Anotace

- STŘEDNÍK,
- medián v lineárním čase,
- několik odstrašujících příkladů,
- předávání parametrů programu,
- problémy řešitelné "vyplněním tabulky".

STŘEDNÍK

- je soutěž pro studenty prvních ročníků, která
  - proběhne v pátek 11. prosince (tedy za týden) od 15:00 do 18:00,
  - spočívá v řešení úloh v CodExu,
  - probíhá v kategoriích *matematika* a *informatika*,
  - má za cíl ukázat především vám, jak na tom jste,
  - je pořádána kolegy Töpferem a Holanem $\Rightarrow$ názor nezávislý na vašich vyučujících,
  - může být řešena právě odtud, odkud lze řešit domácí úkoly,
  - má pro případ havárie CodExu stanoven náhradní způsob submitu.

viz `ksvi.mff.cuni.cz/~holan` (sekce aktuality).

Medián v lineárním čase

- Předminule byl algoritmus Quicksort.
- Problémem bylo, jak volit pivot.
- Volíme-li špatně, děláme mnoho iterací rekurze.
- Hledáme-li pivot dlouho, nepříjemně to trvá.
- Jak hledat medián v lineárním čase?
- Ve skutečnosti nebudeme hledat medián, ale k-tý nejmenší prvek.

Medián v lineárním čase

- Rozděl vstup na pětice,
- v každé pětici najdi medián,
- najdi medián mediánů (tedy medián mezi mediány pětic),
- rozděl vstup na menší a větší,
- zjisti, zda se k-tý nejmenší nachází mezi většími nebo menšími
- pokračuj s hledáním příslušné hromádce.

Medián lineárně detaily

- Jak najít mediány pětic?

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).
- Jak "pokračovat na příslušné hromádce?"

Medián lineárně detaily

- Jak najít mediány pětic?
- Hrubou silou (v konstantním čase, opakujeme lineárně-krát).
- Jak najít medián mediánů?
- Rekurzívně (zavolej se na pole mediánů pětic).
- Jak "pokračovat na příslušné hromádce?"
- Rekurzívně (zavolej se buďto na menší hromádku a hledej k-tý nejmenší, nebo máme-li hledat v hromádce větších hodnot budiž l počet prvků na menší hromádce a hledej v hromádce větších $k - l$-tý nejmenší.

Proč je algoritmus lineární?

Protože:

- mediánů pětic je přibližně pětina délky vstupu,
- hromádka menších čísel stejně jako hromádka větších čísel bude mít velikost aspoň $3/10$,
- tudíž každá z hromádek bude mít též velikost nejvýš $7/10$.
- Zbytek je jen indukce.

Indukce:

Složitost algoritmu vede k rekurenci:

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7}{10}n\right) + kn.$$

Ukážeme, že existuje l takové, že $T(n) \leq ln$:

$$T(n) \leq \frac{ln}{5} + \frac{7ln}{10} + kn = kn + \frac{9}{10}ln$$

a tedy stačí volit $l \geq 10k$.

Odstrašující příklady `for`-cyklus

- Co se stane, když změníme obsah cyklící proměnné ve `for`-cyklu?
- ```
  for i:=1 to 10 do
  begin
        writeln(i);
        if (i= 3) then
              i:=1000;
  end;
  ```
- Co řekne FPC? A co BPC? A co GPC? A co jeho option `--borland-pascal`?
  ```

Předávání parametrů referencí

Co je zrádného v následujícím zdrojáku?

```
procedure mixer(var a,b,c:integer);
begin
      a:=b+c;
      a:=b+c;
      a:=b+c;
end; ... a:=1; b:=1; c:=1; mixer(a,b,c);
writeln(c);
```

- Na první pohled v pořádku.

Předávání parametrů referencí

Co je zrádného v následujícím zdrojáku?

