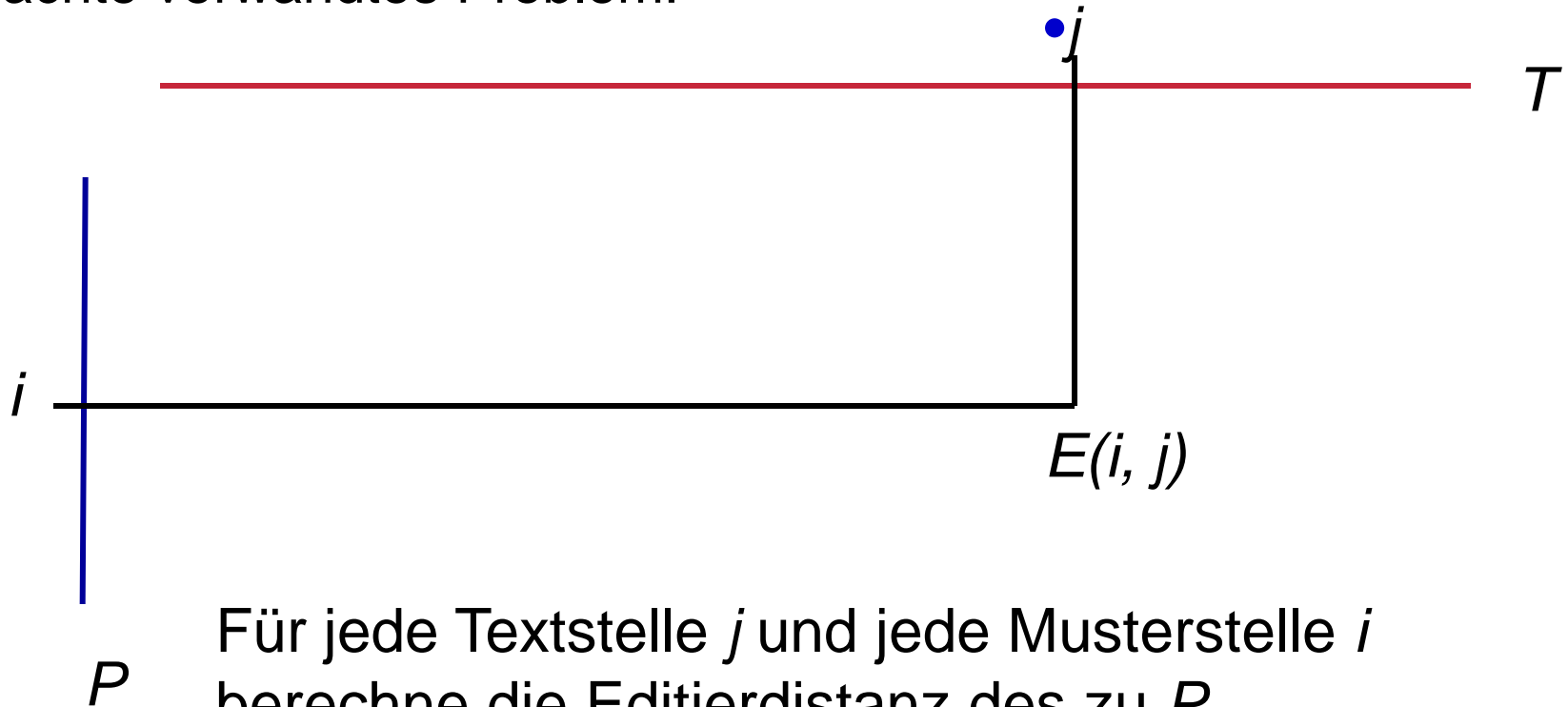
Erster Ansatz

Naives Verfahren:

for all $1 \leq j' \leq j \leq n$ **do**
 Berechne $D(P, T_{j', j})$
wähle Minimum

Trick

Betrachte verwandtes Problem:



Für jede Textstelle j und jede Musterstelle i berechne die Editierdistanz des zu P_i ähnlichsten, bei j endenden Teilstücks $T_{j,j}$ von T .

Algorithmus

Methode:

for all $1 \leq j \leq n$ **do**

Berechne j' , so dass $D(P, T_{j',j})$ minimal ist

Für $1 \leq i \leq m$ und $0 \leq j \leq n$ sei:

$$E_{i,j} = \min_{1 \leq j' \leq j+1} D(P_i, T_{j',j})$$

Optimale Spur:

$$\begin{array}{cccccccc}
 P_i & = & b & a & a & c & a & a & b & c \\
 & & | & | & / & / & | & / & & \\
 T_{j',j} & = & b & a & c & b & c & a & c &
 \end{array}$$

Rekursionsgleichung:

$$E_{i,j} = \min \left\{ \begin{array}{l} E_{i-1,j-1} + c(p_i, t_j), \\ E_{i-1,j} + 1, \\ E_{i,j-1} + 1 \end{array} \right\}$$

Bemerkung:

j' kann für $E_{i-1,j-1}$, $E_{i-1,j}$ und $E_{i,j-1}$ ganz verschieden sein.
Teilspur einer optimalen Spur ist eine optimale Teilspur.

