Pracovní list 10: Složitost. Vztah programu a OS

Co už máme znát

- pojem složitost algoritmu, prostorová složitost, časová složitost;
- třídy složitosti;
- experimentální a analytický způsob stanovení složitosti;
- komunikace programu se standardními soubory OS;
- zpracování parametrů z příkazového řádku;
- zpracování hodnot z proměnných prostředí;
- nastavení návratového kódu;
- práce s diskovými soubory.

Kontrolní otázky

10.1 Vysvětlete pojem „časová složitost programu“.

10.2 Co je to třída složitosti?

10.3 Na čem závisí prostorová složitost algoritmu? Co naopak nemá na stanovení prostorové složitosti žádný vliv?

10.4 Jak se postupuje při experimentálním stanovení časové složitosti?

10.5 Jakými způsoby lze předávat data mezi programem a operačním systémem?

10.6 Jakého datového typu jsou parametry funkce `main` a co je v nich předáváno?

10.7 Jak lze zjistit jméno spuštěného programu?

10.8 Jakou funkcí lze přečíst hodnotu zadané proměnné prostředí?

10.9 Na jaké hodnoty se nastavuje návratový kód programu?

Příprava na cvičení

Ve cvičení budeme potřebovat překladač jazyka C++, editor pro přípravu zdrojových textů a vybavení příkazového řádku. Pro jednotlivé úlohy jsou k dispozici soubory s daty, případně výsledné soubory v adresáři `/home/rybicka/vyuka/progt/cecko/cviceni/cv10` na serveru `akela`. Konkrétní jména těchto souborů jsou uvedena u jednotlivých úloh.

Řešené příklady

Příklad 10.1 Implementujte algoritmus vyhledání hodnoty v binárním uspořádaném stromu. Experimentálně zjistěte jeho časovou složitost.

Řešení: Vytvoříme binární vyhledávací strom implementovaný pomocí dynamické struktury. Vložíme do něj řetězcová data ze standardního vstupního souboru a pak budeme v uložených datech vyhledávat výskyty řetězců přečtených z jiného souboru.

Pro sestavení potřebné *závislosti času vyhledání na množství dat* ve stromu budeme vždy plnit strom předem stanoveným počtem vstupních dat, na nichž pak provedeme 1000 pokusů o vyhledání řetězců z druhého souboru. Z naměřených časových hodnot vypočteme průměr a vypíšeme na standardní výstup potřebnou zprávu.

Měření času provedeme funkcí `clock()`, která dodává počet tzv. tiků (elementárních časových prvků) od nějakého pevného časového okamžiku. Rozdílem počtu tiků získaných na konci a na začátku činnosti algoritmu získáme určitý časový údaj (přepočet na vteřiny můžeme provést dělením této hodnoty konstantou `CLOCKS_PER_SEC`). Počet tiků lze uložit do proměnné typu `clock_t`. K použití uvedených rekvizit potřebujeme připojit knihovnu `ctime`.

Údaj o požadovaném *počtu* zpracovávaných vstupních dat budeme pro jednoduchost zadávat jako první parametr z příkazového řádku, jméno souboru s hledanými řetězci zadáme jako druhý parametr z příkazového řádku.

Pro účely analýzy získaných dat ještě vypíšeme *počet hladin* naplněného stromu, abychom mohli posoudit, do jaké míry se uplatňuje výhoda stromového uspořádání dat a jak to ovlivňuje naměřené hodnoty.

