10. Konečný automat. Vyhledávání, řazení.

Procedurální programování

Stanislav Vítek

Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Přehled témat

Část I

Konečný automat

Konečný automat

Konečný automat = výpočetní model, primitivní počítač

- Řídící jednotka s konečným počtem stavů
- Vstupní posloupnost — textový řetězec, fyzické vstupy
- Výstupní posloupnost, fyzické výstupy

Definice:

- konečná množina stavů Q
- počáteční stav $q_0 \in Q$
- konečná množina vstupních symbolů A
- přechodová funkce $f : Q \times A \to Q$

$$q_{t+1} = f(q_t, a_t)$$

t je čas nebo index do vstupní posloupnosti

- (někdy) množina koncových stavů $F \subseteq Q$
- (někdy) množina akcí G a výstupní funkce $g : Y \to G$ nebo $g : Y \times A \to G$

Konečný automat

Konečný automat = výpočetní model, primitivní počítač

- Řídící jednotka s konečným počtem stavů
- Vstupní posloupnost — textový řetězec, fyzické vstupy
- Výstupní posloupnost, fyzické výstupy

Definice:

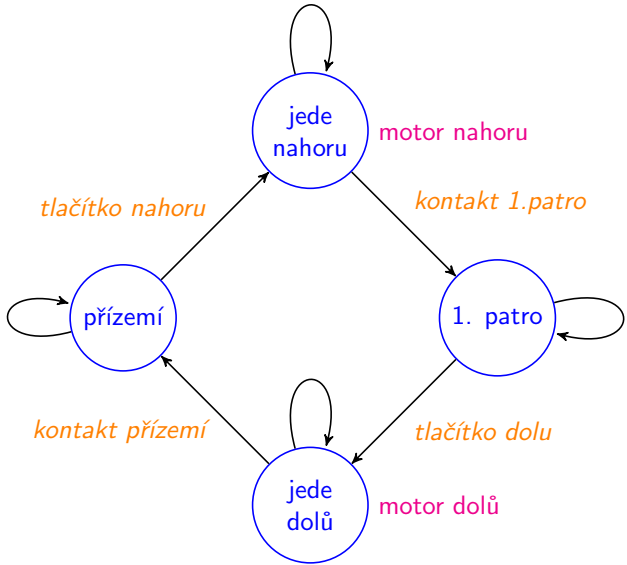
- konečná množina stavů Q
- počáteční stav $q_0 \in Q$
- konečná množina vstupních symbolů A
- přechodová funkce $f : Q \times A \to Q$

$$q_{t+1} = f(q_t, a_t)$$

t je čas nebo index do vstupní posloupnosti

- (někdy) množina koncových stavů $F \subseteq Q$
- (někdy) množina akcí G a výstupní funkce $g : Y \to G$ nebo $g : Y \times A \to G$

Příklad: řízení výtahu – stavový diagram



Příklad: řízení výtahu – přechodová a výstupní tabulka

vstupy				stavy		výstupy	
Tl.d.	Tl.n.	Příz.	1.p.	q_t	q_{t+1}	↑	↓
?	0	?	?	přízemí	přízemí	0	0
?	1	?	?	přízemí	jede nahoru	1	0
?	?	?	0	jede nahoru	jede nahoru	1	0
?	?	?	1	jede nahoru	1. patro	0	0
0	?	?	?	1. patro	1. patro	0	0
1	?	?	?	1. patro	jede dolů	0	1
?	?	0	?	jede dolů	jede dolů	0	1
?	?	1	?	jede dolů	přízemí	0	0

Automat přijímající jazyk

Jazyk = (i nekonečná) množina řetězců.

Rozhodovací konečný automat

- Vstupem je řetězec.
- Dává odpověď ano/ne, zda vstup patří do jazyka.
- V každém kroku čte automat jeden znak z řetězce.
- Vstupní abeceda A jsou všechny přípustné znaky.
- Automat přijímá řetězec $\Leftrightarrow$ na konci řetězce je automat v *přijímajícím (koncovém) stavu.*

I. Konečný automat

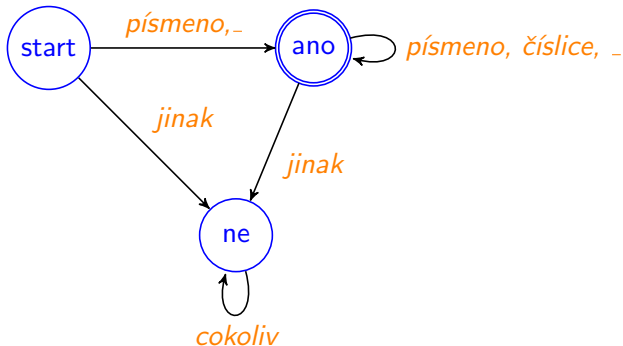
Příklad 10.1 Jméno proměnné

Příklad 10.2 Přeskakování komentářů

Příklad 10.3 Lexikální analýza textu

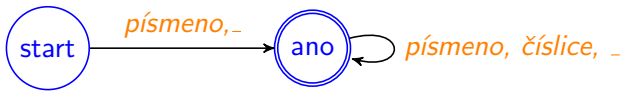
10.1 Jméno proměnné – totální automat

Automat přijímá platná jména proměnných v C.