Approximative Zeichenkettensuche

Anfangsbedingungen:

$$E_{0,0} = E(\varepsilon, \varepsilon) = 0$$

$$E_{i,0} = E(P_j, \varepsilon) = i$$

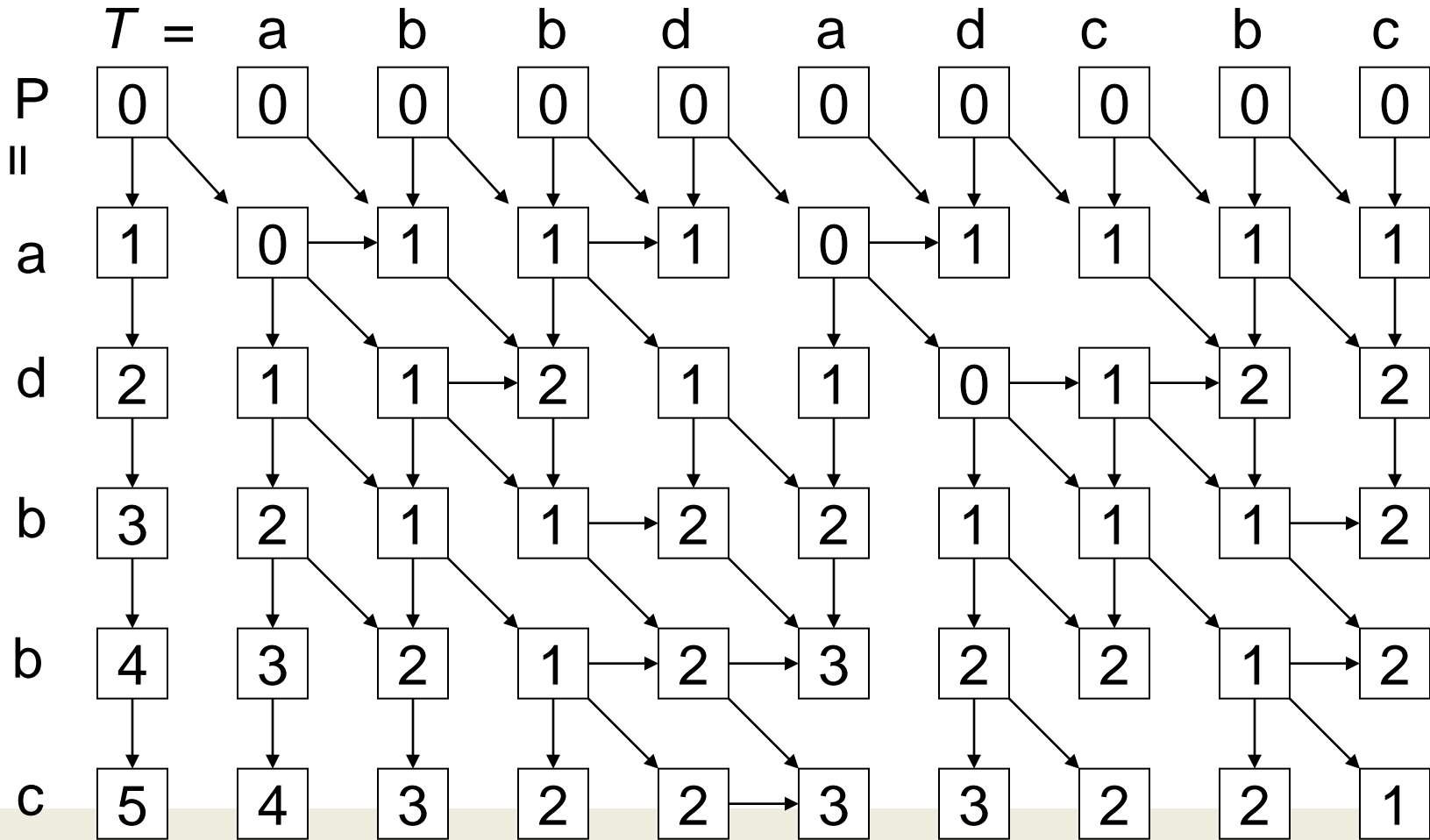
aber

$$E_{0,j} = E(\varepsilon, T_j) = 0$$

Beobachtung:

Die optimale Editiersequenz von P nach $T_{j',j}$ beginnt nicht mit einer Einfügung von $t_{j'}$.

Abhängigkeitsgraph



Approximative Zeichenkettensuche

Satz

Gibt es im Abhängigkeitsgraphen einen Weg von $E_{0, j'-1}$ nach $E_{i, j}$, so ist $T_{j', j}$ ein zu P_i ähnlichstes, bei j endendes Teilstück von T mit

$$D(P_i, T_{j', j}) = E_{i, j}$$

Übersicht

- ▶ Speicherplatzsparende Wörterbücher
 - Bit-State Hashing
 - Kollaps-Kompression
- ▶ Subzeichenketten-Erkennung:
 - Dynamische Programmierung
- ▶ **Teilwort-Erkennung:**
 - Listen und Array
 - Hashing und Tries
 - Unlimited Branching Trees

