# Zápočtový program SOKOBAN - dokumentace

Vojtěch Lanz

14. února 2019

# 0. Obsah

# 1. Úvod

Existuje celá řada témat, která by se dala pojmout jako zápočtový program. Já jsem hledal spíše takové zadání, které by mělo být více přemýšlející a řešící nějaké logické problémy. Proto jsem si zvolil program, co bude řešit známou japonskou logickou počítačovou hru z roku 1980, která se nazývá Sokoban.

# 2. Pravidla hry Sokoban

Sokoban je hra, která probíhá na mřížkované dvourozměrné ploše, kterou nazýváme mapou. Na dané mapě se nachází právě jeden hráč, kterým my smíme pohybovat. Pohybovat se smíme pouze na vedlejší políčko sousedící hranou s políčkem jeho aktuální pozice. Tedy máme čtyři možnosti, kam se posunout - na sever, na jih, na východ či na západ.

Mapa také obsahuje $n$ krabic. Na jednom políčku (jedné mřížce) může být maximálně jedna krabice. Některých $n$ políček mapy jsou speciálně označená. Naším cílem je dostat všechny krabice na označená políčka. A samozřejmě jsou na mapě i různé překážky, na které se hráč, ani krabice nemůžou posunout.

Pravidla posouvání krabic jsou následující: krabici posunu tím způsobem, že se hráčem pokusím dostat na její pozici. Krabice se posune stejným směrem o políčko dál. Pokud je ovšem takové políčko obsazené jinou krabicí, nebo na políčku leží překážka, pak se hráč ani krabice nikam neposouvají.

# 3. Přesné zadání a chování prorgramu

Program má za úkol vyřešit mapu Sokobanu vloženou uživatelem do vedlejšího textového souboru. Vstup je omezen pouze na mapy, jejichž rozměry jsou menší, než 25x25 mřížek. Hra umí ovšem také vyřešit pouze takové mapy, které se dají vyřešit za méně, než 150 posunů krabicemi. Program by i takové situace uměl vyřešit, ovšem by to trvalo velmi dlouhou dobu. Jako výstup přicházejí kroky, které vedou k dostání všech krabic na speciálně vyznačené pozice.

# 4. Jak funguje program jako takový

## 4.1 Zvolený algoritmus

Program víceméně funguje na základním algoritmu, kde hrubou silou procházíme všechny možnosti. Tedy nejdříve si program najde pozice všech krabic a kam je hráč může z aktuální pozice posunout. Rekurzivně prochází všechny možnosti posunů krabic (zatím ignoruje pohyby hráče, pouze řeší, zda hráč umí danou krabici posunout daným směrem), které dávají smysl. To že nějaký posun smysl nedává znamená, že buď už se v rekurzi dříve naskytla stejná situace, a nebo daný posun jasně nevede k cíli. Tedy po takovém posunu je nemožné vyhrát. Což nastává v moment, kdy se nějaká krabice dostane do rohu, k nějaké stěně, od které se neumí dostat, nebo se více krabic dostanou k sobě a tím se navzájem zablokují. V moment, kdy rekurze zjistí, že neexistuje žádná krabice, která by byla mimo vyznačená místa, se zastaví a vrací se zpět, kde v průběhu si ukládá ty pozice, kterýma se k výhře dostala. Teď je už jen potřeba vyřešit posuny hráčů mezi jednotlivými posuny krabic. Vzhledem k tomu, že vím, kterým směrem musím krabici posunout, tak vím, ze které strany musím ke krabici hráčem přistoupit. Tedy je potřeba pro každé 2 po sobě jdoucí posuny krabicemi najít cestu hráče od jednoho posunu k druhému. To řeší algoritmus $A^*$. Po nalezení dané cesty si opět cestu zapamatuje a uživatelovi vypíše přesný postup samotného hráče.

## 4.2 Používané podprogramy a datové struktury

V programu se pracuje s několika novými typy proměnných. První takový je *map*. Jedná se o dvourozměrné pole charu, které do sebe zaznamenává jakoukoliv herní situaci. Obdobný typ je $valueMap$, která do sebe ukládá pro konkrétní souřadnice číselné hodnoty. Souřadnice k mapám se ukládají do typu $field$, což je pole o dvou prvcích - souřadnice $x$ a $y$. Dále si budeme chtít pamatovat, které situace už v průběhu rekurze nastaly. To si zaznamenává program do spojového seznamu reprezentovaným recordem $Situace$. Record $Krabice$ naopak řeší pouze aktuální vlastnosti krabic, tedy jedná se opět o spojový seznam všech krabic, kde zrovna jsou a jak se k nim dá z pohledu hráče přistupovat. Record $VyherniKroky$ je spojový seznam obsahující po sobě jdoucí mapy takovým způsobem, jak pro výhru potřebujeme

posouvat krabicemi. Proto máme ještě record $nejkrCesta$, která pro každé 2 vedlejší mapy z $VyherniKroky$ obsahuje mapy posouvající hráče.

