Úvod do informatiky

přednáška desátá

Miroslav Kolařík

Zpracováno dle

R. Bělohlávek, V. Vychodil: Diskrétní matematika 2,
http://phoenix.inf.upol.cz/esf/ucebni/DM2.pdf

P. Martinek: Základy teoretické informatiky,
http://phoenix.inf.upol.cz/esf/ucebni/zti.pdf

http://cs.wikipedie.org/wiki/

Obsah

Obsah

Rekurze znamená sebeopakování. Velmi často se používá v matematice a informatice.

V programování rekurze představuje opakované vnořené volání stejné funkce (podprogramu); hovoříme o tzv. rekurzivní funkci. Nedílnou součástí rekurzivní funkce musí být ukončující podmínka určující, kdy se má rekurze zastavit. Po každém kroku volání sebe sama musí dojít ke zjednodušení problému. Pokud nenastane koncová situace, provede se rekurzivní krok.

Poznámka: Každý algoritmus využívající rekurzi lze přepsat do nerekurzivního tvaru při použití zásobníku.

Rozlišujeme dva základní typy dělení rekurze:

I. typ

1. **přímá rekurze** – nastává pokud podprogram volá přímo sám sebe
2. **nepřímá rekurze** – je situace, kdy vzájemné volání podprogramů vytváří "kruh"; např. v příkazové části funkce *A* je volána funkce *B*, ve funkci *B* voláme funkci *C*, která volá funkci *A*

II. typ

1. **lineární rekurze** – nastává pokud podprogram při vykonávání svého úkolu volá sama sebe pouze jednou – vytváří se takto lineární struktura postupně volaných podprogramů
2. **stromová rekurze** – nastává pokud se funkce nebo procedura v rámci jednoho vykonávání svého úkolu vyvolá vícekrát (vzniklou strukturu je možné znázornit jako strom)

Humor o rekurzi

Některé definice rekurze v sobě zahrnují prvky humoru a parodie na výkladové slovníky:

Cyklus nekonečný
: viz Nekonečný cyklus

Nekonečný cyklus
: viz Cyklus nekonečný

Tyto žerty v sobě nesou ponaučení. Zde je evidentní absence ukončující podmínky, která je nedílnou součástí rekurze a bez níž program často vede na nekonečný cyklus.

Ukázkou vhodného užití rekurze je rekurzivní algoritmus Quicksort nebo třeba průchod stromem nebo vykonávání určité operace se všemi soubory v adresářové struktuře na pevném disku (neboť na každý nalezený podadresář lze automaticky zavolat stejnou funkci).

Poznámka: Zajímavou oblastí použití rekurze jsou fraktály. Fraktály jsou soběpodobné útvary, které mají na první pohled velmi složitý tvar, přestože jsou generovány opakovaným použitím jednoduchých pravidel.

Příklad

Nejčastějším příkladem rekurzivního postupu je výpočet faktoriálu $N!$, který lze pro $N \in \mathbb{N}_0$ spočítat dle vztahu $N! = N \cdot (N-1)!$, přičemž $0! = 1$.

faktorial(N):
    pokud $N <= 0$, potom vysledek $= 1$,
    jinak vysledek $= N *$ faktorial $(N-1)$

Poznámka: Nerekurzivní řešení dané úlohy bývá efektivnější, avšak většinou ztrácí na přehlednosti. Vhodnou metodou může být iterace:

Příklad

faktorial(N):
    pokud $N < 0$, potom konec,
    jinak
        vysledek $= 1$,
        pro i od 1 do N proveď
            vysledek $=$ vysledek $* i$,
konec

Poznámka: Rekurze je pro programátora velmi silný a užitečný nástroj, je to však také nástroj nebezpečný a při jeho použití musíme být opatrní. Mechanické použití rekurze vede totiž často k algoritmům s exponenciální časovou složitostí, které jsou velice pomalé a prakticky použitelné jen pro malé vstupy.

Fibonacciho posloupnost je dobrou ukázkou, jak lze řešit rekurzivní úlohu různými a různě efektivními způsoby.

Příklad

Určete N-té Fibonacciho číslo pro dané $N \geq 0$, je-li $\mathrm{Fib}(0) = 0$, $\mathrm{Fib}(1) = 1$ a $\mathrm{Fib}(N) = \mathrm{Fib}(N-1) + \mathrm{Fib}(N-2)$ pro $N > 1$.
Nekonečná posloupnost Fibonacciho čísel tedy začíná čísly
0 1 1 2 3 5 8 13 21 34 55 89 144 ...
Toto řešení má však exponenciální časovou složitost a provádí se v něm velké množství zbytečných výpočtů.

Pro zásadní zrychlení výpočtu postačí, jestliže rekurzivní algoritmus doplníme o jedno pomocné pole F. Do něj si budeme průběžně ukládat všechna již jednou spočítaná Fibonacciho čísla. Před každým rekurzivním voláním se podíváme, zda by takové volání nebylo zbytečné, zda hledanou hodnotu již nemáme v poli F. Výsledný algoritmus má rázem lineární časovou složitost.