- Automat je *totální* — přechodová funkce f je definována pro všechna $Q \times A$.

10.1 Jméno proměnné – částečný automat



- Částečný automat – nedefinovaný přechod znamená nepřijetí.

10.1 Jméno proměnné — implementace

```
#define is_let(x) \\
  ((x >= 'a' && x <= 'z' || x >= 'A' && x <= 'Z') ? 1 : 0)
#define is_num(x) ((x >= '0' && x <= '9') ? 1 : 0)

typedef enum {START, ANO, NE} state_t;

state_t eval(char * str) {
  state_t state = START;
  while (*str != 0) {
    if (state == START)
      if (*str == '_' || is_let(*str) == 1) state = ANO;
      else return NE;
    else
      if (*str != '_' && is_let(*str) == 0 && is_num(*str) == 0)
      return NE;
    str++;
  } return ANO; }
```

Transformace textu

Transformační konečný automat

- Vstupem je řetězec.
- Výstupem je řetězec.
- V každém kroku čte automat jeden znak z řetězce.
- Vstupní abeceda A jsou všechny přípustné znaky.
- Součásti každého přechodu (nebo stavu) může být výstup jednoho či více znaků.

I. Konečný automat

Příklad 10.1 Jméno proměnné

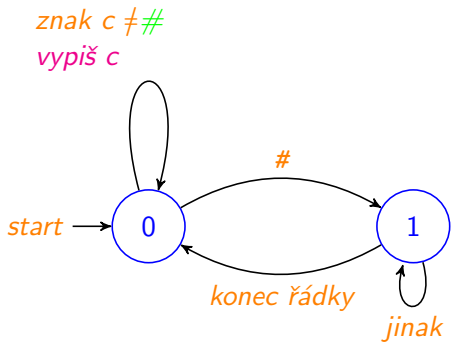
Příklad 10.2 Přeskakování komentářů

Příklad 10.3 Lexikální analýza textu

10.2 Přeskakování komentářů

Problém

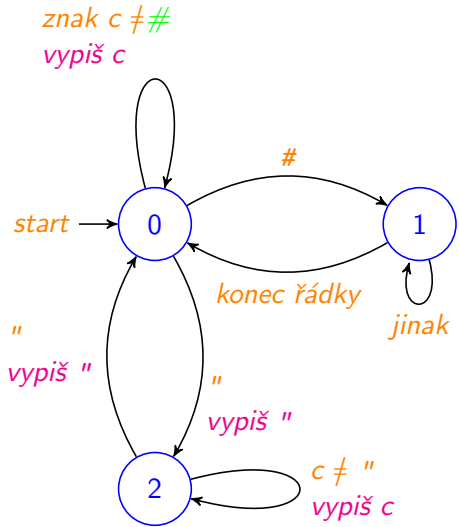
- Vstupem je textový soubor
- Komentář je vše od znaku # (včetně) do konce řádky
- Vypište obsah souboru bez komentářů.



10.2 Přeskakování komentářů – vylepšení

Vylepšení

Verze, kterou nezmate # v řetězci.



I. Konečný automat

Příklad 10.1 Jméno proměnné

Příklad 10.2 Přeskakování komentářů

Příklad 10.3 Lexikální analýza textu

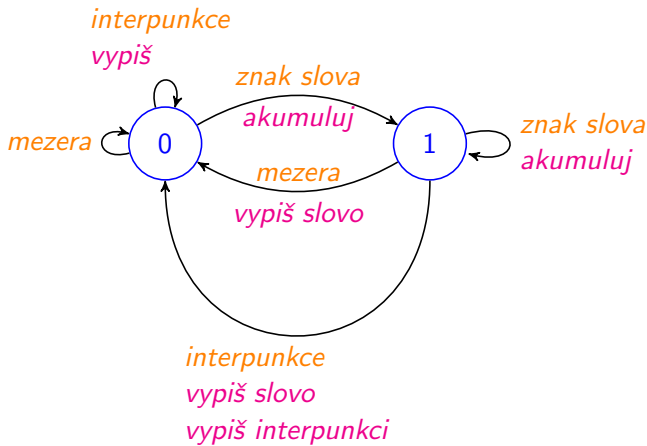
10.3 Lexikální analýza textu (parsing) 1/2

