Procházení stavového prostoru

Základy algoritmizace – 8. cvičení

Zuzana Petříčková

8. dubna 2020

Stavový prostor problému a jeho procházení

Typické příklady úloh:

- hlavolamy a hry (nejen) pro jednoho hráče (sudoku, šachy, aj.)

Úloha:

- **Zadání:** je dán počáteční stav světa (např. zadání sudoku) a množina cílových stavů (přípustná řešení problému)
- **Cíl:** pomocí nějakých akcí / tahů se potřebujeme dostat ze startu (počátečního stavu) do cíle (jednoho z cílových stavů)

Stavový prostor úlohy

- speciální typ grafu:
  - vrcholy = stavy (situace)
  - hrany = přechody mezi stavy (akce, tahy)

  $\rightarrow$ stavový strom (nebo obecnější graf) úlohy

Stavový prostor problému a jeho procházení

Samostatná práce: nastudujte si skripta:

- kapitola 3.4 (Metoda hledání s návratem), strana 92-96, podrobněji nastudujte příklad 3.7 (Úloha osmi dam, úloha 4 dam)
- kapitola 3.5 (Obecné metody prohledávání stavového prostoru), strana 97-98

Ukázka stavového stromu pro úlohu 4 dam na šachovnici:

- skripta, kapitola 3.4, strana 95

Procházení stavovým prostorem (grafem) úlohy

- (na rozdíl od běžného grafu): v paměti nemáme uložen celý stavový strom (graf), ale vytváříme ho postupně průchodem
  1. do hloubky ... metoda prohledávání s návratem (backtracking)
  2. do šířky ... algoritmus vlny
- algoritmy pro průchod budou hodně podobné jako u běžného grafu (ukážeme za chvíli)
- časová složitost ... záleží na implementaci (v ideálním případě $O(n+m)$, ale může být i exponenciální $O({m_i}^n)$)
  - n je počet stavů, m je počet ,,hran" celkem, m_i je počet tahů z jednoho stavu

Procházení stavovým prostorem (grafem) úlohy Motivační příklad: vlk, koza a zelí

Úloha:

- na začátku je zahradník, vlk, koza, zelí a loďka na jednom břehu řeky
- cílem zahradníka je převést vlka, kozu a zelí na druhý břeh
- zahradník má k dispozici loďku, do které se kromě něho samotného vejde jen jedna z ,,věcí"

Pravidla:

- loďka nemůže plout bez zahradníka
- pokud zůstane vlk s kozou bez dozoru, vlk kozu sežere
- pokud zůstane zelí s kozou bez dozoru, koza zelí sežere

Procházení stavovým prostorem (grafem) úlohy Motivační příklad: vlk, koza a zelí

Jak budeme reprezentovat stav světa?

- počáteční stav ... L: VKZL, P: $\emptyset$ (na levém břehu (L:) je vlk (V), koza (K), zelí (Z) a zahralnik s loďkou (L), na pravém břehu (P:) není nic ($\emptyset$))

Jaké jsou možné tahy (obecně)?

1. $\emptyset$ ... zahradník přepluje na druhý břeh sám ($L \rightarrow P$ nebo $P \rightarrow L$)
2. Z ... zahradník převeze zelí ($L \rightarrow P$ nebo $P \rightarrow L$)
3. V ... zahradník převeze vlka ($L \rightarrow P$ nebo $P \rightarrow L$)
4. K ... zahradník převeze kozu ($L \rightarrow P$ nebo $P \rightarrow L$)

$\rightarrow$ tahů je celkem 8 (nebo 4, pokud nerozlišujeme směr)

Procházení stavovým prostorem (grafem) úlohy Motivační příklad: vlk, koza a zelí

Jaké jsou možné tahy z počátečního stavu L: VKZL, P: ∅?