```
procedure mixer(var a,b,c:integer);
begin
      a:=b+c;
      a:=b+c;
      a:=b+c;
end; ... a:=1; b:=1; c:=1; mixer(a,b,c);
writeln(c);
```

- Na první pohled v pořádku.
- A co když zavoláme `mixer(c,c,c);`

Parametry programu

- copy c:\io.sys c:\windows\kram.krm
- copy con program.exe
- dir *.pas
- jedná se o volání programů s parametry.
- Parametry jsou elegantní způsob načtení dat.

Parametry – myšlenky a syntax

- Ve většině jazyků se ovládá takto:
- umíme zjistit počet argumentů
  (pomocí proměnné nebo funkce),
- umíme zjistit obsah *i*-tého argumentu,
- jednotlivé argumenty jsou zpřístupněny jako řetězec (string).

Parametry – implementace v Pascalu

- Počet zjistíme pomocí funkce `ParamCount`,
- obsahy zjistí funkce `ParamStr`.
- `ParamCount` nebere žádné parametry,
- `ParamStr` bere jeden celočíselný parametr.
- Parametry jsou číslovány od 0 do `ParamCount`.

Parametry specifika

- `ParamStr(0)` vrátí jméno programu.
- Použití: Napíšeme jeden program, který něco dělá a vrací a podle jména rozhodneme, co máme udělat.
- Užitečné v kontextu linků (UNIX), na Windows asi moc ne.
- V IDE Borland Pascalu nastavíme v menu IDE "Run Parameters".

Parametry – příklad

```
program x;
var i:integer;
begin
      writeln('Pocet parametru je: ',paramcount);
      for i:=0 to paramcount do
            writeln(i,'. parametr je: ',paramstr(i));
end.
```

- Přednášející jde do F1,
- nejdelší rostoucí podposloupnost,
- počet korektních uzávorkování pomocí n párů závorek,
- nalezení všech rozkladů zadaného čísla na sčítance,
- Pascalův trojúhelník,
- pořadí násobení matic.

Přednášející jde dnes do F1 potřetí:

- Rekurence $T(n) = T(n-1) + T(n-2)$,

Přednášející jde dnes do F1 potřetí:

- Rekurence $T(n) = T(n-1) + T(n-2)$,
- vedla na program:

```
function schody(a:integer):longint;
begin
if a=1 then schody:=1;
else  if a=2 then schody:=2;
                 else  schody:=schody(a-1)+schody(a-2);
```

Přednášející jde dnes do F1 potřetí:

- Rekurence $T(n) = T(n-1) + T(n-2)$,
- vedla na program:

```
function schody(a:integer):longint;
begin
if a=1 then schody:=1;
else  if a=2 then schody:=2;
                 else  schody:=schody(a-1)+schody(a-2);
```

- Proto jsme si pořídili pole, kde už byly staré výsledky.

Přednášející jde dnes do F1 potřetí:

- Rekurence $T(n) = T(n-1) + T(n-2)$,
- vedla na program:

```
function schody(a:integer):longint;
begin
if a=1 then schody:=1;
else  if a=2 then schody:=2;
                 else  schody:=schody(a-1)+schody(a-2);
```

- Proto jsme si pořídili pole, kde už byly staré výsledky.
- Pole ani nebylo potřeba, pokud jsme začali "odpředu" a pamatovali si poslední dvě hodnoty.

Varianta s "cache"

```
program nic;
const MAX=150;
var cache:array[1..MAX] of longint;
function schody(a:integer):longint;
begin
      if cache[a]<>0 then schody:=cache[a]
      else
      begin
            if a= 1 then cache[a]:=1
            else if a=2 then cache[a]:=2
            else cache[a]:=schody(a-2)+schody(a-1);
            schody:=cache[a];
      end;
end;
```

Nejdelší rostoucí podposloupnost

- Utvoř posloupnost dvojic (třeba pomocí pole recordů),
- první prvek obsahuje příslušnou hodnotu, druhý ukazuje délku nejdelší rostoucí podposloupnosti končící dotyčným prvkem.
- Vyplňuj zleva doprava, pro každý prvek najdi mezi prvky jemu předcházejícími takový menší prvek, ve kterém končí nejdelší dostupná podposloupnost.
- Podposloupnost najdi od konce:
- Najdi prvek nabízející největší možnou délku,
- poznamenej si HODNOTU a DÉLKU.
- postupuj od konce a pokud najedeš na prvek nabízející délku DÉLKA s hodnotou nejvýše tolik, kolik HODNOTA, prvek si poznamenej, sniž DÉLKU o jedna a HODNOTU na hodnotu nalezeného prvku.
- Nalezenou posloupnost otoč.

Nejdelší rostoucí podposloupnost – výpočet (vyplnění tabulky)

```
for i:=1 to n do begin
     maximum:=0; maxindex:=0;
     for j:=i-1 downto 1 do begin
          if pole[j].hodnota<pole[i].hodnota
                         and pole[j].delky>maximum then
          begin
               maximum:=pole[j].delky;
               maxindex:=j;
          end;
     pole[i].delky:=maximum+1;
end;
```

Počet korektních uzávorkování

- Jak budeme řešit?
- Pomocí rekurze podle rostoucího počtu přidaných závorek.
- Uděláme funkci, která:
- zkusí přidat otevírací závorku (rekurze),
- zkusí přidat zavírací závorku (rekurze),
- pokud jsou použity všechny závorky, zvyš počet uzávorkování o 1.

Počet korektních uzávorkování

