Theory of Computer Games (Fall 2018) Homework #1

National Taiwan University

Due Date: 14:20 (UTC+8), October 25, 2018

Homework Description

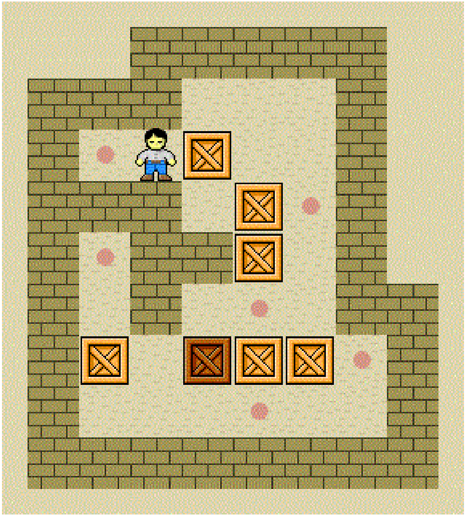
In this homework, you are asked to

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

Rules of Sokoban

- A **Sokoban** (倉庫番) 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



Rules of Sokoban (cont.)

- The player can move either horizontally or vertically (namely, UP, DOWN, LEFT, RIGHT) to an adjacent square only if
  - The adjacent square is a floor that doesn't contain a box, or
  - The adjacent square is a box, and the square beyond that box is a floor that doesn't contain another box. In this case the box is pushed after the move.
- A solution to a Sokoban puzzle is a sequence of moves that makes all boxes on goal squares eventually.

Play Sokoban Yourself

- Under directory `sokoban`, use the command
  ```
  $ make
  ```
  to build the execution files, `game` and `verifier`.
- Use
  ```
  $ ./game -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
  ```
  $ ./game -i ../testdata/small.in
  ```

Part I: Sokoban Solver

- Write a program to read puzzles from standard input and write solutions to standard output. An example code can be found under directory `b07902000`.
- We provide you 3 puzzle files under directory `testdata`, namely `small.in`, `large.in`, and `large2.in`.
  - Each puzzle in `small.in` contains no more than 4 boxes.
  - Each puzzle in `large.in` and `large2.in` contains at least 5 boxes.
- Each puzzle file contains several puzzles. Your program should read until the EOF.
- The time limit of each puzzle file is 60 seconds.

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`: UP
  - `d` and `D`: DOWN
  - `l` and `L`: LEFT
  - `r` and `R`: RIGHT
- By convention, one uses uppercase to indicate a box being pushed. Nevertheless, in this homework we neglect the letter cases of a solution.
- Under directory `testdata`, you can find `small.out` solving `small.in`.

verifier

Under directory `sokoban`, 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 Sokoban puzzle in `[your_id].in` (e.g., `b07902000.in`) and a corresponding solution in `[your_id].out` (e.g., `b07902000.out`).
- Your puzzle file and solution file should be validated by `verifier`.

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 Sokoban puzzle.
    - The complexity of each algorithm.

Submission

- Directory hierarchy:
  - `your_id` // e.g. b07902000
    - `source` // 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/2018`.
- 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.

Grading Policy

There are 15 points in total.

1. Sokoban solver (8 points)
   - Besides the three puzzle files in directory `testdata`, your solver is required to pass a private puzzle file, `small2.in`.
     - Puzzle file `small2.in` contains no more than 10 puzzles, and each puzzle contains no more than 4 boxes.
   - Each puzzle file counts for 2 points. If your solver fails to solve a puzzle file correctly within the time limit, you won't get any point.
   - The less moves your solver gives, the more points you'll get.
   - If your solver is super fast on `large.in` or `large2.in`, you'll get an extra bonus.
2. Puzzle creation (2 points)
   - The more complex your puzzle is, the more points you'll get.
3. Report (5 points)

References

- Sokoban - Wikipedia
  `https://en.wikipedia.org/wiki/Sokoban`
- Sokoban Online Game
  `http://www.game-sokoban.com`