FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV/PRO

Přibližné porovnávání regulárních výrazů s více řetězci

Jméno a Příjmení

Osobní číslo

Datum

E-mail

1. Úvod a formulace problému

Mějme dán regulární výraz *R* a chceme provést následující úkol: v textu *T* najděte všechny jeho podřetězce, které se shodují s regulárním výrazem *R* s nejvýše *k* chybami (chybou může být smazání, vložení, nebo nahrazení jednoho symbolu).

Tento problém vyvstává ve spoustě aplikací pracujících s textem, jako například hledání v textu, úprava textu, ale také v počítačové biologii a síťovém zabezpečení. Existuje známé řešení v čase *O(mn)* [2], kde *m* je délka regulárního výrazu a *n* je délka textu. Také existuje řešení v čase *O(dn)* [3] v případě *k=0* (přesné porovnávání), kde *d* je počet řetězců v regulárním výrazu. D. Belazzougui a M. Raffinot spojili obě tato řešení do jednoho [1] a dosáhli tak času *O(kdn)* pro libovolné *k*. A pokud uvážíme, že většina regulárních výrazů hledá slova nebo jejich části, pak algoritmus, lineárně závislý na počtu slov (*d*) regulárního výrazu, může být mnohem rychlejší než ten, který je závislý na délce (*m*) regulárního výrazu. Tato časová úspora bude o to větší, čím více slov bude regulární výraz obsahovat.

2. Notace a definice

Následující notace a definice jsou uvedeny tak, jak byli použity v článku [1].

Regulární výraz je obecný vzor složený ze (i) základních řetězců, (ii) sjednocení, zřetězení a Kleenovy hvězdy jiných regulárních výrazů. Symbolem *m* označujeme délku našeho regulárního výrazu, nepočítaje operační symboly. Abeceda je označena Σ a *n* je délka textu, který prohledáváme.

Slovo je řetězec znaků, patřících do konečné abecedy Σ. Prázdné slovo je označováno ε a množina všech slov složená ze symbolů abecedy Σ včetně ε se označuje jako Σ^*. Slovo $x \in \Sigma^*$ je předponou (resp. příponou) slova *p*, pokud $p=xu$ (resp. $p=ux$), kde $u \in \Sigma^*$. Sjednocení je označeno znakem "|",Kleenova hvězda znakem "*", a zřetězení znakem ".", nebo jednoduše vložením podvýrazů za sebe. Závorky jsou použity ke změně přednosti, která normálně následuje takto: "*","."," | ". Znakem *R* značíme náš regulární výraz, který má délku *m* a obsahuje *d* řetězců. Znakem *L(R)* značíme množinu všech slov generovaných *R*. A nakonec, v textu T označíme znaky *T[i..j]* podřetězce textu *T*, které začínají na pozici *i* a končí na pozici *j*.

Definice 1. *Vzdáleností* mezi dvěma řetězci s_1 a s_2 označujeme minimální počet operací potřebných k transformaci s_1 na s_2, kde povolenými operacemi jsou vymazání znaku, nahrazení znaku jiným, nebo vložení znaku. Dále předpokládáme, že počet operací $\delta(s_1,s_2)=\delta(s_2,s_1)$.

Definice 2. Přibližné porovnávání řetězců je problém, ve kterém dostaneme řetězec *q*, hraniční počet *k* a text *T*, a naším úkolem je vrátit každé *i* takové, že existuje podřetězec *T[i..j]* textu *T* takový, že $\delta(T[i..j],q) \leq k$.

Definice 3. Porovnávání regulárních výrazů je problém, ve kterém máme regulární výraz *R* a text *T,* a naším úkolem je vrátit každé *i* takové, že existuje podřetězec *T[i..j] ∈L(R), kde L(R)* je jazyk generovaný *R*.