1. zahradník přepluje sám → L: VKZ, P: L ... nepřípustný tah (koza sežere zelí a vlk sežere kozu)
2. zahradník převeze zelí → L: VK, P: ZL ... nepřípustný tah (vlk sežere kozu)
3. zahradník převeze vlka → L: KZ, P: VL ... nepřípustný tah (koza sežere zelí)
4. zahradník převeze kozu → L: VZ, P: KL ... přípustný tah

Procházení stavovým prostorem (grafem) úlohy Motivační příklad: vlk, koza a zelí

Jaké jsou přípustné tahy ze stavu L: VZ, P: KL?

1. zahradník přepluje sám → L: VZL, P: K ... přípustný tah
2. zahradník převeze kozu → L: VKZL, P: ∅ ... jsme zpátky v předchozím stavu

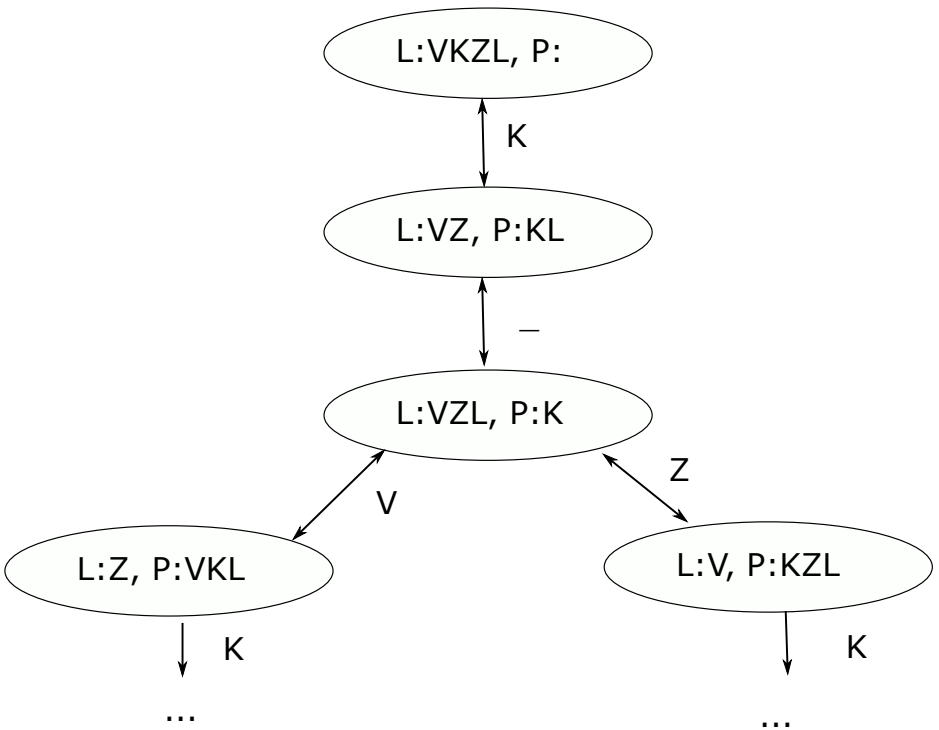
Jaké jsou přípustné tahy ze stavu L: VZL, P: K?

1. zahradník přepluje sám → L: VZ, P: KL ... ... jsme zpátky v předchozím stavu
2. zahradník převeze zelí → L: V, P:KZL ... přípustný tah
3. zahradník převeze vlka → L: Z, P:VKL ... přípustný tah

→ postupně budujeme stavový strom úlohy

Procházení stavovým prostorem (grafem) úlohy Motivační příklad: vlk, koza a zelí

Stavový strom úlohy:



Procházení stavovým prostorem (grafem) úlohy Motivační příklad: vlk, koza a zelí

Samostudium

- Zkuste úlohu vyřešit až do konce a dokreslit celý stavový strom
- Zamyslete se nad tím, jak byste stavy a tahy reprezentovali v programu (pro reprezentaci stavů by stačilo pole 0 a 1 délky 4)
- Jak by při Vámi zvolené reprezentaci stavů program detekoval nepřípustné stavy (tahy)?