Poznámka: Úlohu nalézt N-té Fibonacciho číslo je možné řešit také bez použití rekurze a bez pomocného pole, přes dvě proměnné, ve kterých uchováváme poslední dvě hodnoty Fibonacciho posloupnosti. (Paměťová složitost je pak konstantní.)

Obsah

Graf je grafické vyjádření vztahů mezi nějakými objekty. Objekty jsou v grafu reprezentovány kroužky – nazýváme je **vrcholy** (**uzly**) **grafu**. Vztah mezi dvěma objekty je v grafu zobrazen čárou, tzv. **hranou**, která spojuje vrcholy, které objekty reprezentují. Přitom každý z obou konců hrany musí vycházet z vrcholu; speciálním případem je hrana, jejíž oba dva konce vycházejí ze stejného vrcholu, tzv. **smyčka**. Hrany mohou být orientované nebo neorientované (či vícenásobné).

Definice

Neorientovaný graf je dvojice $G = \langle V, E \rangle$, kde V je neprázdná množina **vrcholů** a $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ je množina dvouprvkových množin vrcholů, tzv. (**neorientovaných**) **hran**.

Definice

Orientovaný graf je dvojice $G = \langle V, E \rangle$, kde $V \neq \emptyset$ je množina **vrcholů** a $E \subseteq V \times V$ je množina uspořádaných dvojic vrcholů, tzv. (**orientovaných**) **hran**.

Definice

Sled v neorientovaném grafu $G = \langle V, E \rangle$ je posloupnost $v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$, kde $v_i \in V$ jsou vrcholy a $e_j \in E$ jsou hrany takové, že $e_i = \{v_{i-1}, v_i\}$ pro $i = 1, \ldots, n$. Číslo n se nazývá **délka sledu**. Sled $v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$, se nazývá

- **uzavřený**, je-li $v_0 = v_n$,
- **tah**, neopakuje-li se v něm žádná hrana ($i \neq j \Rightarrow e_i \neq e_j$),
- **cesta**, neopakuje-li se v něm žádný vrchol ($i \neq j \Rightarrow v_i \neq v_j$),
- **kružnice**, je-li $v_0 = v_n$ a s výjimkou vrcholů v_0 a v_n jsou každé dva vrcholy různé.

Vzdálenost z vrcholu u do vrcholu v je délka cesty z u do v, která má ze všech cest z u do v délku nejmenší.

Definice

Hranové ohodnocení grafu $G = \langle V, E \rangle$ s množinou hodnot D je funkce $w : E \to D$.

Poznámka: Podobně jednoduše lze zavést i pojem ohodnocení vrcholů. My jej ale nebudeme potřebovat.

Dále si ukážeme dva grafové algoritmy: Dijsktrův a Kruskalův.

Dijkstrův algoritmus

Dijkstrův algoritmus je nejznámější algoritmus na nalezení nejkratších cest z daného vrcholu.

Vstup: neorientovaný graf $G = \langle V, E \rangle$, hranové ohodnocení $w : E \to \mathbb{R}^+$, vrchol $s \in V$

Výstup: hodnota $d(v)$ pro každý $v \in V$, $d(v)$ je hodnota nejkratší cesty z s do v

Proměnné: funkce $d : V \to \mathbb{R}_0^+$, číslo $m \in \mathbb{R}_0^+$, množiny $A, N \subseteq V$

Algoritmus:

1. $A := V$; $d(s) := 0$; pro $v \in V - \{s\}$: $d(v) := \infty$
2. pokud neexistuje $v \in A$ takový, že $d(v) \neq \infty$, skonči
3. $m := \min\{d(v) \mid v \in A\}$; $N := \{v \in A \mid d(v) = m\}$; $A := A - N$
4. pro všechny $v \in N$, $u \in A$ takové, že $\{v, u\} \in E$: jestliže $d(v) + w(\{v, u\}) < d(u)$, pak $d(u) := d(v) + w(\{v, u\})$; pokračuj bodem 2.

Definice

Neorientovaný graf $G = \langle V, E \rangle$ se nazývá **souvislý**, právě když $\forall u, v$ existuje sled z u do v.

Definice

Strom je neorientovaný souvislý graf bez kružnic.

Definice

Neorientovaný graf $\langle V_1, E_1 \rangle$ je **podgrafem** grafu $\langle V_2, E_2 \rangle$, právě když $V_1 \subseteq V_2$ a $E_1 \subseteq E_2$.

Definice

Kostra neorientovaného grafu G je jeho podgraf, který je stromem a obsahuje všechny vrcholy grafu G.

Poznámka: Graf má kostru, právě když je souvislý.

Kruskalův algoritmus

Kruskalův algoritmus je hladový algoritmus (z roku 1956) hledající minimální kostru souvislého grafu $G = \langle V, E \rangle$.