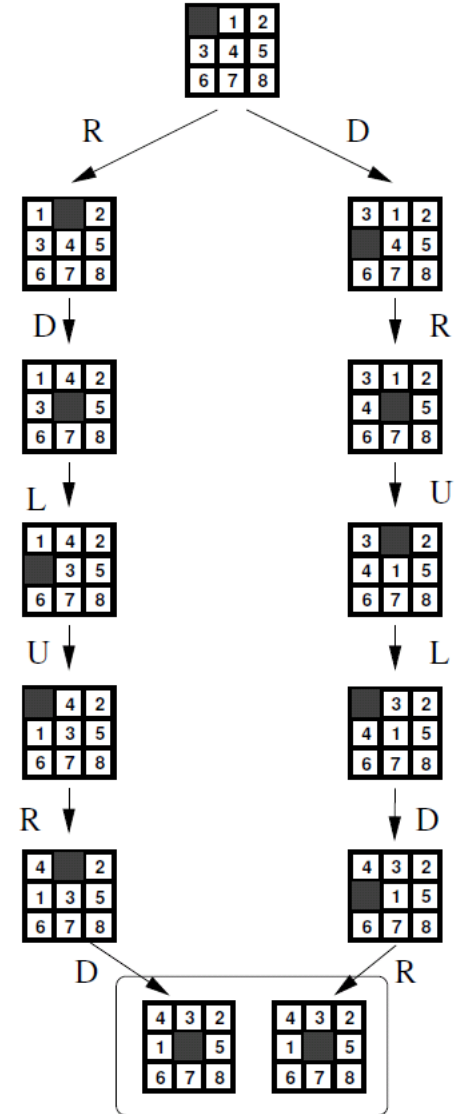
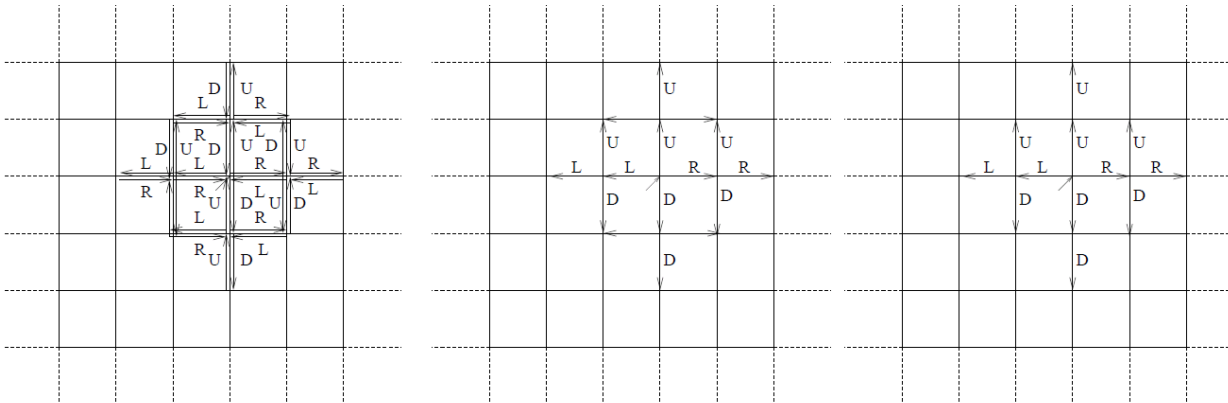
Teilmengen-Wörterbücher

- ▶ Sei D eine Menge von n Teilmengen aus U .
- ▶ Das SUBSET QUERY (CONTAINMENT QUERY) Problem fragt für q ob es ein p in D gibt mit q ist Teilmenge von p (p ist Teilmenge von q).
- ▶ Ein Teilmengen Wörterbuch ist eine abstrakte Datenstruktur die Einfügen erlaubt und Teilmengen- (Supermengen-) Anfragen anbietet