Procházení stavovým prostorem do hloubky s navracením (BACKTRACKING)

Možnosti

1. procházení do hloubky s využitím rekurze
2. procházení do hloubky s využitím zásobníku

Rozlišujeme dvě možnosti:

1. stačí nám nalézt jen jedno řešení úlohy
2. chceme postupně nalézt všechna řešení úlohy

Procházení stavovým prostorem do hloubky s navracením (BACKTRACKING) s využitím rekurze

V průběhu algoritmu

- postupně budujeme cestu z pořátečního stavu s do některého z cílových stavů
- do cesty zaznamenáváme, kterými stavy jsme zatím prošli a které tahy jsme přitom provedli:

  $$s, t_1, s_1, t_1, s_2, t_2, ..., t_k, s_k$$

  (s je počáteční stav, z něj jsme pomocí tahu t_1 přešli do stavu s_1, z něj jsme pak pomocí tahu t_2 přešli do stavu s_2, atd., až jsme nakonec došli pomocí tahu t_k do cílového stavu s_k)
- v cestě se žádný stav nesmí opakovat vícekrát $\rightarrow$ abychom se nezacyklili

Procházení stavovým prostorem do hloubky s navracením (BACKTRACKING) s využitím rekurze

Jeden krok rekurze:

- jsme v nějakém aktuálním stavu u
- pokud je to cílový stav, zaznamenáme řešení (aktuální cestu)
- jinak vyzkoušíme postupně všechny přípustné tahy z u
  - přejdeme pomocí tahu t do dalšího stavu v ... **krok dopředu** (vložíme t a v na konec cesty)
  - zavoláme metodu rekurzivně na stav t
  - po návratu z rekurze ,,vrátíme" tah t ... **krok zpět (BACKTRACKING)** (vyjmeme t a v z konce cesty)

Procházení stavovým prostorem do hloubky s navracením (BACKTRACKING) s využitím rekurze

vstup: stavový prostor s množinou stavů V, počáteční stav $s \in V$
výstup/proměnná: cesta $= \{s, ..\}$ ... posloupnost stavů a tahů na aktuální cestě k řešení
rekurze: DFS(s)

```
DFS(u) ... u je aktuální stav
  if u je koncový stav
    zaznamenej řešení (cestu)
    return TRUE/FALSE ... TRUE: stačí mi jedno řešení / FALSE: chci všechna
  endif
 ∀t ∈ všechny přípustné tahy z u do
   v je nový stav po provedení tahu t
   if (v ∉ cesta) ... pokud v není součástí aktuální cesty
      vlož t a v na konec cesty ... krok dopředu
      if DFS(v)
        return TRUE
      endif
      vyjmi v a t z konce cesty ... krok zpět (BACKTRACKING)
   endif
  enddo
  return FALSE
```

Procházení stavovým prostorem do hloubky s navracením (BACKTRACKING) s využitím zásobníku

- rekurze může vést až na exponenciální časovou složitost $O(m_i^n)$ (n je počet stavů, m_i je počet možných tahů z jednoho stavu)
- rekurze se (podobně jako u obecného grafu) můžeme zbavit s využitím zásobníku

Zásobník

- prvky zásobníku budou trojice (v,t,T):
  - v ... aktuální stav
  - t ... tah, kterým jsme se dostali do stavu v
  - T ... množina dosud neprozkoumaných přípustných tahů z v

