Theory of Computer Games (Fall 2019) Homework #1

National Taiwan University

Due: 14:20 (UTC+8), October 24, 2019

Homework Description

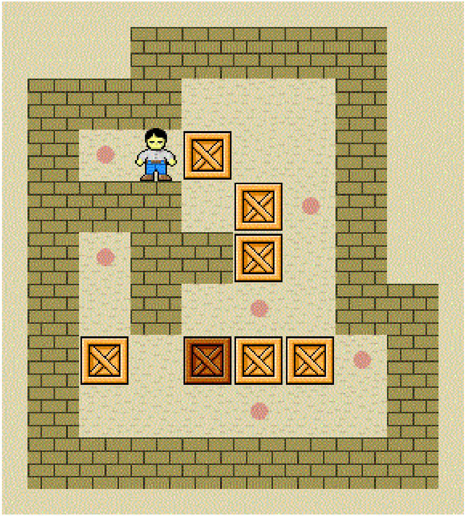
In this homework, you are asked to

1. Implement a solver of Pukoban.
2. Create a Pukoban puzzle.
3. Analyze the performance of different search algorithms.

Original Game - Sokoban

- A **Sokoban** ($\overset{\text{そう}}{\text{倉}}\overset{\text{こ}}{\text{庫}}\overset{\text{ばん}}{\text{番}}$) game is played on a board of squares, each of which is either a **floor** or a **wall**.
- Some of the floor squares contain **boxes**.
- Some of the floor squares are marked as **goal squares**.
- The number of boxes is equal to that of goal squares.
- The player is initially on a floor square that doesn't contain a box.

An Example



Sokoban Variation - Pukoban

- In this homework, you are going to solve a variation of Sokoban - **Pukoban**.
- In Sokoban, the player only allowed to **push** the box.
- In Pukoban, the player is also allowed to **pull**.

Rules of Pukoban

- The player can move either horizontally or vertically (namely, UP, DOWN, LEFT, RIGHT) to an adjacent square.
- **Push** action can be performed iff
  - There is a box on player's destination.
  - The box's *dest.* is not occupied. (by wall or block)

Rules of Pukoban (cont.)

- **Pull** action can be performed iff
  - These is a box on the opposite destination of player's *dest.*.
  - The player's *dest.* is not occupied. (by wall or block)
- In other words, you cannot perform **push** and **pull** at the same time.

Play Pukoban Yourself

- Under directory `pukoban`, use the command
  ```
  $ make
  ```
  to build the execution files, `pukoban` and `verifier`.
- Use
  ```
  $ ./pukoban -i filename [-o filename2] [-s n]
  ```
  to start the game from stage *n* in puzzle file `filename` and record the solution in file `filename2`.
- To begin with, execute
  ```
  $ ./pukoban -i ../testdata/tiny.in
  ```

Part I: Pukoban Solver

- Write a program to read puzzles from standard input and write solutions to standard output.
- We provide you 3 puzzle files under directory `testdata`, namely `tiny.in`, `small.in`, and `medium.in`.
    - The state space of is tiny in `tiny.in`.
    - The state space of is small in `small.in`.
    - The state space of is medium in `medium.in`.
    - There is a hidden testfile `large.in` reserved.
- Each puzzle file contains several puzzles. Your program should read until the EOF.
- You can use at most 2 threads.
- The time limit of each puzzle file is 60 seconds.
- The memory limit of each puzzle file is 4 GB.

Puzzle File (Input) Format

- The first line of each puzzle contains two positive integers, n and m, separated by a space.
    - $1 \leq n, m \leq 15$
    - $nm \leq 50$
- The following n lines describe the initial board. Each line is a string composed of `#`, `@`, `+`, `$`, `*`, `.`, `-` of length m.
- There is at least 1 `$` square.

Puzzle File (Input) Format (cont.)

