9 Řešení rekurentních problémů

Kreslíková, 13. 10. 2023

9.1 Úvod

V předchozích kapitolách jsme se seznámili s různými přístupy, jak vyjadřovat algoritmy a specificky jsme se seznámili se syntaxí a sémantikou jazyka C. V této a následujících kapitolách se budeme zabývat řešením různých druhů problémů a budeme je řešit za pomoci programů v jazyce C. Okruhy problémů, které jsme vybrali, zahrnují použití různých algoritmizačních technik od iteračních algoritmů, přes rekurzi, až po dynamické datové struktury a vytváření abstraktních datových typů.

Tato kapitola se zabývá řešením rekurentních problémů. Nezaměňujte je prosím s problémy rekurzivními. Rekurentní problémy se obvykle řeší pomocí iteračních algoritmů a použití rekurze je zde sice možné, ale většinou vzhledem k charakteru těchto úloh nežádoucí. Tento typ úloh se využívá při výpočtu funkčních hodnot matematických funkcí (např. sinus, logaritmus, exponenciální funkce). V některých procesorech je aproximace těchto funkcí řešena hardwarově, ale je užitečné znát postup, jak je vypočítat pouze pomocí základních aritmetických operací. Existují totiž procesory, které to neumí a ne vždy máme k dispozici knihovnu matematických funkcí.

Při řešení tohoto problému narazíme na technické problémy, které bude třeba vyřešit, aby naše řešení bylo efektivní. Seznámíme se zde proto také s pojmem heuristika.

9.2 Rekurentní vztahy

Při programování se často setkáváme se situací, kdy řešený problém vede na iteraci, kde každý krok iterace závisí na výsledku z předchozího kroku. Takovýmto problémům říkáme rekurentní problémy. Typickým příkladem je počítání s nekonečnými posloupnostmi čísel a jejich součty či součiny.

Naivním přístupem k řešení těchto problémů lze často vytvořit sice funkční, ale velmi neefektivní program. U problémů, kde potřebujeme efektivní program, může být problematické i použití rekurze (tu probereme v následující kapitole). V této kapitole se podíváme na matematickou podstatu rekurentních problémů a ukážeme si, jak pro tuto třídu problémů vytvořit efektivní algoritmy.

Rekurentní vztah je rovnice nebo nerovnice, kde je hodnota funkce pro větší argumenty vyjádřena pomocí hodnot, kterých funkce nabývá pro menší argumenty.
Rekurentní vztah můžeme napsat ve tvaru:

$$Y_{i+1} = F(Y_{i-k}, \ldots, Y_{i-2}, Y_{i-1}, Y_i)$$

$Y_{i-k}, Y_{i-k+1}, \ldots, Y_i, Y_{i+1}$ – zevšeobecněné proměnné,
F – funkce na základě které získáme z hodnot, $Y_{i-k}, \ldots, Y_{i-2}, Y_{i-1}, Y_i$ hodnotu Y_{i+1}.
$k \geq 0$ – počet předchozích kroků řešení, které bereme v úvahu. Často bereme v úvahu všechny předchozí kroky, ale není to vždy nutné.

Vlastnosti:
Pro výpočet další hodnoty potřebujeme pouze $k+1$ posledních hodnot. Musí existovat takové n, že Y_n je požadovanou hodnotou, po jejímž získání iterační výpočet končí. Musíme umět rozpoznat požadovanou hodnotu.

9.3 Posloupnosti

Uvažujme rekurentní vztah:

$$Y_{i+1} = F(Y_i)$$

Pro výpočet hodnoty Y_{i+1} je potřeba zjistit hodnotu Y_i. Na začátku musí být dané Y_0, ze kterého celý výpočet začíná. Postupně dostáváme hodnoty $Y_1, Y_2, \ldots, Y_n$, pro které platí:

1. $Y_{i+1} = F(Y_i)$ pro $i \geq 0$
2. $Y_i \neq Y_j$ pro $i \neq j$
3. Y_i pro $i < n$ – nesplňuje podmínky požadované hodnoty.
4. Y_n splňuje podmínky požadované hodnoty.