Procházení stavovým prostorem do hloubky s navracením (BACKTRACKING) s využitím zásobníku

vstup: stavový prostor s množinou stavů V, počáteční stav $s \in V$
proměnná/výstup: zásobník (cesta) ... posloupnost stavů a tahů na aktuální cestě k řešení, prvky zásobníku jsou trojice (v,t,T) (viz minulý snímek)
nechť T je množina přípustných tahů z s, zásobník $\leftarrow (s, \_, T)$
if s je koncový stav
  zaznamenej řešení, případně return... pokud stačí jedno řešení
endif
while zásobník není prázdný **do**
  nechť na vrchu zásobníku je prvek (u, t_u, T_u) ... u je aktuální stav
  if $T_u == \emptyset$
    vyjmi (u, t_u, T_u) z vrchu zásobníku, continue ... krok zpět (BACKTRACKING)
  else
    vyjmi z T další tah t, nechť v je nový stav po provedení tahu t
    if $v \notin$ zásobník ... pokud v není součástí aktuální cesty
      if v je koncový stav
        zaznamenej řešení, případně return ... pokud stačí jedno řešení
      endif
      nechť T_v je množina přípustných tahů z v
      vlož (v, t, T_v) na zásobník ... krok dopředu
    endif
  endif enddo

Procházení stavovým prostorem do šířky s využitím fronty (algoritmus vlny)

- pro úlohy, kdy hledáme nejkratší cestu (min. počet tahů) k řešení
- použijeme dvě pomocné datové struktury: frontu a množinu navštívených stavů (můžeme ji implementovat jako pole *a*, *p*, *d* z minule)

Fronta

- prvky fronty budou vrcholy

Množina navštívených stavů

- prvky množiny budou trojice:
  - v ... nějaký stav
  - d ... vzdálenost v od s (počet tahů, pomocí kterých přejdu z s do v)
  - p ... stav, předchůdce v na jedné z cest z s do v

Procházení stavovým prostorem do šířky s využitím fronty (algoritmus vlny)

vstup: stavový prostor s množinou stavů V, počáteční stav $s \in V$
proměnné:
$M = \{\}$ ... množina navštívených stavů, obsahuje trojice (v,d,p)
v ... stav, d ... vzdálenost v od s
p ... předchůdce v na cestě z s do v

přidej (s,0,-1) do M
fronta $\leftarrow$ s
while fronta není prázdná **do**
  u $\leftarrow$ fronta
  najdi v M záznam pro vrchol u: $(u, d, \_)$
  $\forall t \in$ všechny přípustné tahy z u **do**
    v je nový stav po provedení tahu t
    if v M není záznam pro v $((v, \_, \_) \notin M)$
      přidej (v,d+1,u) do M
      fronta $\leftarrow$ v
    endif
  enddo
endddo

Procházení stavovým prostorem do hloubky a do šířky

- pro konkrétní úlohy si často můžeme obecné algoritmy pro procházení stavovým prostorem do hloubky a do šířky zjednodušit a přizpůsobit
- ukážeme si to na následujících hříčkách

Příklady (úlohy nad šachovnicí):

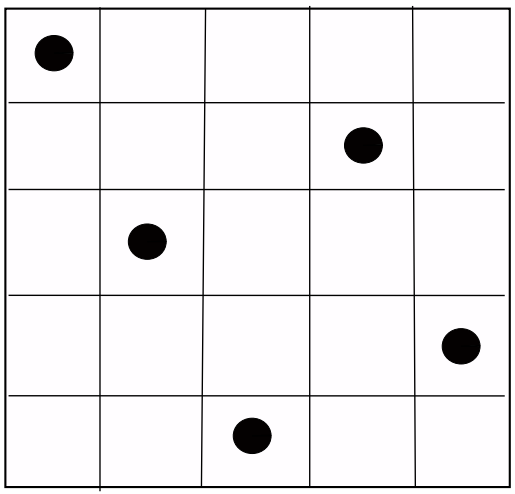
- Problém n dam
- Kůň na šachovnici
  1. Procházka koněm po šachovnici
  2. Nejkratší cesty koněm z jednoho pole na ostatní

(dají se vymyslet různá další rozšíření a modifikace těchto úloh, např. ,,kulhavý" kůň)

