Anotace

- Dámy na šachovnici – dominance a nezávislost.
- Aritmetické výrazy, notace a převody mezi nimi,
- Problémy řešitelné "vyplněním tabulky":
  - Přednášející jde do M1,
  - nejdelší rostoucí podposloupnost.

Dámy na šachovnici

- Jak rozmístit n dam na šachovnici $n \times n$, aby se vzájemně neohrožovaly?
- Jak rozmístit co největší počet dam, aby se navzájem neohrožovaly? (nezávislost)
- Jak rozmístit co nejmenší počet dam, aby ohrožovaly celou šachovnici? (dominance)

n dam na $n \times n$

- Každá dáma přijde do jednoho řádku,
- stačí dámu postupně zkoušet umísťovat na dosud neohrožená pole a spustit rekurzi na další řádky:

```
function dama(radek:integer);
begin  if radek>pocet_radku then vypis
       else  for i:=1 to pocet_radku do
                    if volny_sloupec[i] and
volna_diag1??? and volna_diag2??? then
                    begin
                          obsad_to_vsechno;
                          dama(radek+1);
                          uvolni_to_vsechno;
                    end;
end;
```

Dámy na šachovnici

- Volné řádky můžeme evidovat v poli, abychom nemuseli prohledávat šachovnici.
- Volné diagonály taky, protože
- pro jednu diagonálu je součet konstantní $(1 + 5 = 2 + 4 = 3 + 3 = 4 + 2 = 5 + 1)$,
- pro druhou diagonálu je konstantní rozdíl $(1 - 1 = 2 - 2 = 3 - 3 = ...)$,
- uděláme tedy tři pole booleanů.

Dámy – dominance a nezávislost

- Situace je horší, nevíme předem, kolik jich bude,
- ale máme horní a dolní odhady. Pro dominanci i nezávislost nejvýše n, pro nezávislost stačí zkoušet každou dámu do jiného řádku.
- Kupříkladu napřed vždy zkus dámu nepřidat, pak až přidat. Tak ovšem nestačí jen najít první řešení, je potřeba projít všechny kandidáty (nikdo nezaručuje, že to s nejméně dámami najdeme první).

Dámy – dominance a nezávislost

- Obecně funkční možnost: Zkusíme $1 - -n$ dam přidávat na šachovnici všemi možnými způsoby a testovat, kolik polí ohrozíme, nebo zda ohrozíme jinou dámu.
- Kdo jaké triky vymyslí, takové má.
- Většinou nestačí pamatovat si, zda je pole ohrožené, ale kolikrát, abychom věděli, zda po odebrání "současné" dámy zůstane ohrožené nebo ne!

Dominance a nezávislost – zobecnění

- Jiné šachovnice (tórus, Möbiův list, Kleinova láhev),
- jiné figurky (věž, kůň, Maharadža...),
- Více problémů spočívajících v horší analýze, kolik figurek stačí a kolik jich je naopak aspoň potřeba, lezení přes okraj pole...

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),
- $/ \ * \ + \ 10 \ 5 \ - \ 15 \ 4 \ 2$ – tzv. prefixní notace (operátor je před operandy),

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),
- $/ \ * \ + \ 10 \ 5 \ - \ 15 \ 4 \ 2$ – tzv. prefixní notace (operátor je před operandy),
- $/ \ (* \ (+ \ 10 \ 5) \ (- \ 15 \ 4)) \ 2$ (uzávorkování pro větší názornost),

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),
- $/ \ * \ + \ 10 \ 5 \ - \ 15 \ 4 \ 2$ – tzv. prefixní notace (operátor je před operandy),
- $/ \ (* \ (+ \ 10 \ 5) \ (- \ 15 \ 4)) \ 2$ (uzávorkování pro větší názornost),
- $10 \ 5 \ + \ 15 \ 4 \ - \ * \ 2 \ /$ – postfixní notace (operátor je za operandy),