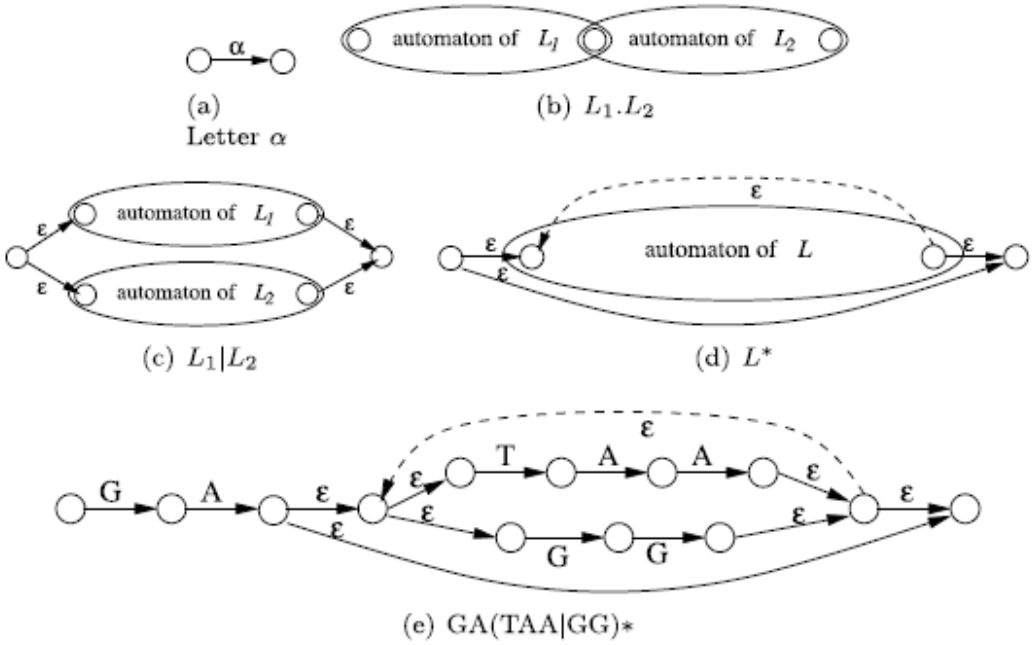
Definice 4. Přibližné porovnávání regulárních výrazů je problém, ve kterém máme regulární výraz *R*, text *T* a hraniční počet *k*, a naším úkolem je vrátit každé *i* takové, že existují podřetězec *T[i..j]* a řetězec *p∈L(R) takové, že δ(p,T[i..j])≤ k.*

3. Nový Belazzouguiův a Raffinotův algoritmus

Belazzougui a Raffinot v jejich článku [1] představily algoritmus, který dokáže řešit problém přibližného porovnávání regulárních výrazů v čase *O(kdn).* Jejich algoritmus je založen na Myersovu a Millerovu algoritmu pro přibližné porovnávání [2] a algoritmu P. Billa a M. Thorupa pro přesné porovnávání [3] a proto si nejprve přiblížíme tyto dva algoritmy.

3.1. Thompsonův automat

Algoritmus využívá klasickou konstrukci Thompsonova automatu k sestavení non-deterministického automatu, který akceptuje regulárním výrazem *R* generovaný jazyk. Automat obsahuje ε-přechody a rozlišuje mezi dvěma typy uzlů: *ε-uzly*, ve kterých jsou všechny vstupující přechody ε-přechody, a *L-uzly*, do kterých vede jeden přechod označen znakem l_i. Automat je sestaven rekurzivně, použitím vzorů z následujícího obrázku (převzato z [1]):



Obrázek 1. Vzory pro rekurzivní sestavení Thompsonova automatu na daném regulárním výrazu a příklad takto sestaveného automatu na výrazu *GA(TAA|GG)*.*

