Anotace

- Direktivy překladače
- Soubory (textové)
- Struktury (record)
- Základní třídicí algoritmy
- Quicksort

Direktivy překladače

- Překladač kontroluje plno věcí, například:
- zda nekoukáme za konec pole,
- zda nám nepřetekl zásobník,
- anebo zda nenastala chyba na vstupu/výstupu...
- Většinou je užitečné mít kontroly zapnuté, někdy však "víme, co děláme".
- V tom případě můžeme na nezbytnou dobu chování překladače změnit pomocí tzv. *direktiv překladače.*
- Direktivy vypadají jako komentář, tedy jsou ve složených závorkách, ovšem začínají znakem string ($), jméno je zpravidla 1znakové a následuje přepínač $+/-$.

Direktivy překladače:

- Příklad: $\{\$R-\}$ – vypni *range-checking*.
- Nejdůležitější:
  - $\$Q$ – overflow-checking,
  - $\$R$ – range-checking,
  - $\$I$ – test vstupu a výstupu,
  - Úplný seznam najdete v helpu (některé direktivy se liší podle překladačů).

Soubory a práce s nimi

- V tomto semestru budou pouze soubory textové. Ačkoliv existují i soubory binární, ty budou předmětem výuky až v létě!
- Textový soubor ovládáme pomocí proměnné typu `Text`.
- Příslušnou proměnnou "napojíme" na daný soubor pomocí funkce `Assign`,
- otevřeme pomocí funkce `Reset`, `Rewrite` nebo `Append`,
- soubor čteme pomocí funkce `Read` (resp. `Readln`), které jako první argument předáme příslušnou proměnnou typu `Text`, zapisujeme analogicky pomocí funkcí `Write` a `Writeln`.
- Nakonec soubor uzavřeme pomocí funkce `Close`.

Práce se souborem – syntax (1)

- `var f:Text;`
- `Assign(f,'soubor.txt');` – asociuj proměnnou `f` se souborem `soubor.txt`.
- `Reset(f);` – otevři soubor `f` (pro čtení).
- `Rewrite(f);` – otevři soubor `f` a jeho dosavadní obsah znič.
- `Append(f);` – otevři soubor `f` pro zápis za jeho dosavadní konec.

Práce se souborem – syntax (2)

- `Writeln(f,'Zapiseme text do souboru');` – zapiš do souboru příslušný text.
- `Read(f,a);` – načti ze souboru proměnnou `a`.
- `Close(f);` – uzavři soubor (už s ním nebudeme pracovat).
- `eof(f);` – funkce, která sdělí, zda jsme na konci souboru.
- `eof;` – funkce oznamující konec standardního vstupu (z klávesnice).
- Existuje mnoho dalších funkcí jako `Rename`, `Erase`, ...

Potíže se soubory

- Často se stane, že otevíraný soubor neexistuje.
- Tato událost vyvolá input/output error.
- Nechceme-li při zapnuté této direktivě překladače soubor zničit (pomocí `Rewrite`, které sice neexistující soubor založí, ale existující přemaže), použijeme direktivu překladače a zda nastala chyba zjistíme pomocí funkce `IOResult`.

Příklad

```
Assign(f,'soubor.txt');
{$I-} {Vypni test na vstupně-výstupní chyby}
Reset(f);
{$I+} {Zapni test vstupně-výstupních chyb}
if IOResult<>0 then
begin writeln('Chyba!'); halt;
end;
while not eof(f) do begin
      readln(f,s);
      writeln(s);
end;
```

Pozor, `IOResult` je funkce a po zavolání ztratí hodnotu, nelze ji tedy číst opakovaně a její výsledek je případně třeba uložit do proměnné!

Využití souborů

- Pokud máme vstupy několika typů (například popis kódu a text),
- konfigurace (například zápočtového programu),
- ladění: Vyrobíme několik vzorových vstupů a zkusíme, co program udělá.
- Ladit můžeme kupříkladu hned třídící programy (algoritmy).