Jestliže podmínky požadované hodnoty vyjádříme pomocí predikátu $B(Y)$ (podmínky, která závisí na aktuální hodnotě Y), můžeme algoritmus realizující rekurentní vztah $Y_{i+1} = F(Y_i)$ napsat takto:

1. $Y = y_0$
2. while $(\neg B(Y))$
   {
   $Y=F(Y)$
   }

V uvedeném algoritmu jsou použity všeobecné symboly:

- proměnná Y
- predikátový symbol B
- funkční symbol F

Nejde tedy o řešení konkrétního rekurentního vztahu. Algoritmickou konstrukci, ve které symboly proměnných, funkcí a predikátů nejsou interpretované nazýváme ***algoritmické schéma***. Pro konkrétní rekurentní vztah uvedeného charakteru nám stačí interpretace příslušných symbolů. Jestliže iterační výpočty můžeme specifikovat pomocí rekurentních vztahů, můžeme uvedené algoritmické schéma použít na řešení celé řady problémů.

Příklad:Výpočet druhé odmocniny reálného čísla $A \geq 0$ je možné popsat rekurentním vztahem:

$$y_{i+1} = \frac{1}{2}\left(\frac{A}{y_i} + y_i\right)$$

Jako počáteční hodnotu y_0 je možné zvolit pro jednoduchost 1. Zůstává nám stanovit způsob ukončení algoritmu. Ten se obvykle volí tak, že se výpočet opakuje, pokud absolutní hodnota rozdílu dvou po sobě jdoucích hodnot y_i, y_{i+1} není menší než daná hodnota. Této hodnotě se někdy říká přesnost výpočtu. Označujeme ji *eps*. Na výpočet nové hodnoty y potřebujeme jednu předcházející hodnotu. Algoritmus můžeme konstruovat pomocí dvou proměnných *stareY, noveY*.

Algoritmus:

1. zadej A, eps
2. inicializuj stareY (1)
3. vypočítej noveY (podle vzorce)
4. opakuj pokud abs(noveY – stareY)≥ eps
   {
   ulož noveY do stareY
   vypočítej noveY
   }
5. zobraz výsledek

9.4 Řady

Uvažujme řadu vytvořenou z členů:

$t_0, t_1, t_2, \ldots$

Nechť částečné součty členů jsou:

$s_0, s_1, s_2, \ldots$

$s_i = t_0 + t_1 + \ldots + t_i$

Pro částečné součty členů je možné napsat rekurentní vztah:

$s_i = s_{i-1} + t_i$

$s_0 = t_0$

Rekurentní vztah je možné zapsat i pro členy řady:

$t_i = f(t_{i-1})$, pro $i > 0$

Při konstrukci algoritmu je nutné zohlednit rekurentní vztah pro:

- částečné součty,
- členy řady, jejich vzájemný vztah,
- způsob ukončení.

Budeme se zajímat o použití řad na aproximaci funkcí, při které počet členů řady a tedy i počet sčítanců částečné sumy není dopředu známý. Ukončení algoritmu takovýchto rekurentních vztahů je založený na požadavku na přesnost aproximace uvažované funkce.

Ta je vyjádřená buď dosaženou hodnotou částečné sumy nebo přírůstkem, který dostaneme posledním sčítancem. Proto podmínku ukončení vyjádříme predikátem nad dosaženou částečnou sumou a posledním členem řady.

Na základě uvedené analýzy můžeme uvést algoritmické schéma pro řady:

1. $T = t_0$
2. $S = T$
3. while ($\neg B(S, T)$)
   {
   $T = f(T)$
   $S = S + T$
   }

Analogicky jako pro posloupnosti tak i pro řady platí tvrzení:

1. $t_i = f(t_{i-1})$ pro $i > 0$
2. $t_i \neq t_j$ pro $i \neq j$
3. $s_i = s_{i-1} + t_i$ pro $i > 0$
4. $\neg B(s_i, t_i)$ pro $i < n$
5. $B(s_n, t_n)$ změní se hodnota predikátu B

Při použití algoritmického schématu pro řady je třeba si uvědomit, že musí existovat *n*, pro které se změní hodnota predikátu $B(s_i, t_i)$. Podmínku ukončení založenou na aproximaci funkce je třeba volit uváženě s ohledem na funkci, argument a řadu, protože konvergence řady může být velmi pomalá. Může se nám dokonce stát, že požadovanou přesnost aproximace nedosáhneme, anebo její cena je velmi velká.