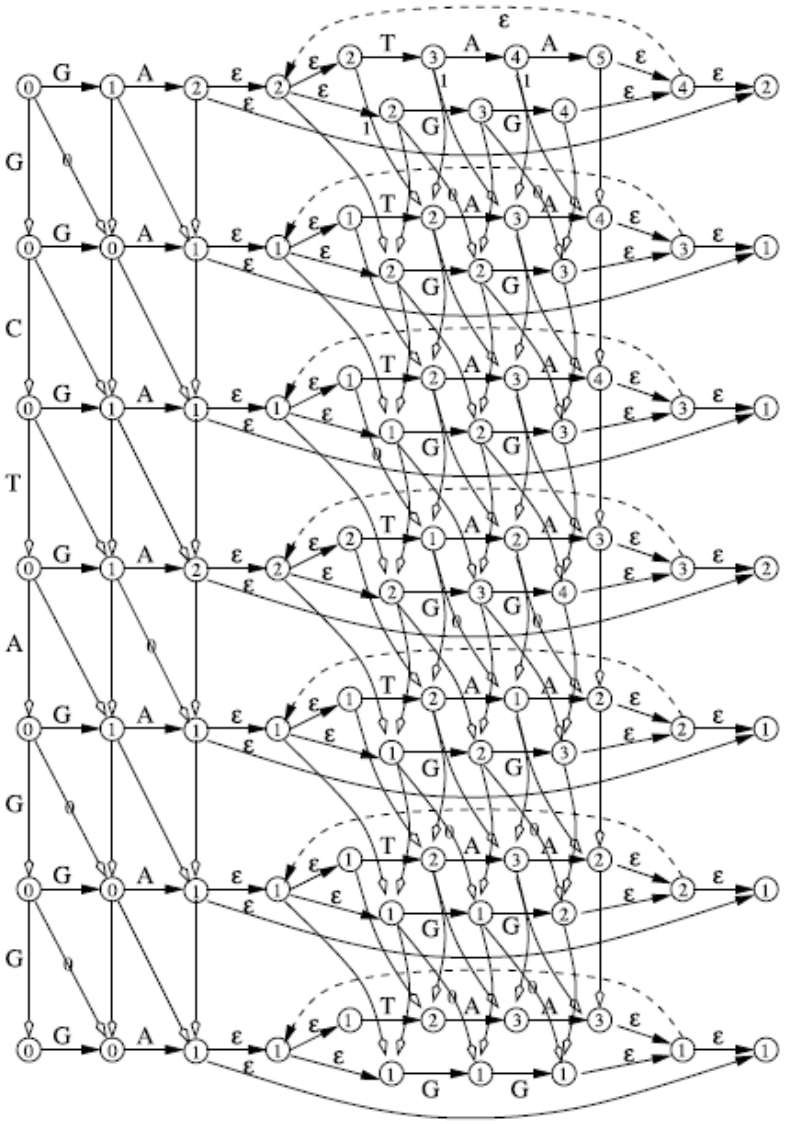
3.2. Myersův a Millerův algoritmus

Během prohledávání textu *T* udržujeme počítadlo c_i o velikosti horní celé části z *log(k+1)* bitů pro každý stav *i* automatu. V každém kroku *j* čtení textu *T* bude toto počítadlo obsahovat nejmenší vzdálenost mezi jakoukoli příponou *T[1..j]* a jazykem reprezentovaným stavem automatu *i*. A protože jazyku *L(R)* odpovídá akceptační stav automatu *F*, bude počítadlo tohoto stavu c_F obsahovat minimální vzdálenost mezi jakoukoli příponou *j* a jazykem *L(R).*

Symbolem *E[i,j]* označíme nejmenší vzdálenost mezi stavem i a příponami *T[1..j]*. To znamená, že v kroku j, kdy čteme znak textu *T[j]*, se $E[i,j]=c_i$. Stavy automatu značíme znakem *I* pro vstupní stav a znakem *F* pro výstupní stav automatu. Stavy jsou číslovány od *1* do *D* kromě počátečního stavu. Stav *i* nazýváme *L-uzlem,* pokud do něj vede jen jeden přechod označený znakem l_i. Jinak bude stav *ε-uzlem* se všemi do něj vstupujícími přechody označenými symbolem *ε.* Pro libovolný uzel *i* označíme *Pre(i)* množinu všech uzlů, z nichž vede přechod do *i* (pokud je *i L-uzlem*, potom množina *Pre(i)* bude mít právě jeden prvek). Navíc pro *ε-uzly* označíme $\overline{Pre}(i)$ množinu uzlů, ze kterých vede přechod do *i*, kromě zpětných přechodů. Hodnoty *E[i,j]* jsou nastaveny podle následujícího algoritmu. Pseudokód a schéma algoritmu můžeme vidět na následujících obrázcích (převzaty z [1]):