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),
- $/ \ * \ + \ 10 \ 5 \ - \ 15 \ 4 \ 2$ – tzv. prefixní notace (operátor je před operandy),
- $/ \ (* \ (+ \ 10 \ 5) \ (- \ 15 \ 4)) \ 2$ (uzávorkování pro větší názornost),
- $10 \ 5 \ + \ 15 \ 4 \ - \ * \ 2 \ /$ – postfixní notace (operátor je za operandy),
- $((10 \ 5 \ +) \ (15 \ 4 \ -) \ *) \ 2 \ /$ (uzávorkováno),

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),
- $/ \ * \ + \ 10 \ 5 \ - \ 15 \ 4 \ 2$ – tzv. prefixní notace (operátor je před operandy),
- $/ \ (* \ (+ \ 10 \ 5) \ (- \ 15 \ 4)) \ 2$ (uzávorkování pro větší názornost),
- $10 \ 5 \ + \ 15 \ 4 \ - \ * \ 2 \ /$ – postfixní notace (operátor je za operandy),
- $((10 \ 5 \ +) \ (15 \ 4 \ -) \ *) \ 2 \ /$ (uzávorkováno),
- stromem: Ve vrcholu buďto operátor (vrchol má dva syny) nebo operand (číslo, takový vrchol je listem).

Zápis výrazu

- Jakými způsoby lze zapsat aritmetický výraz?
- $(10 + 5) * (15 - 4)/2$ – tzv. infixní notace (operátor je mezi operandy),
- $/ \ * \ + \ 10 \ 5 \ - \ 15 \ 4 \ 2$ – tzv. prefixní notace (operátor je před operandy),
- $/ \ (* \ (+ \ 10 \ 5) \ (- \ 15 \ 4)) \ 2$ (uzávorkování pro větší názornost),
- $10 \ 5 \ + \ 15 \ 4 \ - \ * \ 2 \ /$ – postfixní notace (operátor je za operandy),
- $((10 \ 5 \ +) \ (15 \ 4 \ -) \ *) \ 2 \ /$ (uzávorkováno),
- stromem: Ve vrcholu buďto operátor (vrchol má dva syny) nebo operand (číslo, takový vrchol je listem).
- Pozor, evaluace bude jen naznačena, na slidech bude pseudokód, který je třeba interpretovat s fantazií!

Aritmetické výrazy a různé notace

- Výhody a nevýhody jednotlivých notací...
- Lze všchny tyto notace (zápisy) vyhodnotit?
- Lze mezi těmito notacemi převádět?
- Ano a dokonce velmi snadno pomocí stromového zápisu.

Evaluace prefixní notace

Rekurzí:

```
function vyhodnot:integer;
begin
      if (na vstupu je číslo) then
            vyhodnot:=hodnota_čísla_na_vstupu
      else
      begin operator:=na_vstupu();
            arg1:=vyhodnot;
            arg2:=vyhodnot;
            vyhodnot:=proved(operator,arg1,arg2);
      end;
end;
```

Strom z prefixní notace

```
function pref_strom:strom;
begin
      if(na vstupu je číslo) then
            pref_strom:=list(hodnota_na_vstupu);
      else
      begin pom:=vrchol(operator);
            pom.arg1:=vyhodnot;
            pom.arg2:=vyhodnot;
            vyhodnot:=pom;
      end;
end;
```

funkce `list` vytvoří list,
funkce `vrchol` vytvoří vrchol stupně 2,
vrchol stupně 2 obsahuje prvky `arg1` a `arg2`.

Jak ze stromu vygenerovat všechny notace?

- Rekurzívně:

Jak ze stromu vygenerovat všechny notace?

- Rekurzívně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,

Jak ze stromu vygenerovat všechny notace?

- Rekurzívně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,
- v jedné fázi do pravého syna a v jedné fázi vypíšeme údaj o operátoru.

Jak ze stromu vygenerovat všechny notace?

- Rekurzívně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,
- v jedné fázi do pravého syna a v jedné fázi vypíšeme údaj o operátoru.
- Všechny tři notace získáme správným uspořádáním těchto tří fází.

Jak ze stromu vygenerovat všechny notace?