Vstup: souvislý graf $G = \langle V, E \rangle$, $|V| = n$, $|E| = m$, hranové ohodnocení $w : E \to \mathbb{R}^+$

Výstup: minimální kostra grafu, tj. kostra grafu s minimálním součtem ohodnocení jejích hran

Algoritmus: Algoritmus postupně prochází hrany a vytváří z nich množiny $E_0, E_1, E_2, \ldots$, z nichž poslední je minimální kostra grafu $G = \langle V, E \rangle$:

1. uspořádej hrany z E dle váhy: $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
2. vytvoř prázdnou množinu hran E_0
3. pokud přidáním e_i vznikne kružnice, ponechej $E_i := E_{i-1}$, jinak $E_i := E_{i-1} \cup \{e_i\}$
4. opakuj krok 3, dokud $i < m$ a E_i ještě nemá $n-1$ hran.

Obsah

Konečný automat je teoretický výpočetní model používaný v informatice pro studium vyčíslitelnosti a obecně formálních jazyků. Popisuje velice jednoduchý počítač, který může být v jednom z několika stavů, mezi kterými přechází na základě symbolů, které čte ze vstupu. Množina stavů je konečná (odtud název). Konečný automat nemá žádnou další paměť kromě informace o aktuálním stavu. Konečné automaty se používají pro zpracování regulárních výrazů, například jako součást lexikálního analyzátoru v překladačích.

Definice

Deterministický konečný automat je pětice $\mathscr{A} = \langle T, Q, \delta, q_0, F \rangle$, kde

- T je konečná množina vstupních symbolů (nazývaná **abeceda**)
- Q je konečná množina stavů
- δ je tzv. **přechodová funkce**, $\delta : Q \times T \to Q$, popisující pravidla přechodů mezi stavy
- q_0 je **počáteční stav**, $q_0 \in Q$
- F je množina **koncových stavů**, $F \subseteq Q$.

Popis činnosti automatu:
Na počátku se automat nachází v definovaném počátečním stavu q_0. Dále v každém kroku přečte jeden symbol ze vstupu a přejde do stavu, který je dán přechodovou funkcí (aktuálním stavem a přečteným symbolem). Poté pokračuje čtením dalšího symbolu ze vstupu, dalším přechodem dle přechodové funkce atd.
Skončí-li automat po přečtení vstupu v koncovém stavu, pak daný vstup přijal (rozpoznal). Jinak ho nepřijal.

Poznámka: Existují i tzv. **nedeterministické konečné automaty** s přechodovou funkcí $\delta : Q \times T \rightarrow 2^Q$. Lze o nich dokázat, že mají stejnou výpočetní sílu jako deterministické konečné automaty.

Konečný automat lze reprezentovat:

- přechodovou funkcí *delta*
- tabulkou
- (mírně modifikovaným) orientovaným grafem – stavy jsou vrcholy, přechody jsou vyznačeny hranami, které jsou ohodnocené vstupními symboly, koncové stavy jsou vyznačeny dvojitým kroužkem.

Příklad

Reprezentujte (přechodovou funkcí δ, tabulkou, orientovaným grafem) DKA rozpoznávající všechna binární čísla, v nichž nejsou za sebou více než dvě nuly a nekončí dvěma nulama. (Automat tedy například přijímá vstupní slovo 1101001, ale nepřijímá vstupní slovo 100.)

Řešení: Jednoduché.

Definice

Řetězec α prvků množiny V je libovolná konečná posloupnost této množiny. Počet prvků v řetězci je jeho **délkou** a označuje se $|\alpha|$. Speciálně pro **prázdný řetězec** ε definujeme $|\varepsilon| = 0$.

Příklad

Nechť $V = \{0, 1\}$. Pak pro $\alpha = 001011$ a $\beta = 0000$ je $|\alpha| = 6$ a $|\beta| = 4$.

Definice

Abeceda V je konečná množina symbolů.

Definice

Pozitivní uzávěr V^+ konečné množiny V je množina všech neprázdných řetězců prvků množiny V. **Uzávěr** V^* je množina $V^+ \cup \{\varepsilon\}$.

Definice

Nechť je dána abeceda V. Pak libovolná podmnožina uzávěru V^* je **formální jazyk** nad V.

Příklad

Pro $V = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ je zřejmě množina celých čísel formální jazyk nad V.

Příklad

Nechť $T = \{a, b, c\}$. Sestavte konečný automat, který rozpozná slova, která obsahují podřetězec *abc*.

Řešení: Jednoduché.

Příklad

Nechť $T = \{0, 1\}$. Sestavte konečný automat, který přijímá regulární jazyk řetězců, které vyjadřují binární číslo dělitelné třemi (beze zbytku).

Řešení: Jednoduché.

KA – shrnutí

Počáteční podmínky: na vstupu je nějaké slovo $w \in T^*$, automat je ve stavu q_0.

V každém kroku: Odebere se nejlevější symbol x ze vstupního slova w a z aktuálního stavu q se přejde do stavu $\delta(q, x)$ (dle přechodové funkce).

Ukončení činnosti KA: Buď je vstupní slovo celé zpracováno, nebo pro aktuální stav q a odebraný symbol x neexistuje funkční hodnota $\delta(q, x)$, tj. není možný další přechod.

Slovo w je **přijímáno** KA, jestliže je celé zpracováno a KA skončil v koncovém stavu.

Definice

Množinu všech slov $w \in T^*$ přijímaných KA nazveme **jazykem** rozpoznávaným tímto automatem.