1: **for** $j = 1$ **to** n **do**
2: $E[I, j] \leftarrow 0$
3: $E'[I, j] \leftarrow 0$
4: **end for**
5: **for** $i \in [1, D]$ **do**
6: $E[i, 0] \leftarrow \begin{cases} \min E[\overline{Pre}(i), 0] + 1 & \text{if } i \text{ is an } L\text{-node} \\ \min E[\overline{Pre}(i), 0] & \text{if } i \text{ is an } \varepsilon\text{-node} \end{cases}$
7: **end for**
8: **for** $j \in [1, n]$ **do**
9: **for** $i \in [1, D]$ **do**
10: $E'[i, j] \leftarrow \begin{cases} \text{if } i \text{ is an } L\text{-node} \\ \quad \min(E[i, j-1] + 1, E[\overline{Pre}(i), j-1] + \delta(\ell_i, T[j]), E'[\overline{Pre}(i), j] + 1) \\ \text{if } i \text{ is an } \varepsilon\text{-node} \\ \quad \min E'[\overline{Pre}(i), j] \end{cases}$
11: **end for**
12: **for** $i \in [1, D]$ **do**
13: $E[i, j] \leftarrow \begin{cases} \min(E'[i, j], E[\overline{Pre}(i), j] + 1) & \text{if } i \text{ is an } L\text{-node} \\ \min(E'[Pre(i), j], E[\overline{Pre}(i), j]) & \text{if } i \text{ is an } \varepsilon\text{-node} \end{cases}$
14: **end for**
15: **end for**

Obrázek 2. Myersův a Millerův algoritmus.



Obrázek 3. Schéma Myersova a Millerova algoritmu aplikovaného na výraz *GA(TAA|GG)*.*

3.3 Billův a Thorupův algoritmus pro přesné porovnávání

Algoritmus nejdříve vygeneruje nový regulární výraz *R'* seskupením všech maximálních sekvencí po sobě jdoucích operátoru "." v původním regulárním výrazu *R* do jednoho řetězce znaků. Každý takto vzniklý řetězec přidáme do množiny *S* a ve výrazu *R'* ho nahradíme jediným meta-znakem. Nově vzniklý výraz má *O(d)* stavů, kde *d* je počet slov v původním výrazu. Ve druhém kroku je sestaven Aho-Corasick automat na množině řetězců *S* a Thompsonův automat na výrazu *R*.

Pro porovnávání regulárního výrazu je udržována fronta délky m_i pro každý řetězec *s* množiny S_i délky m_i. Fronta využívá m_i bitů a označujeme ji $q_i[1..m_i]$. Na začátku jsou všechny bity nastaveny na *0*. V každém kroku porovnávání jsou prvky fronty posunuty o pozici doprava a na pozici *q[1]* je vložen bit 0 nebo 1. Algoritmus pokračuje v následujících krocích:

1. Do Aho-Corasick automatu načti další znak *c* z textu.

2. Automat vratí množinu výskytů řetězců z množiny *S*. a pro každý takovýto řetězec $s_i \in S$ vlož bit 1 do fronty q_i. Pro všechny ostatní řetězce $s_i \in S$ vloží do fronty bit 0.

3. Nakonec je proveden Thompsonův automat, ve kterém si však všímáme jen těch přechodů označených *ε nebo meta-znakem* $s_i \in S$, pro které je $q_i[m_i]=1$.

