Organizačné poznámky

- DÚ na stránke, termín do stredy pred Veľkou nocou (27.3.)
- Témy nepovinného projektu do konca marca

Overview of string matching algorithms

Algorithm	Worst case	Average case	Comp.	Note
Trivial	$O(nm)$	$O(n+m)$	Y	simple
DFA	$O(n+\sigma m)$	$O(n+\sigma m)$	N	real-time
(K)MP	$O(n+m)$	$O(n+m)$	Y	
Boyer Moore	$O(nm), O(n+m)$	$O(\frac{n}{\sigma}+m)$	N	
Rabin-Karp	$O(nm)$	$O(n+mk)$	N	randomized
Shift-and	$O(n+m+\sigma)$	$O(n+m+\sigma)$	N	m-bit register
BNDM	$O(nm+\sigma)$	$O(n\frac{\log_\sigma m}{m}+m+\sigma)$	N	m-bit register

Multiple patterns: Aho Corasick $O((n+m)\log\sigma+k)$

2D patterns: Baker Bird $O((n+m)\log\sigma)$

Nepovinná domáca úloha

V Boyer-Moorovom algoritme s použitím iba slabého pravidla dobrého sufixu nájdite príklady P a T, ktoré vyžadujú kvadratický počet porovnaní, aj keď sa P v T nenachádza.

Boyer-Moore algorithm 1977

```
1  i=0;
2  while(i<=n-m){
3     j=m-1;
4     while(j>=0 && P[j]==T[i+j]){
5        j--;
6     }
7     if(j<0){ print i; }
8     i += skip;
9  }
```

Good suffix rule: skip $\gamma[j]$:

$$\gamma[j] = m - \max\{k \mid 0 \leq k < m \wedge P[j+1..m-1] \sim P[0..k-1]\}$$

$P \sim Q$ iff P is a suffix Q or Q is a suffix of P

Regular expressions (regulárne výrazy)

- ϵ is a r.e.; $L(\epsilon) = \{\epsilon\}$
- For $a \in \Sigma$, a is a r.e.; $L(a) = \{a\}$
- If R_1 and R_2 are regular expressions,
  - $(R_1.R_2)$ is a r.e.; $L((R_1.R_2)) = L(R_1).L(R_2)$
  - $(R_1|R_2)$ is a r.e.; $L((R_1|R_2)) = L(R_1) \cup L(R_2)$
  - (R_1^*) is a r.e.; $L((R_1)^*) = L(R_1)^* = \cup_{i=0}^{\infty} L(R_1)^i$

Thompson algorithm for simulating NFA

States 0..k-1, 0 - start, k-1 final

Bit array B[0..k-1]; B[i]=1 iff state i is reachable on current prefix of T

```
1   B[0] = 1; B[1..k—1]=(0,0,...);
2   epsilon(B);
3   for(int i=0; i<n; i++) {
4     B2[0..k—1] = 0;
5     foreach transition a—>b labeled T[i] {
6       if(B[a]==1) { B2[b]=1; }
7     }
8     B=B2;
9     epsilon(B);
10    if(B[k—1]) { print occurrence ending at i }
11  }
```

Thompson algorithm for simulating NFA

Handling epsilon transitions

```
1   void epsilon(B) {
2     for(int i=0; i<k; i++) {
3       if(B[i]) { search(B,i); }
4     }
5   }
6   void search(B,i) {
7     foreach epsilon transition i->j {
8       if(! B[j]) {
9         B[j]=1; search(B,j);
10      }
11    }
12  }
```

Cvičenie

Thompsonov algoritmus na simuláciu NKA nám nájde konce výskytov

Čo ak chceme začiatky výskytov?

Čo ak chceme pre každý koniec najbližší alebo najvzdialenejší začiatok?

Regular expressions, summary

- Thompson's algorithm 1968
  - parse R to a tree with m nodes
  - create NFA for $L(R)$ with $\leq 2m$ states and $\leq 4m$ transitions
  - simulate NFA on T in $O(nm)$ time
- DFA: $O(m^2\sigma 2^m + n)$
- Other approaches: hybrid of NFA and DFA, prefiltering, bit parallelism, ...

Patterns with wildcards

- Special character * matches any character from Σ
- E.g. aa*b matches aaab, aabb, aacb,...
- Trivial algorithm $O(nm)$
- Shift-and $O(n + m + \sigma)$ from small m, BNDM good in average case
- Suffix trees $O(nk)$ where k is the number of wildcards
- Algorithm using FFT $O(n \log m)$

Fast Fourier Transform (FFT)

Input: Two polynomials

$A(x) = \sum_{k=0}^{n-1} a_k x^k$

$B(x) = \sum_{k=0}^{n-1} b_k x^k$

Goal: Compute product $C(x) = A(x)B(x)$

$C(x) = \sum_{k=0}^{2n-2} c_k x^k$

$c_k = \sum_{j=0}^{n-1} a_j b_{k-j}$

(define $b_j = 0$ if $j < 0$ or $j \geq n$)

Trivial algorithm: $O(n^2)$

FFT: $O(n \log n)$