- Vstup: řetězec (posloupnost znaků)
- Výstup: posloupnost symbolů *(tokens)* – číslo, operátor, identifikátor, klíčové slovo, ...
  - Symboly mohou mít atributy — řetězec, hodnota, ...
  - Reprezentujeme jako dvojice — typ symbolu + atribut
- Další operace
  - Vynechání komentářů
  - Odstranění mezer
  - Chybová hlášení

Problém

Rozdělte řetězec na posloupnost symbolů:

- *Interpunkce* – `.` `,` `;` `:` `?` `!` `"` `'` `(` `)` (jeden znak)
- *Slovo* – řetězec znaků, neobsahující interpunkci, mezery ani konce řádků.
- Konce řádků, mezery a tabulátory *(whitespace)* ignorujte.



Počáteční stav $q_0 = 0$.

Nedeterministický automat

- Přechodová funkce vrací množinu stavů, $f : Q \times A \to 2^Q$, automat si jeden nedeterministicky 'vybere'.
- Slovo je přijímáno automatem, pokud existuje posloupnost výběrů vedoucí do koncového stavu.
- Implementace:
  - Při simulaci si udržujeme množinu možných stavů.
  - Automaticky převedeme na deterministický automat s 2^n stavy.
  - Často lze nalézt ekvivalentní deterministický automat s méně stavy.

- Automat přijímající slova končící na *"ce"*
- Automat přijímající klíčová slova Pythonu.
- Automat přijímající celá nebo reálná čísla.

Nedeterministický automat

- Přechodová funkce vrací množinu stavů, $f : Q \times A \to 2^Q$, automat si jeden nedeterministicky 'vybere'.
- Slovo je přijímáno automatem, pokud existuje posloupnost výběrů vedoucí do koncového stavu.
- Implementace:
  - Při simulaci si udržujeme množinu možných stavů.
  - Automaticky převedeme na deterministický automat s 2^n stavy.
  - Často lze nalézt ekvivalentní deterministický automat s méně stavy.

- Automat přijímající slova končící na *"ce"*
- Automat přijímající klíčová slova Pythonu.
- Automat přijímající celá nebo reálná čísla.

Část II

Vyhledávání

Vyhledání čísla v řadě

Problém

- Myslím si přirozené číslo X mezi 1 a 1000.
- Možné otázky:
  - Je X menší než N?
  - Kolik otázek potřebujete na odhalení čísla?
  - Mezi kolika čísly jste schopni odhalit skryté číslo na K otázek?

Vyhledání čísla v řadě

Problém

- Myslím si přirozené číslo X mezi 1 a 1000.
- Možné otázky:
  - Je X menší než N?
  - Kolik otázek potřebujete na odhalení čísla?
  - Mezi kolika čísly jste schopni odhalit skryté číslo na K otázek?

Řešení

- Metoda půlení intervalu
- K otázek: rozlišíme mezi 2^K čísly
- N čísel: potřebujeme $log_2 N$ otázek

Vyhledání čísla v řadě

Problém

- Myslím si přirozené číslo X mezi 1 a 1000.
- Možné otázky:
  - Je X menší než N?
  - Kolik otázek potřebujete na odhalení čísla?
  - Mezi kolika čísly jste schopni odhalit skryté číslo na K otázek?

Řešení

- Metoda půlení intervalu
- K otázek: rozlišíme mezi 2^K čísly
- N čísel: potřebujeme $log_2 N$ otázek

Vyhledávání v (připravených) datech je častý problém: web, slovník, ...

Konkrétní problém

Problém

- Mějme posloupnost (pole) $a_0, \ldots, a_{N-1}$
- Mějme hodnotu q
- Úkol: zjistit, zda existuje $a_i = q$

Varianty

- Výstup: ano/ne nebo pozice
- Hledání opakujeme mnohokrát pro stejné a
  - předzpracování
- Posloupnost a je setříděná.
- Stochastické hledání

Vyhledávání a logaritmus

Naivní metoda – průchod seznamu

- lineární vyhledávání, složitost $O(n)$
- pomalé (viz např. databáze s milióny záznamů)
- jen velmi krátké seznamy