3.4. Inkrementální porovnávání řetězců

Předtím než si uvedeme nový algoritmus, potřebujeme ještě vyřešit následující problém. Mějme dán výraz *p* délky *m,* celé kladné číslo *k* a text *T*, který chceme v jakémkoliv kroku *j* přečíst znak po znaku, identifikovat všechny přípony *T[1..j]*, které jsou od *p* ve vzdálenosti *k*. Vzhledem k faktu, že délka těchto přípon se může od *m* lišit maximálně o *k* (kdyby byly kratší, či delší, bylo by zapotřebí moc vložení, či smazání jednoho znaku, abychom dostali znovu *p*), je počet těchto přípon nejvýše *2k+1*. Následující tvrzení předkládá řešení tohoto problému. K tomuto problému využijeme řešení [4], můžeme tedy tvrdit následující:

Tvrzení 1. (viz. [4]) Mějme dán výraz *p* délky *m*, po *O(m)* čase předzpracování jsme schopní vyřešit problém přibližného porovnávání řetězců pro text *T* v čase *O(kn)*. Algoritmus je schopen vrátit v jakémkoliv kroku *j* vzdálenost $\delta(T[i..j])$ pro každé *i* takové, že $\delta(T[i..j]) \leq k$.

3.5. Nový *O(kdn) a*lgoritmus

Nové řešení kombinuje tři výše uvedené algoritmy:

- Algoritmus pro průběžné nastavení vzdáleností pro každý stav automatu. [2]
- A nakonec algoritmus pro zachování vypočítaných vzdáleností pro každý stav z předchozích kroků. [3]
- Algoritmus pro počítání vzdáleností přípon textu *T* a všech řetězců v množině *S*. [4]

3.5.1. Předzpracování

Fáze předzpracování je téměř totožná s tou v algoritmu Billa a Thorupa. Mějme regulární výraz *R*, nový regulární výraz *R'* vznikne seskupením maximální sekvence souvislých operátorů "." do řetězců, přidáním takových řetězců do množiny *S* a jejím nahrazením v regulárním výrazu *R'* meta-znakem. Pro každý prvek množiny $s_i \in S$ uchováváme frontu q_i o m_i+k prvcích, kde každý prvek obsahuje počítadlo o velikosti horní celé části z *log(k+1) bitů,* jejichž hodnota je na začátku nastavena na *k+1*.