Datový typ `record`

- Typicky míváme objekt popsaný několika hodnotami (bod v rovině: dvojice hodnot).
- Je nešikovné mít každou hodnotu někde úplně jinde (je vhodnější mít data co nejvíce pohromadě).
- Definujeme tedy strukturu, kde budeme mít všechny údaje pohromadě.
- Definujeme pomocí klíčového slova `record`.

Typ record – příklad

```
type  bod=record
            x,y:integer;
      end;
var   a,b:bod;
      kn:record
            autor: string[100];
            nazev: string[100];
            rok: integer;
      end;
```

Přístup do struktury

Do struktur přistupujeme pomocí operátoru tečky:
`a.x` – prvek `x` struktury `a`
Jednotlivé prvky se chovají jako běžné proměnné, můžeme tedy číst jejich hodnoty, zapisovat do nich...

Typ record – příklad použití

```
var a,b:bod; kn:kniha;
begin
      a.x:=1;
      a.y:=2;
      b.x:=10;
      b.y:=10;
      kn.autor:='Topfer, P.';
      kn.nazev:='Algoritmy a programovaci techniky';
      kn.rok:=1995;
end.
```

Pole struktur

```
var knihovna:array [1..100] of record
            autor:string[25];
            nazev:string[45];
            rok:integer;
end;
begin
      for i:=1 to 100 do begin
            readln(knihovna[i].autor);
            readln(knihovna[i].nazev);
            readln(knihovna[i].rok);
      end; ...
```

Klíčové slovo `with`

Chceme-li pracovat se strukturami, je otravné stále říkat, ve které struktuře se pohybujeme (`struktura_s_dlouhym_nazvem_kterou_jsme_vymysleli_v_opilosti`). Proto můžeme říct:

`with` struktura `do` příkaz nebo blok;

a v sekci `with` můžeme přistupovat k jednotlivým prvkům struktury rovnou.

Konstrukce `with` – příklad:

```
procedure vypis(kniha_s_dlouhym_nazvem:kniha);
begin
      with kniha_s_dlouhym_nazvem do
            writeln(autor:25,nazev:46,rok:5);
end;
...
      for i:=1 to 100 do
            vypis(knihovna[i]);
...
```

Problém třídění – motivace

- Máme načtena data (například čísla),
- chceme je zpracovat například v rostoucím pořadí.
- Jak to udělat? Setřídíme, zpracujeme.
- Předpokládejme, že data jsou v poli.

Problém třídění – jednoduché třídící algoritmy

- Bublinkové třídění (BubbleSort),
- zatřiďování alias třídění přímým vkládáním (InsertSort),
- třídění vyběrem (SelectSort),
- QuickSort.

Bublinkové třídění

- Geometrická interpretace:
  Bublinky v kapalině jdou zpravidla vzhůru
- Myšlenka: Porovnáváme po sobě jdoucí čísla (ve smyslu sousední v zadaném poli) od prvního k poslednímu, jsou-li v nesprávném pořadí, prohodíme je.
- Prvky "probublávají" "správným" směrem.
- Opakujeme bublání, dokud se prohazuje.

Bubblesort v pseudokódu

```
prohazovalose:=true;
while prohazovalose:=true do
begin
    for i:=1 to pocet - 1 do
    begin
        prohazovalose:=false;
        if pole[i]>pole[i+1] then
        begin prohod(pole[i],pole[i+1]);
              prohazovalose:=true;
        end;
    end;
end;
```

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.
- Složitost `BubbleSortu` je tedy $O(n^2)$.

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.
- Složitost `BubbleSortu` je tedy $O(n^2)$.
- Implementaci, kdy se střídavě bublá z jedné strany na druhou a z druhé na první se říká `ShakeSort` a funguje pro něj stejný odhad složitosti.

