Cvičení Programování I

Cvičící: **Pavel Surynek, KTIML,**
pavel.surynek@seznam.cz
Semestr: **Zima 2005/2006**
Kroužek: **Matematika/59**
Rozvrh: **Pátek 10:40-12:10 (učebna K2)**

Stručné poznámky ke cvičení ze 16.12.2005

1. Organizační záležitosti. První týden po vánočních prázdninách se bude konat schůzka, na které si procvičíme úlohy k praktickému testu. Schůzka se bude konat *nejspíš* v pátek 6.1.2006 v 8:00 v karlínské laboratoři. Zatím se mi ale nepodařilo zamluvit laboratoř (zamlouvací systém má asi potíže s přechodem na nový rok), proto říkám nejspíš. Ale obvykle tam stejně nikdo není, takže to považujme téměř za jisté. Ještě schůzku potvrdím mailem.

Jakékoli **dotazy** ke cvičení lze posílat na uvedenou e-mailovou adresu. Osobní konzultace ke cvičení lze dohodnout e-mailem (alespoň den předem).

Řešení úlohy Koník II. Na jednom z minulých cvičení jsme úlohu vyřešili slovně, tj. sepsali jsme řešící algoritmus neformálně v češtině. Nyní se pokusíme algoritmus formalizovat v Pascalu, tedy učiníme poslední krok v návrhu programu (připomenutí: **1.představa postupu** (obrázky) → **2.postup popsaný slovně → 3. algoritmus v Pascalu**). Připomeňme si ještě jednou myšlenku řešení. Představme si, že jsme již provedli několik tahů koněm. V tomto okamžiku řešíme úlohu, jak projít koněm zbývající políčka šachovnice, tedy ta ještě nenavštívená. Řekněme, že v této situaci zbývá Z políček, které ještě nebyly navštíveny. Uvědomme si, že když $Z = 0$, nebo $Z = 1$, tak je úloha vyřešená, respektive ji umíme snadno dořešit. Mějme ale $Z > 1$. Zkusme učinit tah. Ten tah musí být samozřejmě dovolený, tj. musí jej figurka kůň umět a cílové políčko musí být navštěvované poprvé. Když to uděláme vznikne úloha stejného typu ale menší. Menší v tom smyslu, že zbývá $Z - 1$ nenavštívených políček. Vzniklou úlohu zkusíme vyřešit, když se to povede, máme řešení. K řešení menší úlohy přidáme provedený tah a to je řešení úlohy původní. Když se to ale nepovede, je třeba tah zrušit a zkoušet jiný podle toho, jaké další tahy figurka kůň umí a také podle toho, kam ještě může vstoupit. Když se menší úlohu nepodaří vyřešit ani v jednom případě, znamená to, že původní úloha nemá řešení.

Poznámka: program uvedený zde zkouší začínat posloupnost tahů ze všech možných pozic, na cvičení jsme ukazovali program, kdy zkoušíme začínat jen z pozice 1,1. **Program** v Pascalu tedy vypadá následovně (místy je mírně komentovaný):

```
program Konik2;

const VELIKOST = 6; { velikost sachovnice }

type POZICE = record
                x, y: integer;
              end;

const tahy_kone:array[1..8] of POZICE = ((x:-2; y:-1),(x:-1; y:-2),
                                         (x:-2; y: 1),(x:-1; y: 2),
                                         (x: 2; y:-1),(x: 1; y:-2),
                                         (x: 2; y: 1),(x: 1; y: 2));
  { pole s moznymi tahy sachove figurky kun (relativne vzhledem
    k soucasne pozici kone), vsimneme si inicializace jednotlivych
    prvku recordu }
```

```
var sachovnice: array[1..VELIKOST, 1..VELIKOST] of integer;
  { hraci pole, kazda bunka odpovida policku na sachovnici,
    bunka obsahuje poradove cislo tahu, kdy byla navstivena,
    0 v bunce znamena, ze odpovidajici policko jeste nebylo
    navstivene }


function vyres(vychozi: POZICE; zbyva: integer): boolean;
  { parametr zbyva udava pocet zbyvajicich tahu,
    parametr vychozi urcuje soucasnou pozici figurky }
var uspech: boolean;
var i: integer;
var nova: POZICE;
begin
  if zbyva = 0 then begin
    vyres := true;
  end
  else begin
    uspech := false;
    for i:=1 to 8 do begin
      nova.x := vychozi.x + tahy_kone[i].x;
      nova.y := vychozi.y + tahy_kone[i].y;
      if     (nova.x >= 1) and (nova.x <= VELIKOST)
         and (nova.y >= 1) and (nova.y <= VELIKOST) then begin
         if sachovnice[nova.x][nova.y] = 0 then begin
           sachovnice[nova.x][nova.y] := VELIKOST*VELIKOST-zbyva+1;
           if vyres(nova, zbyva-1) then begin
             uspech := true;
             break;
           end;
           sachovnice[nova.x][nova.y] := 0;
         end;
      end;
    end;
    vyres := uspech;
  end;
end;
```