V následující implementaci jsou formou komentářů uvedeny další podrobnější informace.

```
509 #include <iostream>
510 #include <fstream>
511 #include <ctime>
512 #include <cstdlib>
513 using namespace std;
514
515 struct TypUzel {  //uzel stromu
516    string data;
517    TypUzel *Vlevo, *Vpravo;
518 };
519 typedef TypUzel * UkUzel;  //ukazatel reprezentuje strom
520
521 void Vloz(UkUzel &S, string d){  //vkládání do stromu
522   if (S==NULL){
523     S = new TypUzel;
524     S->data = d;
525     S->Vlevo=NULL;
526     S->Vpravo=NULL;
527   } else if (d<S->data) Vloz(S->Vlevo, d);
528      else Vloz(S->Vpravo, d);
```

```
529 }
530
531 bool Zjisti(UkUzel S, string co){ //vyhledání ve stromu
532   UkUzel pom=S;
533   while (pom!=NULL and pom->data!=co)
534     if (co<S->data) pom=pom->Vlevo;
535       else pom=pom->Vpravo;
536   return pom!=NULL;
537 }
538
539 void Napln(UkUzel &S, int pocdat){
540   string x;
541   for (int i=1; i<=pocdat; i++){
542     cin>>x;       //přečtení z datového souboru
543     Vloz(S, x);   //vložení do stromu
544   }
545 }
546
547 void Hledej(UkUzel S, string odkud){
548   string x; bool b;
549   ifstream Soub;
550   Soub.open (odkud.c_str());
551   for (int i=1; i<=120000; i++){
552     Soub >> x;      //přečtení hledané hodnoty
553     b=Zjisti(S, x); //vyhledání ve stromu
554   }
555   Soub.close();
556 }
557
558 void inorder(UkUzel S){ //použito pro výpis seřazených dat
559   if (S!=NULL) {
560     inorder(S->Vlevo);
561     cout << S->data<<endl;
562     inorder(S->Vpravo);
563   }
564 }
565
566 int Hladina=0, MaxHladina=0;
567 void Hladin(UkUzel S){  //zjištění počtu hladin
568   if (S!=NULL){
569     Hladina++;
570     if (Hladina>MaxHladina) MaxHladina = Hladina;
571     Hladin(S->Vlevo);
572     Hladin(S->Vpravo);
573     Hladina--;
```

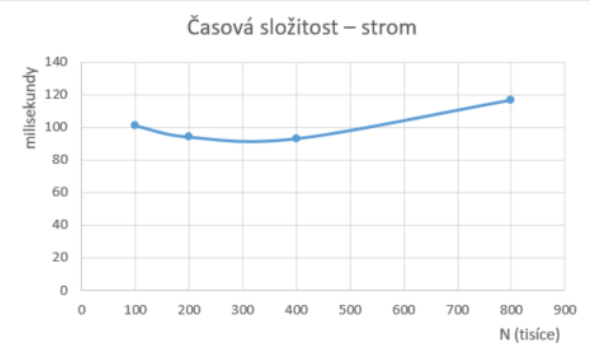
```
574   }
575 }
576
577 void VemPar(int &mnozstvi, string &zdroj, int p, char *par[]){
578   //zpracování parametrů z příkazového řádku
579   mnozstvi=1;
580   zdroj="";
581   if (p>1) mnozstvi=strtol(par[1], NULL, 10);
582      //strtol = konverze string to long; z knihovny cstdlib
583   if (p>2) zdroj.append(par[2]);
584 }
585
586 int main(int pocet, char *param[]){
587   UkUzel strom;
588   clock_t t;
589   int kolik;
590   string hled_retezce;
591   VemPar(kolik, hled_retezce, pocet, param);
592   strom=NULL;         //inicializace stromu
593   Napln(strom, kolik);  //naplnění stromu
594   t = clock();          //uchování aktuálního času
595   Hledej(strom, hled_retezce);
596   t = clock() - t;      //výpočet spotřebovaného času
597   Hladin(strom);        //zjištění počtu hladin
598   cout<<"Hladin stromu: "<<MaxHladina<<"; hodnot ve stromu: "
599       <<kolik<<"; čas trvání "<< t <<" tiků ("
600       <<(((float)t)/CLOCKS_PER_SEC)<<" sec.)."<<endl;
601 //  inorder(strom); -- využito k seřazení pracovního souboru
602   return 0;
603 }
```

S uvedenou implementací byly provedeny následující experimenty:

1. Strom byl naplněn uspořádanými daty (soubor `slovasort.txt`) – v tomto případě bylo dosaženo stavu, kdy stromová struktura degradovala na jedinou větev, v níž byly uloženy všechny vstupní hodnoty, tj. vyhledávání degradovalo na sekvenční hledání místo stromového.
   Do stromu bylo postupně uloženo 1 000, 2 000, 4 000 a 10 000 hodnot a pro každý tento počet bylo provedeno 1 000 vyhledání. Vyhledávaná slova jsou v souboru `hledej.txt`. Každé vyhledání bylo spuštěno 10×. Ze změřených časů byl vypočten aritmetický průměr.
2. Strom byl naplněn neuspořádanými daty (soubor `slova.txt`), v tomto případě však bylo použito mnohem většího objemu dat i mnohem většího počtu vyhledávání. Byly použity tyto objemy: 100 000, 200 000, 400 000 a 800 000 údajů ve stromu, pro každý počet bylo provedeno 120 000 hledání. Hledaná slova byla čtena opět ze souboru `hledej.txt`. To bylo spuštěno vždy opět 10× a ze změřených časů byl vypočten aritmetický průměr.