Třídění přímým výběrem a zatřiďováním

Přímý výběr:

- Opakuj, dokud není tříděné pole prázdné:
- Najdi v poli minimum a přesuň ho na konec setříděného pole.

Zatřiďování:

- Opakuj, dokud není tříděné pole prázdné:
- Vyjmi z něj první prvek a zatřiď do cílového pole, tedy: najdi pozici, kam prvek patří, přidej ho tam a zbytek setříděného pole posuň (o jedna dál).

Analýza složitosti: $n$krát opakujeme proces, který trvá nejvýše n kroků, tedy také $O(n^2)$.

Quicksort – třídění za pomoci rekurze – idea:

- Pokud třídíme jedno číslo, nic nedělej (posloupnost je setříděna),
  tedy vrať posloupnost tak, jak jsme ji dostali.
- V poli POLE vyber jeden prvek (dále pivot).
- Rozděl POLE na pole A obsahující prvky menší než pivot
- a na pole B obsahující prvky větší nebo rovné pivotu.
- Pomocí sebe sama setřiď pole A,
- pomocí sebe sama setřiď pole B,
- Vypiš: pole A, pivot, pole B.

Quicksort – třídění za pomoci rekurze – idea:

- Pokud třídíme jedno číslo, nic nedělej (posloupnost je setříděna),
  tedy vrať posloupnost tak, jak jsme ji dostali.
- V poli POLE vyber jeden prvek (dále pivot).
- Rozděl POLE na pole A obsahující prvky menší než pivot
- a na pole B obsahující prvky větší nebo rovné pivotu.
- Pomocí sebe sama setřiď pole A,
- pomocí sebe sama setřiď pole B,
- Vypiš: pole A, pivot, pole B.

Quicksort

Implementace bude na webu.

Quicksort – analýza složitosti

- V nejhorším případě: $\Theta(n^2)$.
- V nejlepším případě: $\Theta(n \log n)$.
- V průměrném případě: $\Theta(n \log n)$.

Quicksort – varianty a složitosti

- Randomizovaný quicksort: Pivotem volíme náhodný prvek,

Quicksort – varianty a složitosti

- Randomizovaný quicksort: Pivotem volíme náhodný prvek,
- složitost v průměrném případě $\Theta(n \log n)$.

Quicksort – varianty a složitosti

- Randomizovaný quicksort: Pivotem volíme náhodný prvek,
- složitost v průměrném případě $\Theta(n \log n)$.
- Analýza se opírá o netriviální (i když známá) tvrzení teorie pravděpodobnosti.

Quicksort – přirozené otázky?

- Na čem závisí složitost?

Quicksort – přirozené otázky?

- Na čem závisí složitost?
- Jak volbu pivota ovlivnit?

Quicksort – přirozené otázky?

- Na čem závisí složitost?
- Jak volbu pivota ovlivnit?
- Lze volit pivot tak, aby quicksort provedl vždy jen $O(\log n)$ "fází"?

Quicksort – přirozené otázky?

- Na čem závisí složitost?
- Jak volbu pivota ovlivnit?
- Lze volit pivot tak, aby quicksort provedl vždy jen $O(\log n)$ "fází"?
- Mediánem nazveme takový prvek, že alespoň polovina prvků je alespoň taková jako on a alespoň polovina nejvýše taková.

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,
- vyřeší každou zvlášť a z jejich řešení zjistí řešení původní instance.

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,
- vyřeší každou zvlášť a z jejich řešení zjistí řešení původní instance.
- Příklady: Řízení podniku (ředitel leteckých závodů zpravidla neřeší, jak přinýtovat zadní část křídla),

Quicksort – ponaučení – metoda rozděl a panuj

- Rozdělenému nepříteli se lépe vládne.
- Rozděl a panuj rozdělí instanci na přesně definované menší instance,
- vyřeší každou zvlášť a z jejich řešení zjistí řešení původní instance.
- Příklady: Řízení podniku (ředitel leteckých závodů zpravidla neřeší, jak přinýtovat zadní část křídla),
- Příklady (algoritmické): Quicksort, binární vyhledávání.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)
- Co když v Quicksortu jako pivot vybereme medián?