Toto je „jádro algoritmu“, zde je formálně zapsaná myšlenka řešení.

```
procedure inicializuj;
  { vyplni hraci pole nulami, coz znamena, ze zadne
    policko jeste nebylo navstiveno }
var i, j: integer;
begin
  for i:=1 to VELIKOST do begin
    for j:=1 to VELIKOST do begin
      sachovnice[i][j] := 0;
    end;
  end;
end;
```

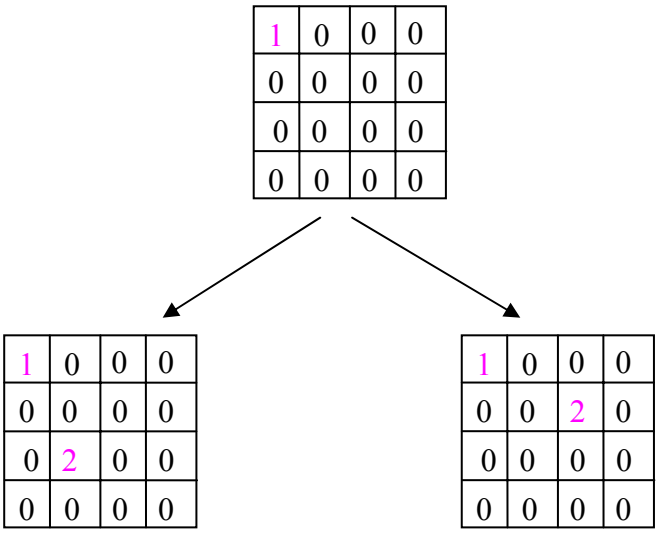
```
procedure vypis;
  { vypise poradova cisla tahu na jednotlivych
    policka sachovnice }
var i, j: integer;
begin
  for i:=1 to VELIKOST do begin
    for j:=1 to VELIKOST do begin
      write(sachovnice[i][j]:3); { vypisujeme vzdy 3 znaky,
                                   doplnuje se mezerami }
    end;
    writeln;
  end;
end;


var uspech: boolean;
var i, j: integer;
var pocatek: POZICE;
begin
  inicializuj;
  uspech := false;

  for i:=1 to VELIKOST do begin
    for j:=1 to VELIKOST do begin
      writeln('Zkousim zacit na pozici ', i, ',', j);
      sachovnice[i][j] := 1;
      pocatek.x := i;
      pocatek.y := j;
      if vyres(pocatek, VELIKOST * VELIKOST - 1) then begin
        uspech := true;
        writeln('Reseni je:');
        vypis;
        halt(1);
      end;
      sachovnice[i][j] := 0;
    end;
  end;
  if not uspech then begin
    writeln('Reseni se nepodarilo nalezt.');
  end;
end.
```

Program řeší úlohu pro šachovnici velikosti 6×6. Konkrétní řešení je následující:

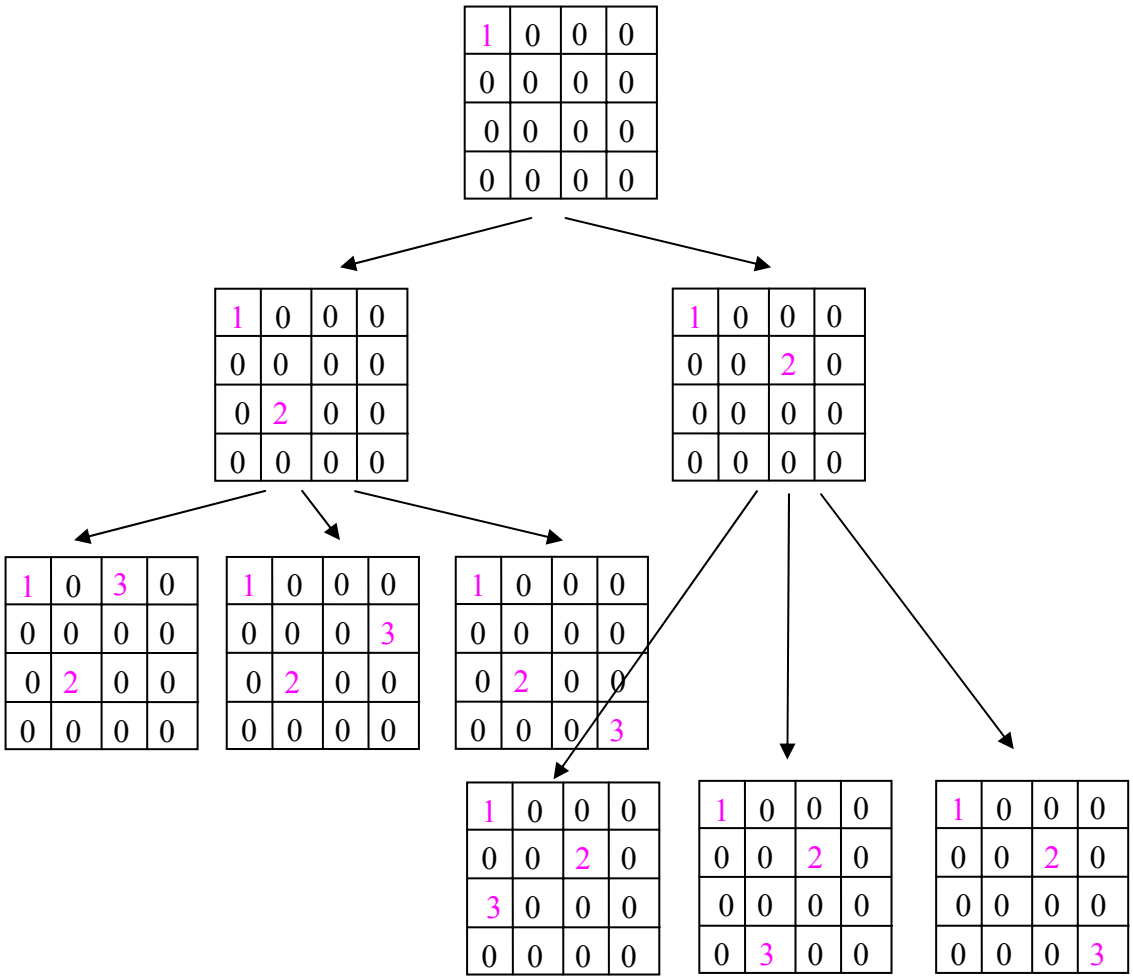
1	18	3	10	7	16
4	9	6	17	32	11
19	2	31	8	15	24
30	5	28	23	12	33
27	20	35	14	25	22
36	29	26	21	34	13