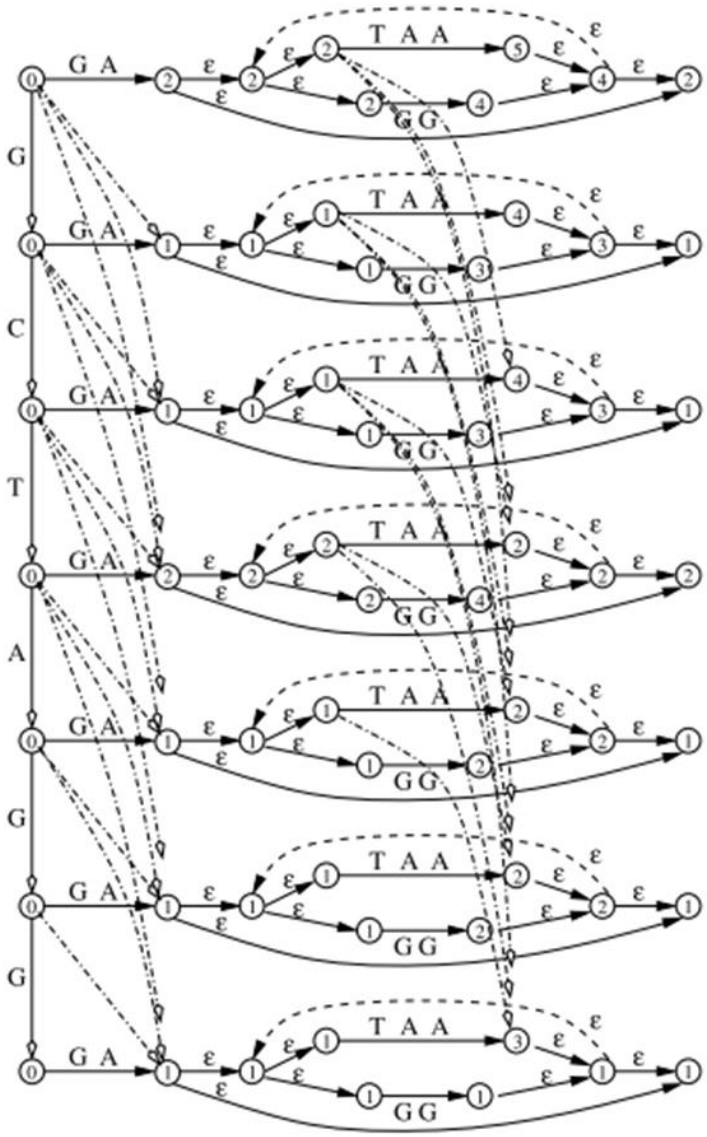
3.5.2. Algoritmus porovnávání

V každém kroku *j*, funkce *ED(i,j)* vrací vektor $V[1..min(m_i,k+1)+k]$, který obsahuje všechny vzdálenosti s_i a podřetězců $T[(j-(m_i+k)-1)..j],\ldots,T[(j-(max(m_i,k+1)-k)+1)..j]$. A pro

libovolnou frontu $q_i[1..m_i+k]$ funkce $push(q_i,x)$ vloží hodnotu x na konec fronty q_i. Dále bude použit dočasný vektor Δ velikosti 2k+1. Pomocné hodnoty algoritmu jsou ukládány do matice $E[1..D, 1..n]$. Následující obrázky obsahují pseudokód a schéma tohoto algoritmu (převzato z [1]):

```
1:  E[I, 0] ← 0
2:  for i ∈ [1, D] do
3:    if i is an L-node then
4:      E[i, 0] ← min E[Pre(i), 0] + m_i
        push(q_i, E[Pre(i), 0])
5:    else
6:      E[i, 0] ← min E[Pre(i), 0] // i is an ε-node
7:    end if
8:  end for
9:  for j ∈ [1, n] do
10:   E'[I, j] ← 0
11:   for i ∈ [1, D] do
12:     if i is an L-node then
13:       Δ[1.. min(m_i, k + 1) + k] = ED(i, j)
14:       E'[i, j] ← E'[Pre(i), j] + m_i
15:       for t = max(m_i − k, 1) to m_i + k do
16:         E'[i, j] ← min(E'[i, j], q_i[t] + Δ[m_i − t + k + 1])
            /* Note that q_i[t] = E[Pre(i), j − t] and Δ[m_i − t + k + 1] = δ(s_i, T[j − t + 1..j]). /*
17:       end for
18:     else
19:       E'[i, j] ← min E'[Pre(i), j] // i is an ε-node
20:     end if
21:   end for
22:   E[I, j] ← 0
23:   for i ∈ [1, D] do
24:     if i is an L-node then
25:       E[i, j] ← min(E'[i, j], E[Pre(i), j] + m_i)
          push(q_i, E[Pre(i), j])
26:     else
27:       E[i, j] ← min(E'[Pre(i), j], E[Pre(i), j]) // i is an ε-node
28:     end if
29:   end for
30: end for
```

Obrázek 4. Pseudokód nového *O(kdn)* algoritmu.



Obrázek 5. Schéma algoritmu aplikovaného na výraz *GA(TAA|GG)** v textu *GCTAGG* pro *k=1*. Pro přehlednost obrázku nejsou výstupní přechody z tučně zvýrazněných stavů zakresleny.

Aby byla dokázána správnost tohoto algoritmu, je třeba dokázat správnost vypočítaných vzdáleností pro *L-uzly* a *ε-uzly*.