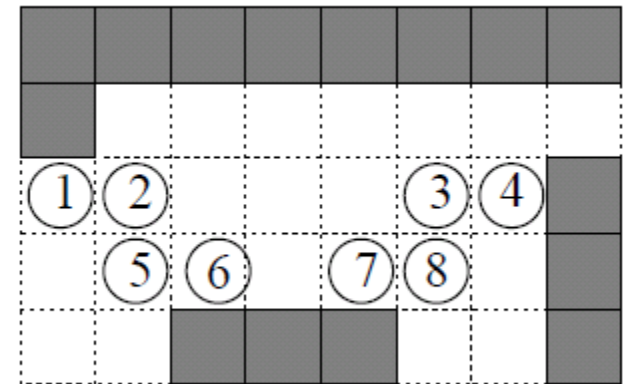
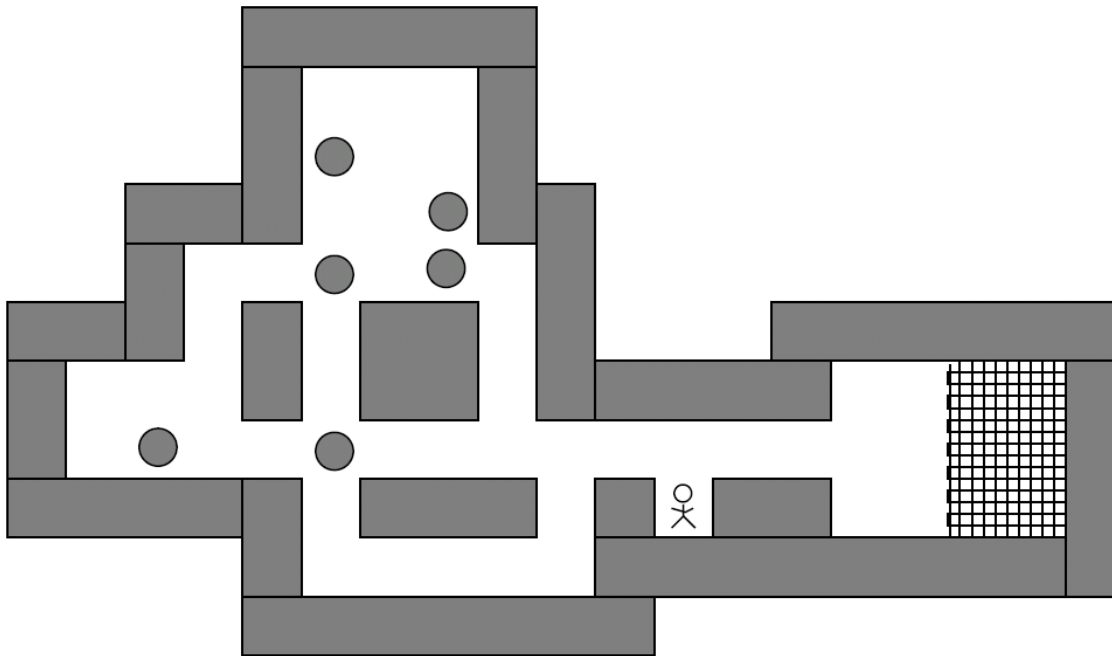
Subzeichenketten- vs. Teilzeichenkettenmatching

- ▶ Subzeichenkette: ADEK in Alphabet ABCD....XYZ
- ▶ Teilzeichenkette: STUV in Alphabet ABCD....XYZ

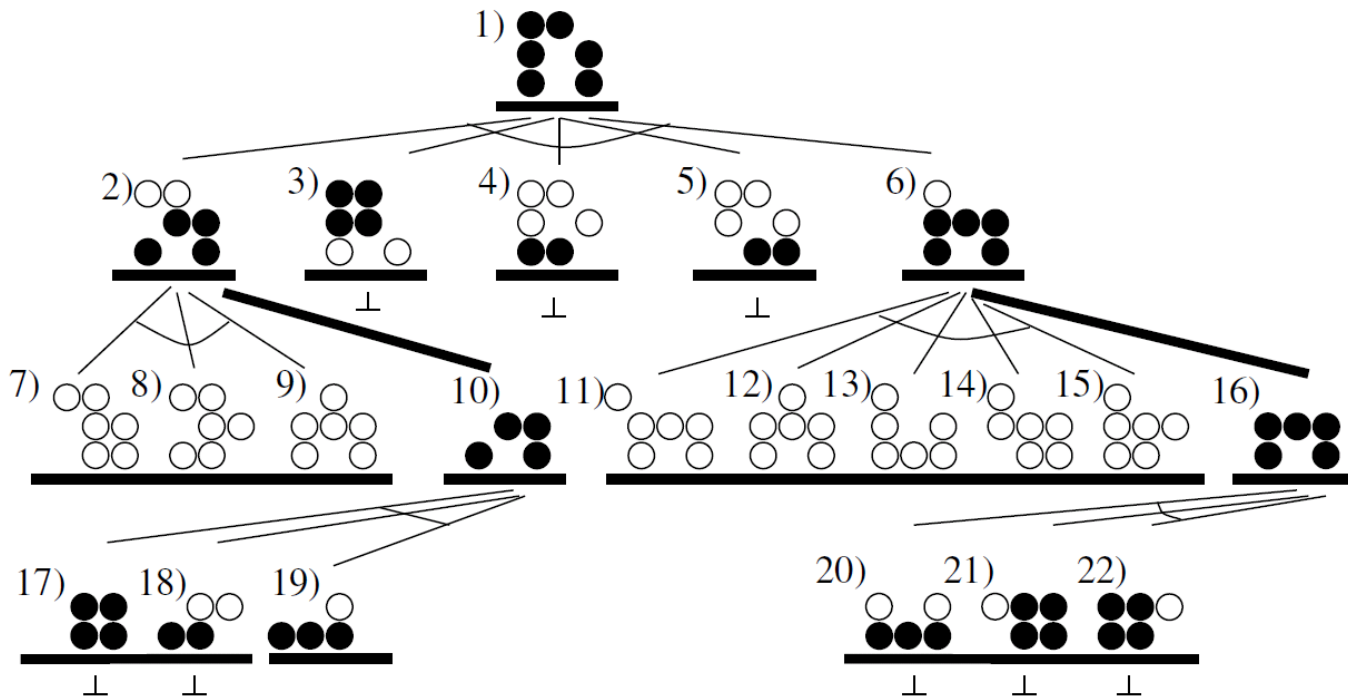
Anwendung Teilzeichenkettenmatching



Beispiel Subzeichenkettenmatching: Sokoban



Generierung größerer Muster



Anfragen mit Wildcards

- ▶ Sei ? ein spezieller don't care das jeden Buchstaben im Alphabet ersetzt (wildcard)
- ▶ Gegeben eine Menge D von n Zeichenketten, dann beantwortet eine Datenstruktur für das PARTIAL MATCH Problem für eine Anfrage q (mit wildcards) ob es einen Eintrag q in D gibt, so dass q zu p passt

