Assignment 3

General Info

You must read fully and carefully the assignment specification and instructions.

- **Course:** COMP20003 Algorithms and Data Structures @ Semester 2, 2021
- **Deadline Submission:** Friday 22nd October 2021 @ 5 pm (end of Week 12)
- **Course Weight:** 15%
- **Assignment type:** individual
- **ILOs covered:** 2,4
- **Submission method:** via ED

Purpose

The purpose of this assignment is for you to:

- Increase your proficiency in C programming, your dexterity with dynamic memory allocation and your understanding of data structures, through programming a search algorithm over Graphs.
- Gain experience with applications of graphs and graph algorithms to solving combinatorial games, one form of artificial intelligence.

Walkthrough

Sokoban

In this programming assignment, you'll be expected to build an AI algorithm to solve Sokoban. The game invented in 1980 (program released in 1982) is one of the classics among arcade puzzle games. You can play the game compiling the code given to you using the keyboard, or using this web implementation with tutorials (alternative without tutorials).

The code in this assignment was adapted from the open-source terminal version using ncurses made available by CorrentinB.

Game Rules
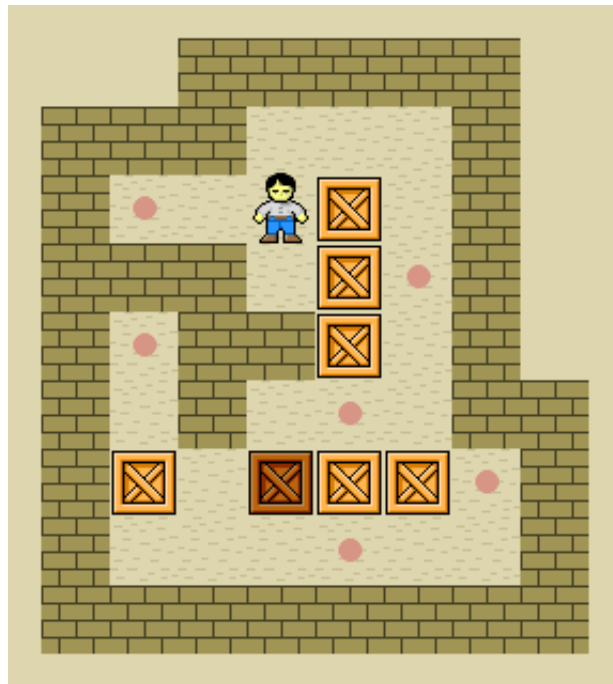
As explained in the wikipedia entry, The game is played on a board of squares, where each square is a floor or a wall. Some floor squares contain boxes, and some floor squares are marked as storage locations.

The player is confined to the board and may **move horizontally or vertically** onto empty squares (never through walls or boxes). The player can **move a box** by walking up to it and **pushing** it to the square beyond. Boxes cannot be pulled, and they cannot be pushed to squares with walls or other boxes.

The **number of boxes equals the number of storage locations**.

The puzzle is solved when **all boxes are placed at storage locations**.

For the courious reader

The Science of Sokoban

Sokoban can be studied using the theory of computational complexity. The problem of solving *Sokoban* puzzles was first proved to be NP-hard [4][5]. Further work showed that it was significantly more difficult than NP problems; it is PSPACE-complete [6]. This is of interest for artificial intelligence (AI) research because solving *Sokoban* can be compared to the automated planning required by some autonomous robots.

NP-complete problems are hard to solve. The best algorithms to solve NP-Complete problems run in exponential time as a function of the size of the problem and the shortest accepted solution. Hence, be aware that your AI solver will struggle more as the number of boxes increases. We talked in the lectures about the Travelling Salesman Problem as one example of an NP-Complete problem. In fact, many real-world problems fall under this category. We are going to learn more about NP-Complete problems in the last lecture of the course via an invited talk in week 12.

Sokoban is difficult not only because of its large branching factor, but also because of its large search tree depth. Some level types can even be extended indefinitely, with each iteration requiring an exponentially growing number of moves and pushes.[7] Skilled human players rely mostly on heuristics and are usually able to quickly discard a great many futile or redundant lines of play by recognizing patterns and subgoals, thereby drastically reducing the amount of search.

Some *Sokoban* puzzles can be solved automatically by using a single-agent search algorithm, such as IDA*; enhanced by several techniques that make use of domain-specific knowledge.[8] This is the method used by *Rolling Stone*,[9] a *Sokoban* solver developed by the University of Alberta GAMES Group. *Festival*[10] was the first automatic solver to solve all 90 levels in the standard benchmark test suite. However, the more complex *Sokoban* levels are out of reach even for the best automated solvers.[11]

Above and Beyond

Below I added interesting material for those that want to take a deeper dive on Sokoban Solvers.

- Official benchmarks and results and best Sokoban solvers from 1980-2021. The best solver, Festival, was introduced at a conference last year, in 2020. A collaboration by a Google employee and a great professor Jonathan Schaeffer who made outstanding contributions to AI

and games.