```
var paru,celkem:longint;
procedure pridej_zavorku(levych,pravych:integer);
begin
     if levych>pravych then
pridej_zavorku(levych,pravych+1);
     if paru>levych then
pridej_zavorku(levych+1,pravych);
     if (levych=pravych) and (paru=levych) then
inc(celkem);
end;
```

Počet korektních uzávorkování

```
var paru,celkem:longint;
procedure pridej_zavorku(levych,pravych:integer);
begin
     if levych>pravych then
pridej_zavorku(levych,pravych+1);
     if paru>levych then
pridej_zavorku(levych+1,pravych);
     if (levych=pravych) and (paru=levych) then
inc(celkem);
end;
```

Jaký problém má toto řešení?

Počet korektních uzávorkování

```
var paru,celkem:longint;
procedure pridej_zavorku(levych,pravych:integer);
begin
     if levych>pravych then
pridej_zavorku(levych,pravych+1);
     if paru>levych then
pridej_zavorku(levych+1,pravych);
     if (levych=pravych) and (paru=levych) then
inc(celkem);
end;
```

Jaký problém má toto řešení?

Počítáme pořád to samé!

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,
- je-li `cache[levych,pravych]` $=0$, spustíme výpočet (výsledek ještě neznáme) a na konci funkce si ho uložíme do pole.

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,
- je-li `cache[levych,pravych]` =0, spustíme výpočet (výsledek ještě neznáme) a na konci funkce si ho uložíme do pole.
- Je-li `cache[levych,pravych]`<>0, připočteme tolik platných uzávorkování.

Závorkování – cache

- Jak z pasti? Položíme si správnou otázku:
- Záleží výsledek funkce `pridej_zavorku` na něčem jiném, než na parametrech `levych` a `pravych`?
- Nezáleží. Proč si potom nezapamatujeme, kolik závorkování přidá tato funkce pro konkrétní hodnoty `levych` a `pravych`:
- `cache:array[1..MAX,1..MAX] of longint`.
- Pole na počátku inicializujeme nulami,
- je-li `cache[levych,pravych]` =0, spustíme výpočet (výsledek ještě neznáme) a na konci funkce si ho uložíme do pole.
- Je-li `cache[levych,pravych]`<>0, připočteme tolik platných uzávorkování.
- Rozdíl mezi variantou "rekurze" a "cachovaná rekurze" je rozdíl mezi nepoužitelným a dobrým algoritmem!

Nalezení všech rozkladů zadaného čísla na součet

- Příklad: $2 = 2$ nebo $2 = 1 + 1$; $3 = 3$, $3 = 2 + 1$ anebo $3 = 1 + 1 + 1$.
- Sčítance chceme vždy v nerostoucím pořadí. Nápady jak řešit?
- Jako obvykle rekurzí. Budeme si pamatovat, kolik ještě zbývá rozdělit a kolik je maximum. A zkusíme všechno od maxima, až k jedné.
- Nezajímají nás samotné rozklady, ale jen jejich počet!

Rekurzívní funkce:

```
procedure rozloz(kolik,maximum:integer);
var i:integer;
begin
     if kolik:=0 then inc(pocet)
     else
          for i:=maximum downto 1 do
               rozloz(kolik-i,i);
end;
```

Jaký je problém?
Pořád ten samý
(počítáme pořád to samé).

Rozklad na sčítance

- Jak z pasti tentokrát?

Rozklad na sčítance

- Jak z pasti tentokrát?
- Stejně jako u závorkování:

Rozklad na sčítance

- Jak z pasti tentokrát?
- Stejně jako u závorkování:
- Uděláme dvourozměrné pole cache a budeme si do něj značit, kolika způsoby lze rozložit KOLIK, je-li první sčítanec nejvýše MAXIMUM:

```
procedure rozloz(kolik,maximum:integer);
var i,nazacatku:integer;
     if cache[kolik,maximum]<>0 then
           rozloz:=cache[kolik,maximum];
     else
     begin  nazacatku:=pocet;
          if kolik= 0 then inc(pocet);
          else  for i:=maximum downto 1 do
               rozloz(kolik-i,i);
          cache[kolik,maximum]:=pocet-nazacatku;
     end;
end;
```

Konec

Děkuji za pozornost...