Kreuzworträtsel - Konstruktionsproblem

Mail-Teil welcher Informationen enthält	B	F	Überfluten eines Dienstes	F	DOS: Vergleich des Inhalt von Dateien	Emergency (scip AG)	S	B ₁	Back Office	O	Objekt orientierte Programmierung	Wonach sucht Wellenrater	W		
Distributed Denial of Service	O	L	Linux: Kopiert Dateien	C	P	Prozessoren in Apple's	M	O	T	O	R	O	L	A	
D	D	O	S	Soll den Data Encryption Standard abkochen	A	E	S	HA/Grundlage für Palladium	T	P	M	Künstliche Intelligenz	A	I	
F	Y	O	D	O	R	Council of European National Top level Domain Register	S	E	Virtual Private Network	V	P	N			
Entwickler von rmap	Begründer der Relativitätstheorie	D	Analog-Digital-Wandler	A	D	C	beginnt (Schach)	W	E ₅	I	S	S	scip monthly Security Summary		
Hersteller von Solaris	E	I	N	S	T	E	I	N	Basic Input Output System	B	I	O	S		
S	U	N	6	Data Encryption Standard	Internet Protocol	Privat-gesamten TLD	N	A	M	E	Protokoll für Fehler und Information	I	C	M	P
Klassischer UNIX-Texteditor	Abk.: Intrusion Detection-System	G	D	I	Speicher-residentes Programm	T	S	R	Forum of Incident Response and Security Teams	F	I	R	S	T	
V	I	Krypto-graphische Weiterentwicklung von Teletext	E	P	Lachend auf dem Boden wälzen	R ₃	O	T	F	L	Hersteller von Real Secure	I	S	S	
Vom NIST vorgeschlagener Standard für digitale Signaturen	D ₄	S	S	Protokoll für das Übertragen von Daten	F	Zeichensatz der für Europäische Institutionen verwendet wird	K	A	N	J	I				
X	S	S	Was schrieb Robert Tappan Morris	N	T	F	S	Unix: TEXT-Datei erzeugen	Port scanner	N	M	A	P		
Cross Site Scripting	Verschlüsselungs-Mechanismus für HTTP	H	W	Dateisystem von Windows NT und 2000	P	Gezielt informiert (scip AG)	Aho, Weinberger, Kerrighan	M	O	R	E				
J	S	UNIX-Kommando equivalent zu dir unter DOS	U	F	Gibt Kommando-Prozessor	P	A ₂	L	L	A	S				
Javascript	S	L	R	P	L	Briefqualität	W								
	L	S	M	U	Q		K								

State-of-the-Art !?

Combus „Crossword Puzzles as a Constraint Problem“

- ▶ CP 2008: Anbulagan and Adi Botea
- ▶ (words 45K, UK 220K Einträge, T sek. N expan. Knoten)

Inst ance	15x15 grids				19x19 grids				21x21 grids				23x23 grids			
	words		UK		words		UK		words		UK		words		UK	
	T	N	T	N	T	N	T	N	T	N	T	N	T	N	T	N
01	86	83	287	71	56	118	320	123	113	471	851	128	1	0	209	157
02	11	75	216	74	23	96	429	109	134	143	1133	141	81	178	697	172
03	41	71	239	73	34	315	245	108	76	139	624	130	455	12121	1185	160
04	18	304	290	66	75	127	738	116	740	15367	525	139	180	1700	715	162
05	25	65	354	70	13	112	121	115	56	238	288	132	105	178	707	170
06	84	1678	521	64	96	126	313	121	99	140	560	137	–	–	462	227
07	146	118	548	65	34	118	296	126	128	152	571	136	86	241	572	162
08	41	76	196	78	62	122	396	120	68	145	551	142	320	7842	766	156
09	25	77	210	75	23	121	342	121	64	141	479	138	689	18109	366	160
10	114	506	462	65	14	119	120	121	–	–	857	119	–	–	680	135

Datenstruktur? 1. Array und Listen

Arrays:

- ▶ Platz $O(2^m)$, $m = |U| \rightarrow$ Zu groß in der Praxis
- ▶ Anfragezeit $O(m)$

Listen:

- ▶ Optimaler Platz $O(n)$
- ▶ Anfragezeit: $O(nm) \rightarrow$ Zu groß in der Praxis

2. Tries (PARTIAL MATCH)

Procedure Lookup

Input: TRIE node u , query q , level l

Output: Display all entries p with q matches p

```
if (Leaf( $u$ ))
    if (Match(Entry( $u$ ),  $q$ ))
        print Entry( $u$ )
if ( $q_l \neq *$ )
    if (Succ( $u$ ,  $q_l$ )  $\neq \perp$ )
        Lookup(Succ( $u$ ,  $q_l$ ),  $q$ ,  $l + 1$ )
else
    if (Succ( $u$ , 0)  $\neq \perp$ )
        Lookup(Succ( $u$ , 0),  $q$ ,  $l + 1$ )
    if (Succ( $u$ , 1)  $\neq \perp$ )
        Lookup(Succ( $u$ , 1),  $q$ ,  $l + 1$ )
```