Příklad 1: Problém n dam

Úloha

- **vstup:** číslo *n* (rozměr šachovnice, počet dam)
- **cíl:** rozmístit na šachovnici *nxn* *n* dam tak, aby se navzájem neohrožovaly
  1. stačí mi jedno řešení úlohy
  2. nebo chci najít všechna řešení úlohy



Příklad 1: Problém n dam

Pravidla (dámy se nesmí navzájem ohrožovat)

- v každém řádku šachovnice musí být nejvýše (právě) jedna dáma
- v každém sloupci šachovnice musí být nejvýše (právě) jedna dáma
- na každé diagonále šachovnice musí být nejvýše (právě) jedna dáma

1. Naivní řešení: metoda generuj a testuj

- generujeme všechna možná umístění n dam na šachovnici
- pro každé řešení ověříme, zda je přípustné (dámy se neohrožují)
- časová složitost $\binom{n^2}{n}$

Příklad 1: Problém n dam

2. Lepší řešení: backtracking

- postupujeme po sloupcích matice (šachovnice) zleva doprava
- do aktuálního sloupce se pokusíme umístit na nějaký řádek dámu
  1. procházíme jednotlivé pozice ve sloupci zhora dolů a pokusíme se na danou pozici umístit dámu
  2. pokud se nám to podaří, postupujeme na další sloupec
  3. pokud se nám nepodaří umístit dámu na žádnou pozici ve sloupci, vrátíme se k předchozímu sloupci
- časová složitost zhora omezená $O(n!)$

Příklad 1: Problém n dam, rekurzivně I

Jak budeme reprezentovat řešení úlohy?

- sachovnice ... matice *nxn* s hodnotami true (na dané pozici je dáma) a false (na dané pozici není dáma)

Algoritmus (procházení do hloubky s navrazením s využitím rekurze)

vstup:

n ... rozměr šachovnice

jenJedno ... true/false (zda chci jedno nebo všechna řešení)

proměnné:

sachovnice ... matice boolovských hodnot *nxn* (šachovnice)

$\forall i, j : sachovnice[i][j] = false$

vyřeš(0, jenJedno) ... zavolej rekurzivní funkci na první sloupec

Příklad 1: Problém n dam, rekurzivně II

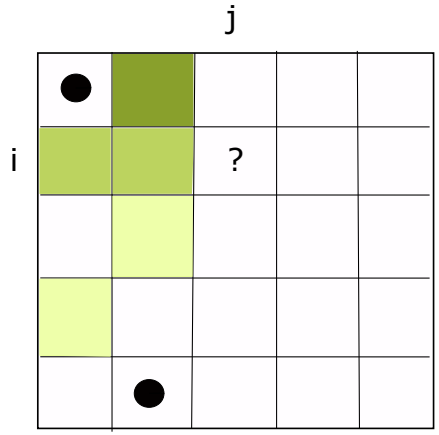
```
vyřeš(index sloupce j, bool jenJedno)
  if j ≥ n ... hotovo
    zaznamenej řešení (matici sachovnice)
    return jenJedno
  endif
  for i = 1 to n do... projdeme pozice ve sloupci
    if [i,j] je bezpečná pozice
        sachovnice[i][j] = true ... krok DOPŘEDU
        if vyřeš(j + 1,jenJedno)
          return true
        endif
        sachovnice[i][j] = false ... krok ZPĚT
   endif
  enddo
  return false
```

Příklad 1: Problém n dam, rekurzivně III

Jak poznám, že je $[i, j]$ je bezpečná pozice pro položení dámy?