Výsledky byly ukládány do textového souboru `vysledky.txt`, který byl následně zpracován v tabulkovém procesoru a byly vyneseny odpovídající grafy – viz obrázek:

N	čas	N	čas
1000	635000	100000	101000
2000	1022000	200000	94000
4000	1592000	400000	93000
10000	3500000	800000	117000



U uspořádaných dat se s velkou přesností potvrdil očekávatelný lineární průběh časové složitosti. U neuspořádaných dat byly dosahované časy i přes velký počet dat i vyhledávání velmi krátké a byly ovlivněny náhodnými faktory. Časová složitost, která má být teoreticky logaritmická, se při experimentu poněkud zkreslila.

Příklad 10.2 Napište program, který přečte a na standardní výstup vypíše prvních 10 bajtů ze souboru, jehož jméno je zadáno jako parametr z příkazového řádku. Pokud tento soubor nelze číst, podívá se do proměnné prostředí s názvem `ZDROJ`. Pokud ani tam nenajde platné jméno souboru, vypíše chybové hlášení do standardního chybového souboru a nastaví návratový kód na 4.

Řešení: Soubor, z něhož chceme číst, je potřebné chápat jako binární – čteme jednotlivé bajty a celočíselné hodnoty těchto bajtů budou vypisovány na standardní výstup.

Hlavní částí programu je získání jména použitelného souboru. Je-li zadán parametr z příkazového řádku, má přednost před proměnnou prostředí. Budeme tedy pracovat v takovém pořadí, aby poslední zpracovávaný údaj potlačil svou validní hodnotou údaje získané v předchozích krocích.

První krok je zjištění obsahu proměnné prostředí s názvem `ZDROJ`. Bude-li existovat a bude-li mít nějakou hodnotu, ověříme, zda se tento řetězec vztahuje k čitelnému souboru – soubor se pokusíme otevřít a metodou `is_open()` ověříme funkčnost. Výsledkem prvního kroku je naplnění pracovní proměnné, která signalizuje použitelnost (nebo nepoužitelnost) otevřeného souboru.

Druhým krokem je zpracování prvního parametru z příkazového řádku. Pokud parametr existuje a obsahuje nějaké jméno souboru, opět se pokusíme soubor otevřít. V případě úspěchu bude tento soubor použit jako vstup, v opačném případě bude jako vstup použit soubor zpracovaný v prvním kroku. Pokud ani v prvním kroku nebyl získán použitelný soubor, program vypisuje chybové hlášení a nastavuje návratový kód na 4.

Další detaily jsou uvedeny formou komentářů ve zdrojovém textu.

```
604 #include <iostream>
605 #include <fstream>
606 #include <cstdlib>
607 #include <cstring>
608 using namespace std;
```

```
609
610 bool TestSoub(string Jmeno){ //test použitelnosti souboru
611   ifstream temp;
612   temp.open(Jmeno.c_str(), ios::binary); //pokus o otevření
613   if (temp.is_open()){
614      temp.close();
615      return true;       //soubor je použitelný
616   } else return false;
617 }
618
619 bool JeJmeno(string &pom, int pocet, char *param[]){
620   bool vysled=false;
621   if (pocet>1) {   //je zadán parametr z příkazového řádku
622     pom=””; pom.append(param[1]); //parametr přiřazen
623     if (TestSoub(pom)) vysled=true;
624   }
625   if (not vysled) {   //parametr selhal
626     char *XX = getenv(”ZDROJ”);  //zkusíme proměnnou prostředí
627     pom=””; if (strlen(XX)>0) pom.append(XX);
628     if (TestSoub(pom)) vysled=true;
629   }
630   return vysled;
631 }
632
633 int main(int pocpar, char *par[]){
634   string jmeno;    //jméno souboru
635   if (JeJmeno(jmeno, pocpar, par)) { //soubor úspěšně zadán
636      char X;
637      ifstream data (jmeno.c_str(), ios::binary);
638      for (int i=1; i<=10; i++){  //přečtení 10 bajtů
639        data.get(X);
640        cout << int(X) << ” ”;
641      }
642      cout << endl;
643      return 0;
644   } else {cerr << ”Nenalezen vstupní soubor.”<<endl;
645      return 4;
646   }
647 }
```

Příklady

Příklad 10.3 Implementujte lineární dynamický jednosměrný seznam fungující jako zásobník. Experimentálně určete časovou složitost vyhledání zadané hodnoty.