Rozumná metoda – půlení intervalu

- logaritmický počet kroků (vzhledem k délce seznamu)
- složitost $O(log(n))$

II. Vyhledávání

Lineární vyhledávání

Binární vyhledávání

Lineární vyhledávání

- Procházíme postupně `a` než narazíme na `q`

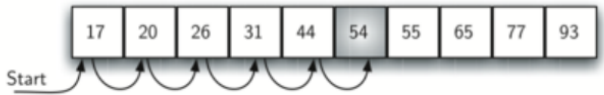


Image from Miller & Ranum: Problem solving with algothms and data structures

Lineární vyhledávání

- Procházíme postupně `a` než narazíme na `q`
- **Počet porovnání:**

	nejméně	nejvíce	průměrně
$q \notin a$	N	N	N
$q \in a$	1	N	N/2

  - Složitost $O(N)$, kde `N=len(a)`.
  - Rychleji to nejde, protože na každý prvek a_i se musíme podívat.

II. Vyhledávání

Lineární vyhledávání

Binární vyhledávání

Binární vyhledávání

Vyhledávání v setříděné posloupnosti

- Mějme posloupnost (pole) $a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_{N-1} \leq a_N$
- Mějme hodnotu q
- Úkol: zjistit, zda existuje $a_i = q$
- Je vyhledávání v setříděném poli rychlejší?

Binární vyhledávání

Hlavní myšlenky

- Postupně zmenšujeme interval indexů, kde by mohlo ležet q
- V každém kroku porovnáme q s prostředním prvkem intervalu a podle toho zvolíme jeden z podintervalů.
- Skončíme, pokud je prvek nalezen, nebo pokud je podinterval prázdný.

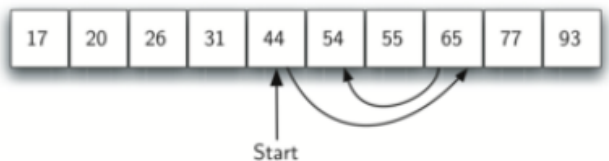


Image from Miller & Ranum: Problem solving with algothms and data structures

10.4 Rekurzivní implementace binárního vyhledávání

```
1  int binarySearch(int arr[], int low, int high, int x)
2  {
3    if (high >= low) {
4      int mid = low + (high - low) / 2;
6    // hledane cislo je uprostred intervalu
7    if (arr[mid] == x)
8      return mid;
10   // hledane cislo je mensi nez stred -> leva strana pole
11   if (arr[mid] > x)
12     return binarySearch(arr, low, mid - 1, x);
14   // jinak -> prava strana pole
15   return binarySearch(arr, mid + 1, high, x);
16   }
17   return -1; // prvek v poli neni
18 }
```

Část III

Řazení / Třídění

Úvod

Terminologická poznámka

- anglicky "sorting algorithms"
- česky používáno: řadicí algoritmy nebo třídicí algoritmy
- řadicí vesměs považováno za "správnější"

Proč se tím zabývat

- procvičení práce se seznamy
- ilustrace algoritmického myšlení, technik návrhu algoritmů
- typický příklad drobné změny algoritmu s velkým dopadem na rychlost programu

Doporučené zdroje

- http://www.sorting-algorithms.com/
  - animace
  - kódy
  - vizualizace
- více podobných: Google → sorting algorithms
- a na zpestření:
  - xkcd Ineffective Sorts: https://xkcd.com/1185/
  - Bubble-sort with Hungarian folk dance: http://www.youtube.com/watch?v=lyZQPjUT5B4

Třídění

- Mějme posloupnost (pole) $A = \left[a_0, \ldots, a_{N-1}\right]$ a relaci '$\leq$'
- Najděte takovou permutaci $B = \left[b_0, \ldots, b_{N-1}\right]$ pole A, aby $b_0 \leq b_1 \leq \cdots \leq b_{N-1}$.

Poznámky:

- Formy výstupu:
  - Třídění na místě *(in place)*
  - Vytvoření nového pole B, pole A zůstává nezměněno.
  - Najdeme indexy $j_1, j_2, \ldots, j_N$, tak aby $b_i = a_{j_i}$ (`a[j[i]]`)
- Stabilní třídění — zachovává pořadí ekvivalentních prvků.