Příklad: Aproximace $y = e^x$

Exponenciální funkci e^x můžeme aproximovat řadou:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

Pro částečný součet s_i je možno napsat:

$$s_i = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^i}{i!}$$

Závislost sousedních členů řady je možné vyjádřit rekurentním vztahem:

$$t_j = t_{j-1}\frac{x}{j} \qquad \text{pro } j > 0$$

Řada konverguje pro všechna reálná čísla, a proto přírůstek jednotlivých členů řady do celkové sumy bude od určitého i klesat. Tuto skutečnost využijeme pro stanovení podmínky ukončení cyklu v algoritmickém schématu. Cyklus se bude opakovat, dokud hodnota přírůstku, tedy hodnota členu řady neklesne pod danou hranici, kterou označíme *eps*.

Algoritmus pro aproximaci e^x:

1. zadání x, eps
2. inicializace: t = 1, soucetRady = t, i = 0
3. opakuj pokud abs(t)≥ eps
   {
   inkrementace i
   výpočet dalšího členu t
   soucetRady = soucetRady + t
   }
4. zobrazení výsledku

Řada, pomocí které se aproximuje exponenciální funkce konverguje pro všechna reálná čísla, neznamená to však, že rychlost konvergence je stejná pro všechna reálná čísla. Rychlost konvergence je velká pro malé hodnoty argumentu *x*. Pro velké hodnoty *x* se doporučuje rozdělit argument *x* na celou část *c* a zlomkovou část *d* a použít vztah:

$$e^{c+d} = e^c \cdot e^d$$

Hodnota e^c se vypočítá opakovaným násobením *e*.

Příklad: Aproximace *y = sin(x)*
Pro částečný součet s_i je možno napsat:

$$s_i = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ....(-1)^i \frac{x^{2i+1}}{(2i+1)!}$$

Členy řady:

$$t_j = -t_{j-1} \frac{x^2}{k_j(k_j - 1)}, \qquad \text{pro j > 0}$$

$k_j = k_{j-1} + 2$
počáteční hodnoty: $t_0 = x$, $k_0 = 1$
pro $x \in \langle 0; \Pi/4 \rangle$ – konverguje nejrychleji (ověřte!)

Podmínka pro ukončení nespecifikuje absolutní hodnotu posledního členu, ale určuje relativně velikost posledního členu vzhledem k celkové sumě.

Algoritmus pro aproximaci y = sin(x):

1. zadání x, eps
2. inicializace t = x, soucetRady = t, k = 1
3. opakuj pokud abs(t) ≥ eps × abs(soucetRady)
   {
   k = k + 2
   výpočet dalšího členu t
   soucetRady = soucetRady + t
   }
6. zobrazení výsledku

Platí: *sin(2kΠ + x) = sin x*

9.5 Heuristika

Jak jsme viděli výše, množství řad pro aproximaci funkcí konverguje pouze na úzkém intervalu hodnot v rámci definičního oboru funkce. Aby byl výpočet v celém definičním oboru stejně přesný a zhruba stejně rychlý, musíme data (v tomto případě argument funkce) předzpracovat tak, aby vlastní výpočet probíhal v tom nejvýhodnějším intervalu. Obecně se všem opatřením, které vedou ke snížení náročnosti výpočtu a ke zvýšení efektivity říká ***heuristika***.

Příklad takového opatření jsme viděli například u algoritmu pro výpočet exponenciální funkce. Použitím vzorce $e^{c+d} = e^c \,.\, e^d$ zajistíme, že iterační výpočet podle algoritmického schématu bude probíhat pouze v úzkém intervalu hodnot, kde Taylorova řada konverguje nejrychleji a s nejmenší chybou. Pomocí řady totiž budeme počítat pouze desetinnou část exponentu. Mocninu čísla *e* na celou část exponentu jsme schopni efektivněji a přesněji spočítat například pomocí samostatného cyklu.