1. na i-tém řádku od indexu 0 až do indexu $(j - 1)$ není dáma
2. na příslušné horní ani dolní diagonále až do indexu sloupce $(j - 1)$ není dáma

(zbylé pozice není třeba kontrolovat)



Příklad 1: Problém n dam

Na rozmyšlenou

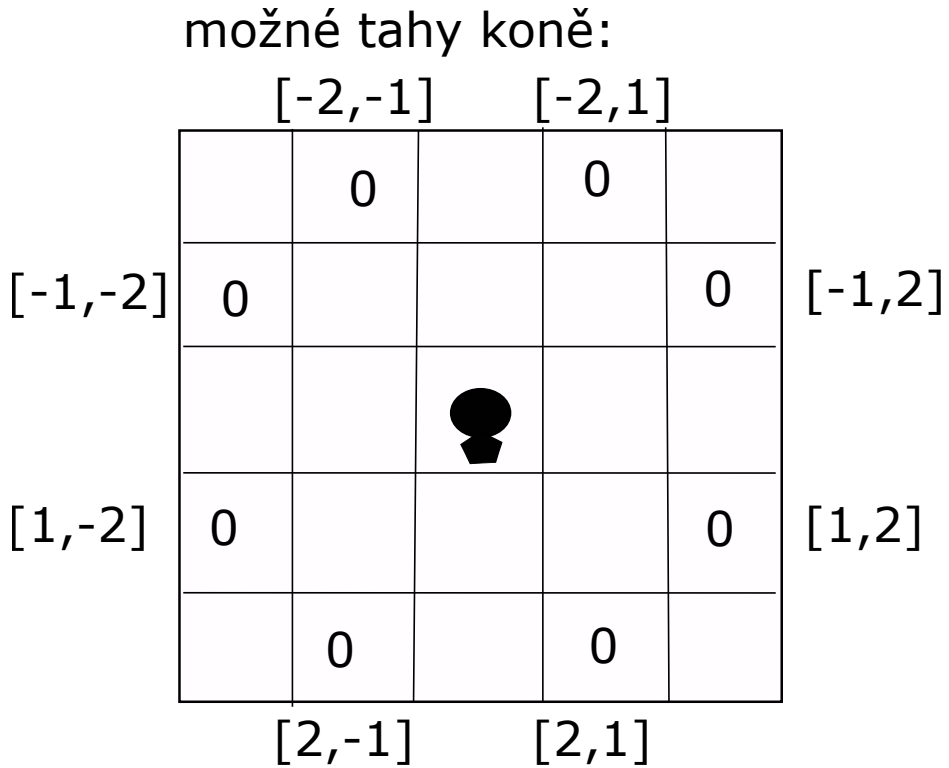
- zkuste algoritmus upravit tak, aby místo rekurze využíval zásobník

Příklad 2: Kůň na šachovnici

- **vstup:** šachovnice *nxn*, počáteční pozice koně
- **cíl:**
  1. nalézt procházku koněm po šachovnici (cestu koněm, kdy navštíví každé pole právě jednou)
     $\rightarrow$ procházení stavovým prostorem do hloubky s navracením (backtracking)
  2. nalézt nejkratší cesty koněm na všechna ostatní pole šachovnice
     $\rightarrow$ procházení stavovým prostorem do šířky

Příklad 2: Kůň na šachovnici

- z každé pozice na šachovnici je max. 8 možných tahů koněm:



Příklad 2: Kůň na šachovnici

1. Procházka koněm po šachovnici

- jak si jednoduše zapamatovat cestu (posloupnost vrcholů a tahů na cestě k řešení)?
  $\rightarrow$ přímo na šachovnici (plus pomocná matice s předchůdci)
- časová složitost $O(8n^2)$ (z každého políčka je max 8 možných tahů)

cesta ze stavu [0,0]:

0		4		
5				3
	1			
			2	

předchůdci stavů:

[-1,-1]		[1,4]		
[0,2]				[3,3]
	[0,0]			
			[2,1]	

Příklad 2: Procházka koněm po šachovnici, rekurzivně I

Algoritmus (procházení do hloubky s navrazením s využitím rekurze)
vstup:
n ... rozměr šachovnice
$s = [s_x, s_y]$ ... počáteční pozice koně
tahy ... pole dvojic délky 8 (možné tahy koněm)
jenJedno ... true/false (zda chci jedno nebo všechna řešení)
proměnné:
sachovnice ... matice čísel nxn (cesta koněm)
p ... matice dvojic (souřadnic) nxn (předchozí pozice koně po cestě)