3. Hashing (PARTIAL MATCH)

Procedure Lookup

Input: Chained hash table T , hash function h ,

Output: All entries p with q matches p

$L \leftarrow \emptyset$

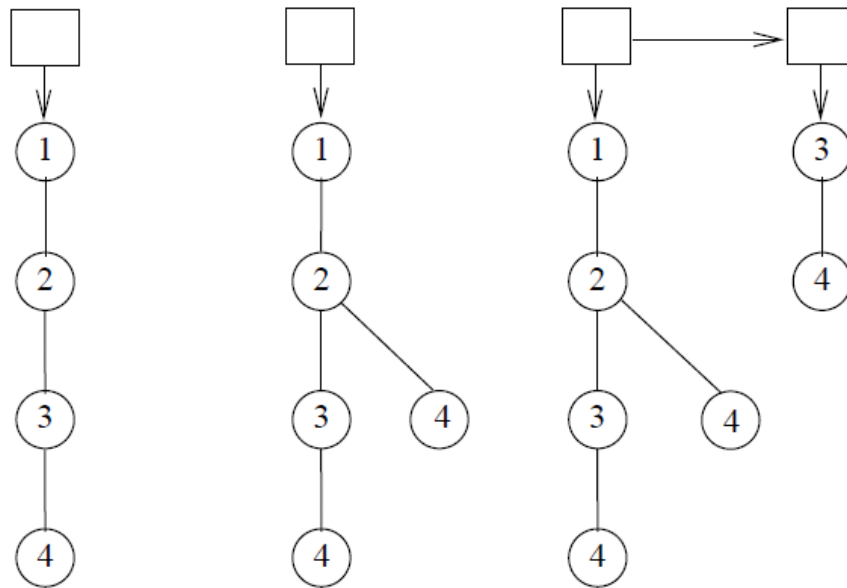
for each $j \in h(q)$

for each $p \in L_j$

if ($Match(p, q)$) $L \leftarrow L \cup p$

return L

4. „Unlimited“ Branching Trees (CONTAINMENT QUERY)



$\{1, 2, 3, 4\}$, $\{1, 2, 4\}$, $\{3, 4\}$

Einfügen UBT

Procedure Insert

Input: UNLIMITED BRANCHING TREE $L = (T_1, \dots, T_k)$, sorted set $p = \{p_1, \dots, p_l\}$

Side Effect: Modified UNLIMITED BRANCHING TREE data structure

```
for each  $i$  in  $\{1, \dots, k\}$                                      ;; Consider all tries
  if ( $p_1 = \text{root}(T_i)$ )                                       ;; Matches root list
    Trie-Insert( $T_i, q$ ) ; return                                ;; Insert into trie and quit
Generate a new trie  $T'$  for  $p$                                    ;; Temporary trie for inserted set
Insert  $T'$  into list  $L$                                          ;; Include new trie into sorted list
```

Suche UBT

Procedure Lookup

Input: UNLIMITED BRANCHING TREE $L = (T_1, \dots, T_k)$, sorted query $q = \{q_1, \dots, q_m\}$

Output: Flag indicating, whether or not p is contained in L with $q \supseteq p$

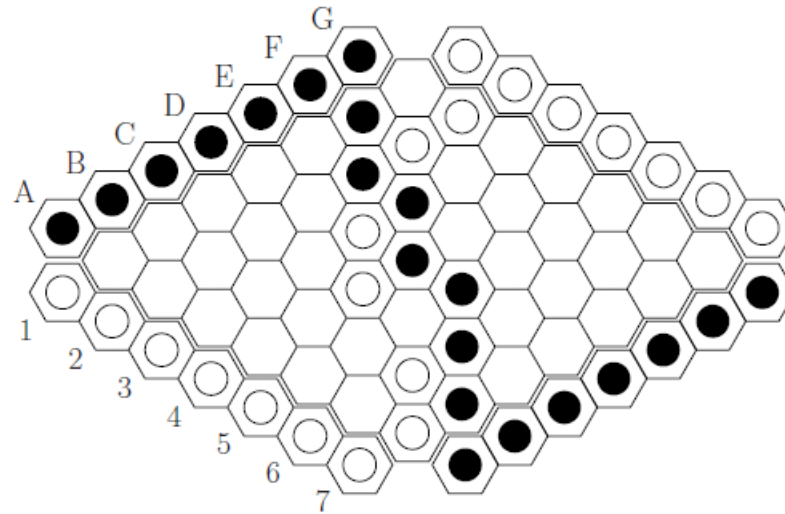
```

Q ← ∅
for each i in {1, ..., k}
    if (root(Ti) ∈ q)
        Q ← Q ∪ {Ti}
for each Ti in Q
    if (Trie-Lookup(Ti, q)) return true
return false

```

;; Initialize queue
 ;; Consider all tries
 ;; Matches root list
 ;; Insert trie to candidate set
 ;; Process Queue
 ;; Search individual trie
 ;; Search failed