Kód obsahuje jeden hlavní program, ve kterém běží cyklus skenování map z textového souboru, řešení daného problému a vypsání výsledku. Načítání map dělá funkce $function\ nactiMapu() : map;$. Zmíněná rekurze prohledávající všechny tahy krabic, co dávají smysl, je procedura $procedure\ JduToVyhrat();$. Jestli daný tah dává smysl řeší pomocná funkce $function\ muzuVyhrat() : boolean;$. Po doběhnutí rekurze hlavní program spustí $procedure\ ukazKroky();$. Tato procedura spustí cyklus běžící do té doby, dokud neprojede všechny prvky spojového seznamu $VyherniKroky$. Pro každé 2 vedlejší kroky zavolá funkci $function\ nejkratsiCesta() : Pnejkrcesta;$, která zpět vrací spojový seznam kroků hráče mezi danými kroky krabic. Kroky hráče vypíše zavoláním procedury $procedure\ chciVypsatPandulaky();$. Po dokončení vypisování se hlavní program podívá, zda byla aktuálně řešená mapa poslední v textovém souboru. Pokud ano, program se ukončí. Pokud ne, načte novou mapu a řeší další problém.

# 5. Reprezentace dat pro uživatele

## 5.1 Reprezentace vsuptních dat a jejich příprava

Vstupní data se zapisují do textového souboru $maps.TXT.$, který se nachází ve složce s programem. Každá vložená mapa musí začínat řádkem obsahující symbol # s názvem mapy. Na dalších řádcích bude už samotná mapa. Pokud je daná mapa v souboru poslední, musí se na další 2 řádky napsat opět symbol #. Tedy poslední 2 řádky souboru musí obsahovat #. Samotnou mapu velikosti $mxn$ vkládáme do textového souboru na $m$ řádcích o $n$ znacích. $l.$ symbol na $k.$ řádku reprezentuje políčko se souřadnicemi $[k, l]$ vložené mapy. Vložená mapa musí být obdelníkovitá. Znak $x$ symbolizuje překážku. Znak $P$ symbolizuje hráče stojícího na prázdném políčku. Znak $p$ symbolizuje hráče stojícího na speciálně označeném políčku. Znak $K$ symbolizuje krabici ležící na prázdném políčku a znak $k$ symbolizuje krabici ležící na speciálně označeném políčku. Prázdné políčko je symbolizováno znakem mezerou. Speciálně označené políčko, kam musíme přemístit krabice, je označené symbolem $O$. Vložená mapa $mxn$ musí mít maximální velikost $m = 25$ a $n = 25$. Žádný řádek nesmí být prázdný a žádný řádek nesmí obsahovat informace nesouvisející s mapou, jejich úvodem, resp. závěrem. Pokud mapa není obdélníkovitá, pak je potřeba mapu doplnit do obdélníku pomocí překážek.

## 5.2 Reprezentace výsutpních dat a jejich interpretace

Výstupní data přícházejí v příkazové řádce. Nejdříve přijde zpráva, která nás informuje, jestli program dokázal vyřešit daný problém do 150 posunů krabicemi. Pokud ano, program bude vypisovat krok po kroku tahy, kterými se dostaneme k cíli problému. K dalšímu kroku se dostaneme stisknutím klávesy *Enter*. Pokud nezvládne daný problém vyřešit do 150 posunů, pak napíše, že to vyřešit nezvládne a začne řešit další mapu. Příchozí krok je mapa, která obsahuje symboly $x$(překážka), $P$(hráč na volném políčku), $p$(hráč na označeném políčku), $K$(krabice na volném políčku), $k$(krabice na označeném políčku), „ “(volné políčko), $O$(speciálně označené políčko). Po vypsání všech map program napíšu hlášku upozorňující na fakt, že zrovna vypsa poslední mapu. Tím se vypne.

# 6. Závěr

Původně jsem chtěl udělat program pro všechny možné mapy. Tento by sice měl fungovat pro jakékoliv příchozí mapy, ovšem ty větší by trvaly velmi dlouhou dobu pro vyřešení. Aktuální program by se dalo ještě hodně vylepšovat a určovat víc situací a kroků, které už nemají smysl, tedy nevedou k cíli, ale základní myšlenku už splňuje a je schopen vyřešit základní sokobanovské úlohy. Je celá řada dalších možných algoritmů, které se pro tento problém dají použít. Většina z nich je založená na machine learning (například algoritmus Portfolio). Tyto možnosti jsem, i přes jejich zajímavost, zavrhl kvůli jejich komplikovanosti.