$\forall i, j : sachovnice[i][j] = -1$
$\forall i, j : p[i][j] = [-1, -1]$
$sachovnice[s_x][s_y] = 0$
vyřeš(s_x, s_y, jenJedno) ... zavolej rekurzivní funkci

Příklad 2: Procházka koněm po šachovnici, rekurzivně II

vyřeš(u_x, u_y, jenJedno)
  if $sachovnice[u_x][u_y] = n^2 - 1$
    zaznamenej cestu (matici sachovnice a p)
    return jenJedno
  endif
  $\forall t \in tahy$ **do**
    $[v_x, v_y]$ je nová pozice koně po provedení tahu t
    if $[v_x, v_y]$ je uvnitř šachovnice a $sachovnice[v_x][v_y] = -1$ ... pole je nenavštívené

      $sachovnice[v_x][v_y] = sachovnice[u_x][u_y] + 1$ ... krok DOPŘEDU
      $p[v_x][v_y] = [u_x, u_y]$

      if vyřeš(v_x, v_y)
        return true
      endif

      $sachovnice[v_x][v_y] = -1$ ... krok ZPĚT
      $p[v_x][v_y] = [-1, -1]$
    endif
  enddo
  return false

Příklad 2: Procházka koněm po šachovnici

Na rozmyšlenou

- zkuste algoritmus upravit tak, aby místo rekurze využíval zásobník

Příklad 2: Kůň na šachovnici

2. Nejkratší cesty koněm

- úlohu budeme řešit procházením do šířky (algoritmus vlny)
- jak si jednoduše zapamatovat délky nejkratších cest?
  $\rightarrow$ přímo na šachovnici (plus pomocná matice s předchůdci)
- jak evidovat všechna pole s hodnotou k?
  1. prohledáváním šachovnice ... v čase $O(n^2)$
     $\rightarrow$ celková časová složitost ... $O(8n^4)$
  2. ve frontě
     $\rightarrow$ celková časová složitost ... až $O(8n^2)$

vlna ze stavu [0,0]:

0		2		2
		1	2	
2	1			2
	2		2	
2		2		

předchůdci stavů:

[-1,-1]		[2,1]		[1,2]
		[0,0]	[2,1]	
[1,2]	[0,0]			[1,2]
	[1,2]		[2,1]	
[2,1]		[2,1]		

Příklad 2.2: Nejkratší cesty koněm po šachovnici

Algoritmus (procházení do šířky s využitím fronty)

1. prvky fronty budou pozice $[i, j]$

Příklad 2.2: Nejkratší cesty koněm po šachovnici

vstup: n ... rozměr šachovnice
$s = [s_x, s_y]$ ... počáteční pozice koně
tahy ... pole dvojic délky 8 (možné tahy koněm)
proměnné:
sachovnice ... matice čísel nxn (cesta koněm)
p ... matice dvojic (souřadnic) nxn (předchozí pozice koně po cestě)
fronta ... prvky budou dvojice (souřadnice)

$\forall i, j : sachovnice[i][j] = -1$
$\forall i, j : p[i][j] = [-1, -1]$
$sachovnice[s_x][s_y] = 0$
fronta $\leftarrow [s_x, s_y]$
while fronta není prázdná **do**
  $[u_x, u_y] \leftarrow$ fronta
  $\forall t \in tahy$ **do**
    $[v_x, v_y]$ je nová pozice koně po provedení tahu t
    if pozice je uvnitř šachovnice a $sachovnice[v_x][v_y] = -1$ ... pole je nenavštívené
      $sachovnice[v_x][v_y] = sachovnice[u_x][u_y] + 1$
      $p[v_x][v_y] = [u_x, u_y]$
      fronta $\leftarrow [v_x, v_y]$
    endif
  enddo
endddo