Beispiel: Hex



Virtuelle Verbindungen

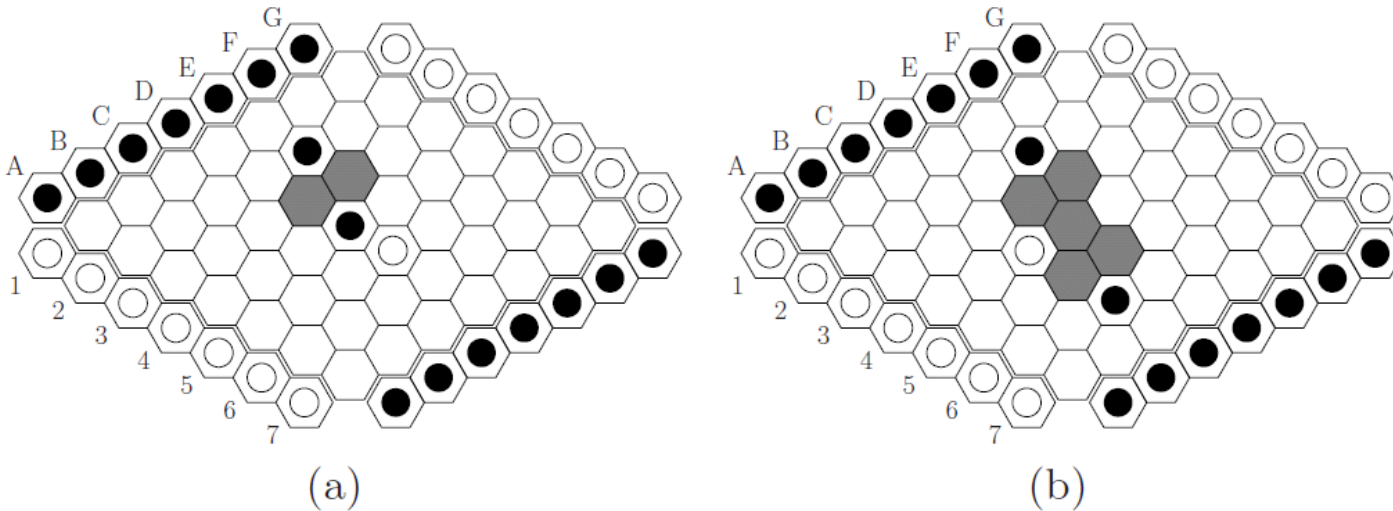
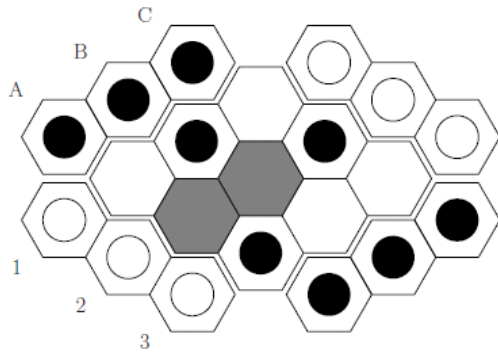
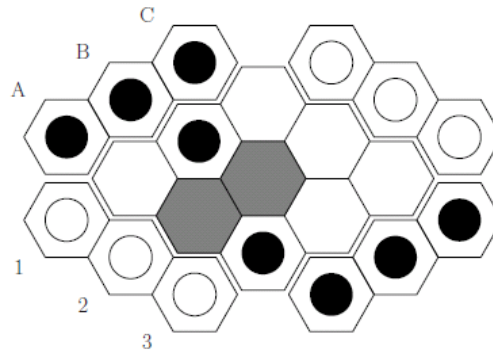