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)
- Co když v Quicksortu jako pivot vybereme medián?
- V tom případě máme rekurenci: $T(n) = 2T(\frac{n}{2}) + kn$.

Metody analýzy složitosti problémů typu rozděl a panuj

- Složitost zpravidla získáme ve tvaru rekurentní formule $T(n) = T(n_1) + T(n_2) + ... + T(n_k) + f(n)$.
- My chceme rekurenci rozbít. Jedna z metod je indukce.
- Příklad: $T(n) = T(\frac{n}{2}) + k$ (binární vyhledání)
- Co když v Quicksortu jako pivot vybereme medián?
- V tom případě máme rekurenci: $T(n) = 2T(\frac{n}{2}) + kn$.
- Nelze použít jinou metodu?

Master theorem (symetrická verze)

Theorem

Je-li $T(1) = c$ *a* $T(n) = a \cdot T(\frac{n}{b}) + \Theta(n^d)$, *kde* $a \geq 1$, $b > 1$, $d \geq 0$ *a* a, b *přirozená čísla, potom platí:*

- $T(n) \in \Theta(n^d)$, *pokud* $a < b^d$,
- $T(n) \in \Theta(n^d \log n)$, *pokud* $a = b^d$
- *a* $T(n) \in \Theta(n^{\log_b a})$, *pokud* $a > b^d$.

Master theorem – myšlenka důkazu:

- Výpočet si představíme "po hladinách" podle hloubky rekurze.
- Zjistíme složitost každé hladiny zvlášť,
- zjistíme, která bude trvat nejdéle,
- posčítáme.
- Hladin bude $\log_b n$, protože čekáme, až z n zbyde v argumentu jen konstanta.
- Jaká je složitost hladiny k?: $a^k \cdot (\frac{n}{b^k})^d$.

- Která hladina bude trvat nejdéle?
- První (pokud $\left(\frac{a}{b^d}\right)^k < 1$), což je první případ.
- Nebo poslední (pokud $\left(\frac{a}{b^d}\right)^k > 1$), což je třetí případ.
- Nebo všechny stejně (pokud $\left(\frac{a}{b^d}\right)^k = 1$), což je druhý případ.
- Složitost je tedy součtem geometrické posloupnosti!
- Kvocient této řady je menší než jedna, větší než jedna, nebo právě jedna.
- Je-li kvocient různý od jedné, odhadneme součet největším z členů, zbytek je konstanta.
- Je-li kvocient jedna, součet je: hloubka krát složitost libovolné hladiny. A hloubka je $\log n$.

Analýza master-theorememem:

- Binární vyhledání: $T(n) = T(\frac{n}{2}) + \Theta(n^0)$
- druhý případ $a = 1, b = 2, d = 0, a = b^d$, tedy složitost binárního vyhledání je $\Theta(\log n)$.
- Quicksort obecně: $T(n) = T(n_1) + T(n - n_1) + \Theta(n^1)$. Master theorem mlčí!
- Quicksort vybereme-li za pivot medián a umíme-li medián vybrat s lineární složitostí: $T(n) = 2T(\frac{n}{2}) + \Theta(n^1)$. Jde opět o druhý případ $a = 2, b = 2, d = 1, a = b^d$, získáváme opět složitost $\Theta(n \log n)$.

Další příklad – násobení matic:

- Naivní algoritmus: $T(n) = \Theta(n^3)$.
- Strassenův algoritmus použije jen 7 násobení matic o polovinu menších,
- režie každé iterace je kvadratická (vůči šířce matice),
- rekurence tedy vychází: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$.
- Jde o 3. případ ($a = 7, b = 2, d = 2, a > b^d$) a tedy složitost Strassenova algoritmu je: $\Theta(n^{\log_2 7})$.
- Dokazujte toto indukcí...