Legend:

- #: a wall square
- @: the player on a non-goal square
- +: the player on a goal square
- $: a box on a non-goal square
- *: a box on a goal square
- .: a goal square
- -: a non-goal square

Solution File (Output) Format

- For each puzzle, the solution contains 2 lines.
- The first line is a nonnegative number k. The second line is a string composed of `u`, `d`, `l`, `r`, `U`, `D`, `L`, `R`, `^`, `˘`, <, > of length k.
  - `u` and `U` and `^`: UP
  - `d` and `D` and `˘`: DOWN
  - `l` and `L` and <: LEFT
  - `r` and `R` and >: RIGHT
- Uppercase indicates **push** actions, while arrow indicates **pull** actions.
- There sould be no infeasible action in your solver's output.
- Under directory `testdata`, you can find `tiny.out` solving `tiny.in`.

Verifier

Under directory `pukoban`, execution file `verifier` checks the format of puzzle/solution files.

- `$ ./verifier -i filename`
  check if `filename` is a valid puzzle file.
- `$ ./verifier -o filename`
  check if `filename` is a valid solution file.
- `$ ./verifier -i filename1 -o filename2`
  if both `filename1` and `filename2` are valid, check if `filename2` solves `filename1`.

Part II: Puzzle Creation

- Give one valid Pukoban puzzle in `[your_id].in` (e.g., `b08902000.in`) and a corresponding solution in `[your_id].out` (e.g., `b08902000.out`).
- Your puzzle file and solution file should be validated by `verifier`.
- You should analyze the complexity of your puzzle.

Part III: Algorithm Analysis

Your report should include but not limited to

- Implementation
  - How to compile and run your code under linux. (If TA has difficulty compiling your code, he may ask you to demonstrate the process.)
  - What algorithm and heuristic you implement.
- Experiment
  - The comparison between different algorithms.
  - The execution times are required.
- Discussion
  - The complexity of a Pukoban puzzle.
  - The complexity of each algorithm.
  - The complexity of your created puzzle.

Submission

- Directory hierarchy:
    - `your_id` // e.g. b08902000 (lowercase)
        - makefile // make your code
        - `src` // a folder contains all your codes
        - `your_id.in` // your puzzle
        - `your_id.out` // your solution
        - `report.pdf` // your report
- Compress your folder into a `zip` file and submit to `https://www.csie.ntu.edu.tw/~tcg/2019/hw1.php`.
- Due to the server limitation, the file size is restricted to 2 MB.
- If your program has a pattern database greater than 2 MB in size, you can simply upload the code that generates the pattern database. The database should be generated within 30 minutes.

Grading Policy

There are 15 points in total, composed of 3 parts.

1. Pukoban solver (8 points)
    - Besides the three puzzle files in directory `testdata`, your solver is required to pass a private puzzle file, `large.in`.
    - The point distribution goes as 1, 2, 2, 3 for `tiny.in`, `small.in`, `medium.in`, `large.in`, respectively.
    - If your solver fails to solve a puzzle file (every stage) correctly within the time limit, you won't get any point.
    - Suppose your solver gives an N-move solution to a single puzzle, and the optimal move number is N_0, you'll get $0.1 + 0.001\lfloor \frac{100N_0}{N} \rfloor$ point. (10 puzzles per puzzle file)

Grading Policy (cont.)

1. Pukoban solver (8 points) (cont.)
   - If you solve `medium.in` within 1 second, an extra 1 point is appointed.
   - If you solve `large.in` within 1 second, another extra 1 point is appointed.
   - You at most can get 8 points in this part.
2. Puzzle creation (2 points)
   - Your puzzle file and solution file should pass `verifier` to get the 2 points.
   - If your puzzle is considered complex enough, you'll get an extra bonus.
3. Report (5 points)
   - Your score will be evaluated with TA's HNN (human neural network) model.

References

- Sokoban - Wikipedia
  `https://en.wikipedia.org/wiki/Sokoban`
- Pukoban Online Game
  `http://puzzles.net23.net/pukoban.htm`
- Contant TA
  - `theory.of.computer.games.2019@gmail.com`
  - `Title: [HW1] your questions`