Fig. 2. Virtual connection (left) and virtual semi-connection (right) in HEX

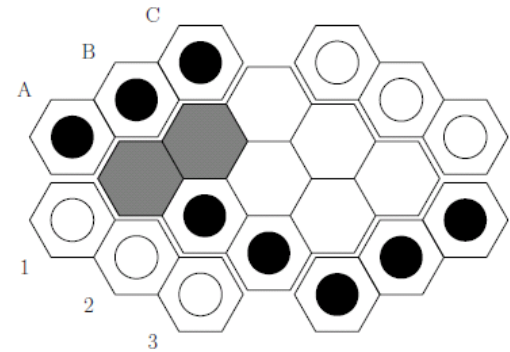
Zielmusterdatenbank



(a) Element $A = (s_A, b_A)$

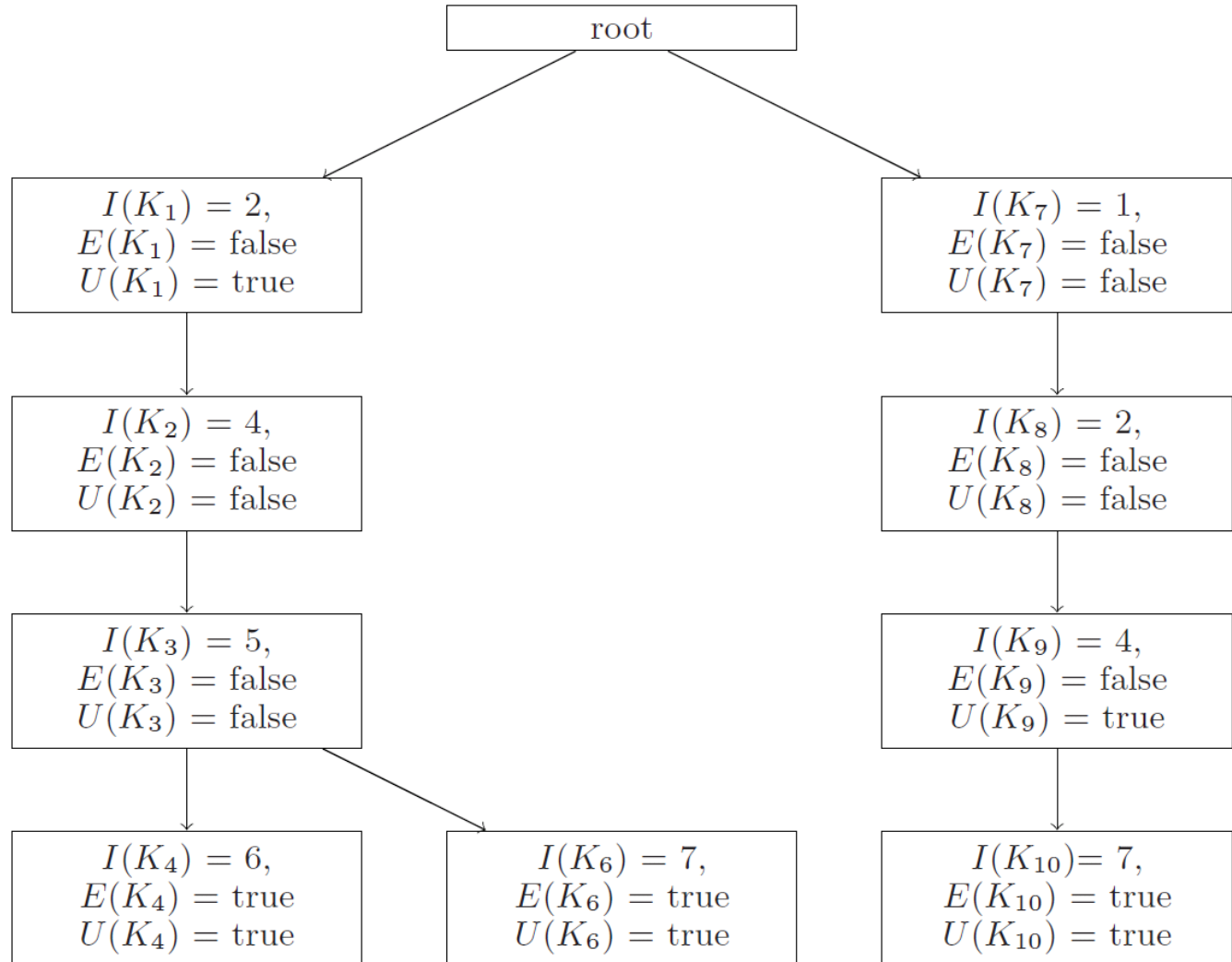


(b) Element $B = (s_B, b_B)$

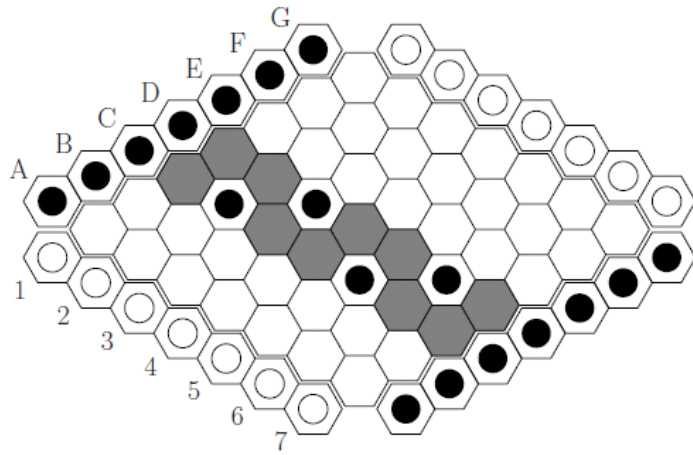


(c) Element $C = (s_C, b_C)$

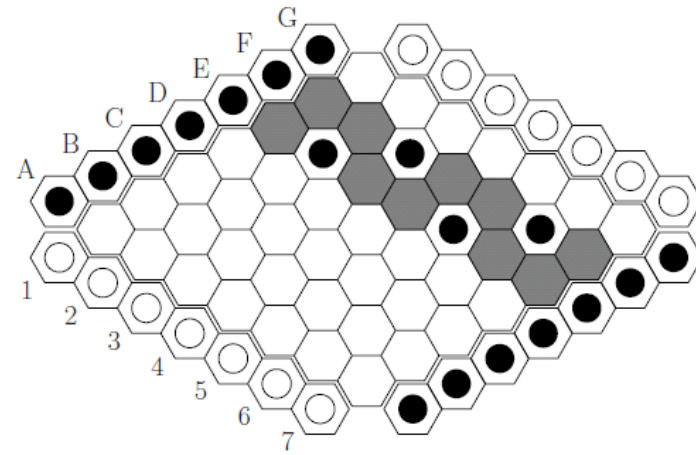
UBT-Adaption



Symmetrien



(a)



(b)