Pro *L-uzly* uvažme libovolný *L-uzel a,* s do něj vstupujícím přechodem označeným řetězcem s_i z uzlu *b*. Tento uzel *a* reprezentuje jazyk $A=B\cdot s_i$ a jeho předchůdce uzel *b* reprezentuje jazyk *B*. Nyní v jakémkoliv kroku *j* musí proměnná *E[a, j]* obsahovat nejmenší vzdálenost mezi libovolným řetězcem $s_a \in A$ a příponou *x* z *T[1..j].* Zjevně musí platit $s_a=s_b\cdot s_i$, kde $s_b \in B$. Dále můžeme *x* zapsat jako $x=x_1\cdot x_2$, kde $\delta(x,s_a)=\delta(x_1,s_b)+\delta(x_2,s_i)$, a kde s_b musí být řetězcem v *B* s nejmenší vzdáleností k libovolné příponě $T[1..j\text{-}|x_2|]$ a tato přípona musí být x_1. To znamená, že $E[b,j\text{-}|x_2|]=\delta(x_1,s_b)$. A aby byla vzdálenost $\delta(x,s_a)$ nejvýše *k*, musí platit $\delta(x_1,s_b)\leq k$ a zároveň $\delta(x_2,s_i)\leq k$. Tudíž $-k\leq |x_2|-|s_i|\leq +k$ a $\delta(x_2,s_i)$ je vypočítána algoritmem [4] pro všechna možná x_2 délky v rozmezí $[s_i\text{-}k,s_i+k]$. V algoritmu si poté můžeme všimnout, že počítadla *L-uzlů* jsou nastavena v řádcích 4, 14, 16 a 25, které počítají minimum všech

možných hodnot $\delta(x_1,s_b)+\delta(x_2,s_i)$. Řádky 4, 14 a 25 se starají o případy, kde $x_2=\varepsilon$ a řádek 16 se stará o všechny možné přípony s_2, kde $\delta(x_2,s_i)$ je získáno z vektoru Δ a $\delta(x_1,s_b)$ je získáno z fronty q_i.

Pro správnost *ε-uzlů* stačí poznamenat, že použitý automat a smyčka pro nastavování těchto uzlů jsou naprosto stejné jako v algoritmu [2] a tedy jejich správnost plyne ze správnosti tohoto algoritmu.

Jak můžeme vidět, algoritmus má cyklus *for* s *n* iteracemi, v něm vnořený cyklus *for* s *d* iteracemi a v něm cyklus s maximálně *2k+1* iteracemi. Tím je dokázána složitost tohoto algoritmu $O(kdn)$.

4. Závěr

Nový D. Bellazzouguiův a M. Raffinotův algoritmus kombinuje Myersův a Millerův algoritmus s algoritmem P.Billa a M. Thorupa a řeší tak problém přibližného porovnávání regulárních výrazů s více řetězci v čase $O(kdn)$, který může být v některých případech velkou úsporou oproti Myersovu a Millerovu algoritmu, který pracuje v čase $O(mn)$. Úspora může být o to větší, čím více výskytů "." než "|" a "*" se ve hledaném regulárním výrazu objevuje.

5. Bibliografie

[1] D. Belazzougui, M. Raffinot, Approximate regular expression matching with multi-strings. Journal of Discrete Algorithms, Elsevier, č. 18, 2013, s. 14-21. http://www.sciencedirect.com/science/article/pii/S1570866712001116

[2] E.W. Myers, W. Miller, Approximate matching of regular expressions, Bull. Math. Biol. 51 (1989), s. 7–37.

[3] P. Bille, M. Thorup, Regular expression matching with multi-strings and intervals, in: Proceedings of the Twenty-First Annual ACM–SIAM Symposium on Discrete Algorithms, SODA'10, 2010, s. 1297–1308.

[4] G.M. Landau, E.W. Myers, J.P. Schmidt, Incremental string comparison, SIAM J. Comput. 27 (2) (1998), s. 557–582.