Příklad 10.4 Implementujte algoritmus násobení čtvercových matic řádu *N*. Ze zápisu algoritmu určete jeho časovou složitost v závislosti na řádu matic.

Příklad 10.5 Napište program, který zpracovává tři parametry z příkazového řádku. Prvním parametrem je jedno reálné číslo, druhým parametrem druhé reálné číslo a třetím parametrem požadovaná aritmetická operace se zadanými čísly. Operace je zadána formou řetězce: „plus“, „minus“, „krat“, „deleno“. Pokud je všechno zadáno správně a operaci lze provést, objeví se výsledek na standardním výstupu. V opačném případě program vypíše odpovídající chybové hlášení do standardního chybového výstupu a nastaví návratový kód na nenulovou hodnotu.

Příklad 10.6 Napište program, který vypíše na standardní výstup prvních 10 bajtů souboru, jehož jméno může být zadáno jako první parametr z příkazového řádku. Pokud parametr není zadán nebo je zadán chybně, program se podívá do proměnné prostředí s názvem INPFILE. Pokud ani tam nebude platný název zadán, pokusí se program přečíst konfigurační soubor, jehož jméno je stejné jako jméno tohoto programu a rozšíření je `.rc`. Je potřebné otestovat, zda konfigurační soubor existuje. Pokud nikde nebude zadáno jméno použitelného souboru, program vypíše chybové hlášení do standardního chybového souboru.

Příklad 10.7 Napište program, který zkopíruje obsah zadaného souboru do nového souboru. Zadaný soubor je specifikován prvním parametrem z příkazového řádku, cílový soubor pak druhým parametrem. Pro kopii použijte operace nad binárními soubory. V případě chybného zadání parametrů vypište chybové hlášení do standardního chybového souboru.

Příklad 10.8 Implementujte rekurzivní variantu výpočtu *N*-tého členu Fibonacciho posloupnosti a experimentálně stanovte její časovou složitost v závislosti na *N*. Požadované číslo *N* čtěte jako první parametr z příkazového řádku, výsledky spotřebovaného času vypisujte na standardní výstup tak, abyste je následně mohli zpracovat dalším programem, který vypočítá potřebné průměry a vytvoří podklad pro zpracování a vykreslení grafu v tabulkovém procesoru. Tento pomocný program zpracujte a použijte jej pro zpracování výsledných dat.

Příklad 10.9 Využijte postupu a implementované podpory z příkladu 10.8 a stanovte experimentálně časovou složitost nerekurzivního výpočtu *N*-tého členu Fibonacciho posloupnosti.

Příklad 10.10 Napište program, který zašifruje soubor, jehož jméno je zadáno jako první parametr z příkazového řádku, metodou XOR s heslem zadávaným z klávesnice. Výstup uloží do souboru, jehož jméno je stejné jako jméno vstupního souboru, ale přidá k němu další rozšíření `.cif`. Není-li jméno vstupního souboru použitelné, nevytvoří se výstupní soubor, ale program vypíše odpovídající chybové hlášení do standardního chybového souboru.

Příklad 10.11 Určete analyticky prostorovou složitost rekurzivní implementace výpočtu N členů Fibonacciho posloupnosti.

Co máme po cvičení umět

- Experimentální určení časové složitosti.
- Analytické určení časové a prostorové složitosti.
- Práci s parametry zadávanými z příkazového řádku.
- Práci s proměnnými prostředí.
- Práci s návratovým kódem programu.

Kontrolní otázky

10.10 Jak se vypočítá při násobení matic hodnota prvku $a_{i,j}$ výsledné matice?

10.11 Jakou časovou složitost má algoritmus pro výpočet N-tého členu Fibonacciho posloupnosti v rekurzivní variantě?

10.12 Jakou časovou složitost má sekvenční hledání v lineárním seznamu?

10.13 Jak lze z řetězcového parametru získat číselnou hodnotu?

10.14 Jak se zjistí, že proměnná prostředí neexistuje?

10.15 Jak lze otestovat, že soubor je použitelný pro čtení?

10.16 V jakém tvaru je vhodné poskytnout data o experimentálním zjištění časové složitosti pro zpracování v tabulkovém procesoru?

10.17 Jak se provede binární kopie souboru?