Podobně lze postupovat u periodických funkcí. Například funkce sinus je periodická s periodou 2Π. Tím, že od argumentu odečteme násobek Π se dostaneme do intervalu <0, 2Π), který je pro výpočet mnohem výhodnější. Pomocí středoškolské matematiky lze argument posunout do ještě výhodnějšího intervalu <0, Π/2), kde výpočet pomocí Taylovory řady konverguje nejrychleji. Podobné vztahy lze nalézt i u jiných funkcí, jejichž aproximaci lze počítat tímto způsobem.

9.5.1 Odstranění zbytečných výpočtů

Při pohledu na vzorce odvozené z Taylorovy řady zjistíme, že se zde počítají mocniny a faktoriály. Algoritmus, který by pro výpočet těchto řad používal funkce pro výpočet mocniny a faktoriálu, bychom však mohli označit jako naivní. Podívejme se například na vzorec pro výpočet exponenciální funkce:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

Při bližším pohledu zjistíme, že každý další prvek součtu se liší od předchozího prvku ve stejném poměru. Například část výpočtu 3! je obsažen ve výpočtu 2! Podobně to dopadne pro mocninu. Bylo by tedy zbytečné počítat pro každý prvek posloupnosti faktoriál nebo mocninu znovu. Pokud si odvodíme závislost mezi jednotlivými prvky řady, zjistíme, že výpočet faktoriálu ani mocniny nebude potřeba:

$$t_j = t_{j-1} \frac{x}{j} \qquad \text{pro j} > 0$$

Mocniny i faktoriály se nyní budou počítat implicitně pomocí iteračního algoritmu. Efektivita algoritmu vzroste minimálně o řád, protože výpočet mocniny a faktoriálu by znamenal další zanořené cykly v hlavním iteračním algoritmu. Celkový počet iterací by tedy rostl kvadraticky s každým dalším prvkem!

Platí pravidlo: Uvnitř cyklu nesmí být žádné zbytečné výpočty. Pokud to jde, je nutné každý takový výpočet eliminovat nebo přesunout mimo vlastní cyklus. Cyklus vnořený do jiného cyklu typicky vede k exponenciálnímu nárůstu počtu iterací.

9.6 Shrnutí

Po zvládnutí této kapitoly čtenář rozumí pojmům rekurentní problém, rekurentní vztah, algoritmické schéma a heuristika. Pro daný rekurentní problému dokáže odvodit správný rekurentní vztah a použít jej pro návrh iteračního algoritmu. Je rovněž schopen odhalit meze a problémy tohoto řešení a uplatnit na ně taková opatření, aby byl výpočet dostatečně přesný, obecný a efektivní.

9.7 Úlohy k procvičení

Napište program, který bude zpracovávat posloupnost racionálních čísel ze standardního vstupu. Upravte pro tento účel zadání č. 6 z kapitoly 6.5.

Vytvořte vlastní uživatelské funkce, které budou realizovat výpočet funkce sinus, kosinus, exponenciální funkci a logaritmus pomocí aproximace Taylorovou řadou se zadanou přesností (konkrétní odvozené vztahy najdete v matematických knihách, např. Bartsch: Matematické vzorce). Funkce musí být stejně použitelné, jako odpovídající funkce z matematické knihovny jazyka C.

Aplikujte tyto funkce na každou hodnotu ze vstupní posloupnosti. Výběr funkce a přesnost aproximace zadávejte jako parametry z příkazového řádku.

9.8 Kontrolní otázky

1. Jak je definován rekurentní vztah?
2. Vysvětlete postup řešení problémů, které jsou definované rekurentním vztahem.
3. Co je to algoritmické schéma?
4. Jakým způsobem se dosahuje zadané přesnosti výpočtu u problémů zadaných rekurentním vztahem?
5. Proč většinou pomocí iteračních výpočtů nedosahujeme absolutně přesných výsledků?
6. Může dojít k zacyklení iteračního výpočtu u problémů zadaných rekurentním vztahem? Pokud ano, uveďte možné příčiny.
7. Co to je heuristika a k čemu se používá?