- Rekurzívně:
- Prolezeme strom tak, že v jedné fázi vlezeme do levého syna,
- v jedné fázi do pravého syna a v jedné fázi vypíšeme údaj o operátoru.
- Všechny tři notace získáme správným uspořádáním těchto tří fází.
- Do pravého syna půjdeme vždy až po návštěvě levého syna, lišit se tedy bude jen okamžik výpisu údajů!

Generování prefixní notace

```
procedure gen_pref(v:vrchol);
begin
      if(list(v)) then
            vypis(v);
      else
      begin vypis(v);
            gen_pref(v.arg1);
            gen_pref(v.arg2);
      end;
end;
```

Funkce `vypis` vypíše operátor resp. číslo.

funkce `list` zjistí, zda je současný vrchol list.

Generování postfixní notace

```
procedure gen_post(v:vrchol);
begin
      if(list(v)) then
            vypis(v);
      else
      begin gen_post(v.arg1);
            gen_post(v.arg2);
            vypis(v);
      end;
end;
```

Funkce `vypis` vypíše operátor resp. číslo.

funkce `list` zjistí, zda je současný vrchol list.

Generování infixní notace

skoro správně

```
procedure gen_post(v:vrchol);
begin
      if(list(v)) then
            vypis(v);
      else
      begin gen_post(v.arg1);
            vypis(v);
            gen_post(v.arg2);
      end;
end;
```

Funkce `vypis` vypíše operátor resp. číslo.

funkce `list` zjistí, zda je současný vrchol list.

Generování infixní notace

správně leč ošklivě

```
procedure gen_post(v:vrchol);
begin
      if(list(v)) then
            vypis(v);
      else
      begin write('(');
            gen_post(v.arg1);
            vypis(v);
            gen_post(v.arg2);
            write(')');
      end;
end;
```

K vyhodnocení postfixní notace

Opakování:
Zásobník je datová struktura osazená operacemi:

- `push` – přidej na konec zásobníku,
- pop – uber z konce zásobníku,
- tedy kdo později přijde, ten je dříve odejit.

Evaluace postfixní notace

```
function eval_post:integer;
begin
      while not eof do
      begin if (na_vstupu_cislo) then
                  push(cislo);
            if (na_vstupu_operator) then
            begin arg2:=pop;
                  arg1:=pop;
                  push(operator(arg1,arg2));
            end;
      end;
      writeln(pop);{Výsledek je na vrchu zásobníku}
end;
```

Strom z postfixní notace

```
function strom_post:vrchol;
begin
      while not eof do
      begin if (na_vstupu_cislo) then
                  push(list(cislo));
            if (na_vstupu_operator) then
            begin pom:=vrchol(operator);
                  pom.arg2:=pop;
                  pom.arg1:=pop;
                  push(pom);
            end;
      end;
      strom_post:=pop;{Výsledek na vrchu zásobníku}
end;
```

Evaluace stromu

je snad jasná, ale přesto:

```
function eval_tree(v:vrchol);
begin
      if(list(v)) then
            eval_tree:=value(v)
      else
      begin arg1:=eval_tree(v.arg1);
            arg2:=eval_tree(v.arg2);
            op:=operator(v);
            eval_tree:=op(arg1,arg2);
      end;
end;
```

Evaluace infixní notace

aneb Masakr u katova stromu

- Jedna možnost je najít operátor, který se provede jako poslední,
- výraz podle něj rozdělit, zavolat se na každou část zvlášť a nakonec provést operátor.
- Výhoda: Po rozmyšlení, jak najít poslední operátor snadno implementovatelné.
- Nevýhoda: Neustále traverzujeme výrazem a hledáme operátory.
- Jak třeba lze najít poslední provedený operátor:
  1. Najdi poslední operátor sčítání nebo odčítání mimo závorky,
  2. pokud nebyl nalezen, hledej poslední operátor násobení mimo závorky,
  3. pokud koukáme na samotné číslo, vyhodnoť ho,
  4. jinak oloupej závorky,
  5. Vyřeš body 1–4 pro výraz uvnitř.

Přednášející jde dnes do M1 potřetí:

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

Přednášející jde dnes do M1 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 M1 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 M1 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;
```