III. Řazení / Třídění

Třídění probubláváním (bubble sort)

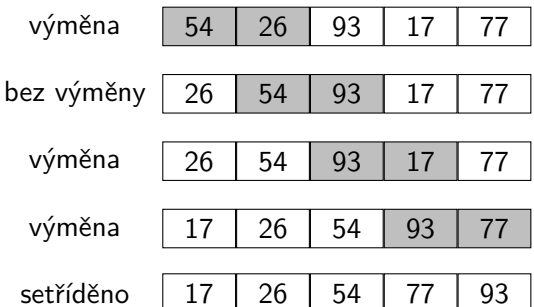
Třídění zatřiďováním (insertion sort)

Třídění výběrem (selection sort)

Rychlé třídění (quick sort)

Řazení probubláváním – bubble sort

- Vyměňuje sousední prvky ve špatném pořadí
- Znázorněn jeden průchod



Implementace algoritmu bubble sort

```
1  #define swap(x,y) {x = x + y; y = x - y; x = x - y;}
3  void bubble_sort(int * arr, int n) {
4    int i, j, swapped;
5    for (i = 0; i < n - 1; i++) {
6      swapped = 0;
7      for (j = 0; j < n - i - 1; j++) {
8        if (arr[j] > arr[j + 1]) {
9          swap(*(arr + j), *(arr + j + 1));
10         swapped = 1;
11       }
12     }
13     if (swapped == 0) break;
14   }
15 }
```

III. Řazení / Třídění

Třídění probubláváním (bubble sort)

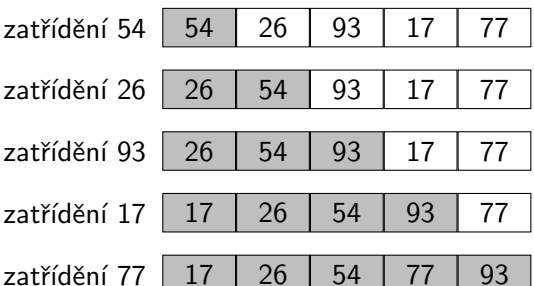
Třídění zatřiďováním (insertion sort)

Třídění výběrem (selection sort)

Rychlé třídění (quick sort)

Třídění zatřiďováním – insertion sort

- Prvek a_i zatřídíme do již setříděných $a_0, \ldots, a_{i-1}$
- Setříděnou posloupnost na začátku tvoří jediný prvek a_0



Implementace

```
1  void insertion_sort(int * arr, int n)
2  {
3    for (int i = 1; i < n; ++i) {
4      int key = arr[i];
5      int j = i - 1;
7      /* Move elements of arr[0..i-1], that are
8      greater than key, to one position ahead
9      of their current position */
10     while (j >= 0 && arr[j] > key) {
11       arr[j + 1] = arr[j];
12       j = j - 1;
13     }
14     arr[j + 1] = key;
15   }
16 }
```

III. Řazení / Třídění

Třídění probubláváním (bubble sort)

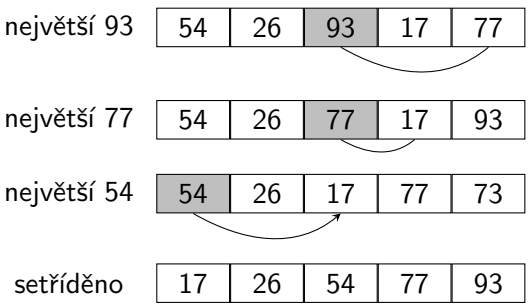
Třídění zatřiďováním (insertion sort)

Třídění výběrem (selection sort)

Rychlé třídění (quick sort)

Třídění výběrem – selection sort

- Vybere maximum mezi $a_0, \ldots, a_{N-1}$, to umístí do a_{N-1}.
- Vybere maximum mezi $a_0, \ldots, a_{N-2}$, to umístí do a_{N-2}...



Implementace

```
1  void selection_sort(int arr[], int n) {
2    for (int i = 0; i < n - 1; i++) {
4      // Assume the current position holds
5      // the minimum element
6      int min = i;
8      // Iterate through the unsorted portion
9      // to find the actual minimum
10     for (int j = i + 1; j < n; j++) {
11       if (arr[j] < arr[min_idx]) {
12
13         // Update min if a smaller element is found
14         min_idx = j;
15       }
16     }
18     // Move minimum element to its correct position
```

III. Řazení / Třídění

Třídění probubláváním (bubble sort)

Třídění zatřiďováním (insertion sort)

Třídění výběrem (selection sort)

Rychlé třídění (quick sort)

Rychlé řazení – implementace ve standardní knihovně C

```c
#include <stdio.h>
#include <stdlib.h>

int comp(const void *a, const void *b) {
  return (*(int *)a - *(int *)b);
}

int main() {
  int arr[] = {5, 2, 3, 1, 4};
  int n = sizeof(arr) / sizeof(arr[0]);

  qsort(arr, n, sizeof(arr[0]), comp);

  for (int i = 0; i < n; i++)
    printf("%i ", arr[i]);
  return 0;
}
```