Všimněme si, že vyřešit tuto úlohu „ručně“ není zas tak úplně snadné, jinými slovy někdy se vyplatí něco řešit pomocí počítače. Úlohu lze také vyřešit pro velikosti šachovnice 5×5 a 7×7. Pro velikost 8×8 program běžel déle, než jsem byl ochoten čekat, nicméně pro tuto velikost řešení také existuje. **Vysvětlete** jak je to s časovou složitostí algoritmu.

Abychom program lépe pochopili, nakreslíme si, jak postupuje při hledání řešení. Pro jednoduchost budeme „kreslit“ úlohu pro velikost šachovnice 4×4 (mimochodem pro tuto velikost šachovnice řešení neexistuje). Budeme znázorňovat pole v programu označené jako *šachovnice*. Začneme na pozici 1,1, to znamená, že na pozici 1,1 napíšeme 1 (tuto pozici navštěvujeme v prvním tahu). Zakreslíme všechna možná pokračování z této situace.

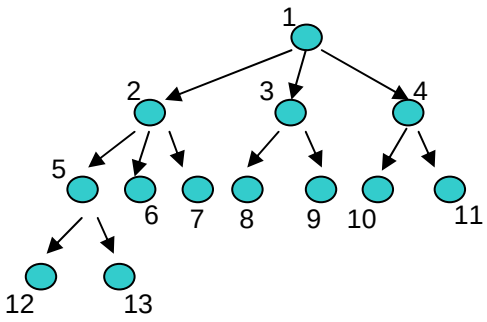


Stejným způsobem nakreslíme všechna možná pokračování z nových situací.



Takto pokračujeme tak dlouho, dokud je možné někde kreslit nějaká pokračování. Všimněme si několika věcí. To co kreslíme, je strom. Každý vrchol tohoto stromu obsahuje možnou posloupnost tahů. Ve stromu jsou obsaženy všechny možné posloupnosti tahů, které začínají na pozici 1,1. Strom tedy obsahuje i řešící posloupnosti (pokud ovšem řešení existuje). Jelikož řešící posloupnost vyplňuje tahy celou šachovnici, bude se vrchol obsahující řešení (případně vrcholy obsahující řešení, když je řešení více) nacházet na nejspodnější hladině stromu. Větve stromu (posloupnost spojnic od nejvrchnějšího vrcholu k jednomu z nejspodnějších vrcholů) mají různou délku, větve maximální délky mají na konci řešení, kratší větve mají na konci situaci, ze které už nejde pokračovat (sice ještě existují nenavštívená políčka, ale figurka už je „uvězněná“).

Podívejme se nyní, jak algoritmus s tímto stromem pracuje. Pro zjednodušení si vrcholy stromu znázorníme jako kroužky. A očíslujeme je, viz. následující obrázek. Každý kroužek nyní reprezentuje šachovnici s tahy jako na předchozím obrázku.



Algoritmus projde jednotlivé situace přesně v pořadí 1, 2, 5, 12, 5, 13, 5, 2, 6, 2, 7, 2, 1, 3, 8, 3, 9, 3, 1, 4, 10, 4, 11, 4, 1. Pokud existuje řešení, může procházení stromu skončit dříve, vypsané pořadí zkoumaných situací tedy odpovídá situaci, kdy neexistuje řešení. Poznámka: pořadí vrcholů odpovídá pořadí, které bychom získali „obtažením“ stromu čarou, přičemž číslo vrcholu bychom zapisovali pokaždé, když čára prochází okolo vrcholu.