- Relevant publications, A great starting point in that list is to read Andreas Junghanns Ph.D. dissertation about Sokoban. 1999. Pushing the Limits: New Developments in Single-Agent Search. *Ph.D. Dissertation, University of Alberta, 1999.*
- Everything about Sokoban: WIKI
- The juice of the great computational ideas & insights behind the best Sokoban solvers.

The Algorithm

A configuration of the Sokoban game is specified by the location of walls, boxes, storage areas and player. A configuration is called a *state*. The Sokoban Graph $G = \langle V, E \rangle$ is implicitly defined. The vertex set V is defined as all the possible configurations (states), and the edges E connecting two vertexes are defined by the legal movements **(right, left, up, down).** All edges have a weight of 1.

Your task is to find the path traversing the Sokoban Graph from the initial state (vertex) leading to a state (vertex) where all the boxes are located on a storage area. The best path is the shortest path. A path is a sequence of movements. You are going to use Dijkstra to find the shortest path first, along with some game-specific optimizations to speed up your algorithm.

When the AI solver is called (Algorithm 1), it should explore all possible paths (sequence of move actions) following a Dijktsra strategy, until a path solving the game is found. Note that we use transposition tables to avoid duplicate states in the search. If a state was already expanded (popped from the priority queue), we will not include it again in the priority queue (line 23 in Algorithm 1). We will also ignore states where the player doesn't move as a result of moving towards an adjacent wall, or a box which cannot move (line 18). We will finally avoid states where boxes are located in a corner (line 18, see file `utils.h`). The algorithm should return the **best solution found**. This path will then be executed by the game engine if the option `play_solution` was used as an argument.

Algorithm 1 AI Sokoban Algorithm

```
 1: procedure FINDSOLUTION(start, showSolution)
 2:     n ← CREATEINITNODE(start)
 3:     PUSHPRIORITYQUEUE(n)
 4:     exploredTable ← create empty array
 5:     hashTable ← initialize Hash Table for duplicate detection
 6:     while priorityQueue ≠ empty do
 7:         n ← PRIORITYQUEUE.POP()
 8:         exploredNodes ← exploredNodes + 1
 9:         exploredTable ← record n in array
10:         if WINNINGCONDITION(n) then                          ▷ Found a solution
11:             solution ← SAVESOLUTION(n)
12:             SolutionSize ← n.depth
13:             break
14:         end if
15:         for each move action a ∈ {Left, Right, Up, Down} do
16:             playerMoved ← APPLYACTION(n, newNode, a)          ▷ Create Child newNode
17:             generatedNodes ← generatedNodes + 1
18:             if playerMoved is false or SIMPLECORNERDEADLOCK(newNode) then
19:                 FREE(newNode)
20:                 continue
21:             end if
22:             flatMap ← FLATTENMAP(newNode)                     ▷ converts the map into a 1D map
23:             if flatMap is a duplicate then                    ▷ Check duplicates in HashTable
24:                 duplicatedNodes ← duplicatedNodes + 1
25:                 FREE(newNode)
26:                 continue
27:             end if
28:         end for
29:         hashTable ← INSERTHASHTABLE(flatMap)
30:         PRIORITYQUEUE.PUSH(newNode)
31:     end while
32: end procedure
```

You might have multiple paths leading to a solution. **Your algorithm should consider the possible actions in the following order**: left, right, up or down.

Make sure you manage the memory well. When you finish running the algorithm, you have to free all the nodes from the memory, otherwise you will have memory leaks. You will notice that the algorithm can run out of memory fairly fast after expanding millions nodes.

The `applyAction` creates a **new node**, that

- points to the parent,
- updates the state with the action chosen,
- updates the depth of the node,

- updates the priority (used by the priority queue) of the node to be the negative node's depth d (if the node is the dth step of the path, then its priority is -d). This ensures the expansion of the shortest paths first, as the priority queue provided is a max heap,
- updates the action used to create the node.

Check the file `utils.h, hash_table.h, priority_queue.h` where you'll find many of the functions in the algorithm already implemented. Other useful functions are located directly in the file `ai.c`, which is the only file you need to edit to write your algorithm inside the function `findSolution`. Look for the comment `FILL IN THE GRAPH ALGORITHM`. All the files are in the folder `src/ai/.`

Deliverables

Deliverable 1 - Dijkstra **Solver** source code

You are expected to hand in the source code for your solver, written in C. Obviously, your source code is expected to compile and execute flawlessly using the following makefile command: `make` generating an executable called `sokoban`. Remember to compile using the optimization flag `gcc -O3` for doing your experiments, it will run twice as quickly as compiling with the debugging flag `gcc -g` (see `Makefile`, and change the `CC` variable accordingly). For the submission, please **submit your makefile with `gcc -g` option**, as our scripts need this flag for testing. Your program must not be compiled under any flags that prevents it from working under gdb or valgrind.

Your implementation should work well over the first 3 layouts, but it will not be expected to find a solution to any layout, as it may exceed the available RAM in your computer before finding a solution. Feel free to explore maps given in the folder `maps_suites`. They are taken from the official benchmarks of sokoban. All you have to do is to copy and paste a single map into a new file, and then call it with your solver.

Deadlock Detection (optimizations)

The simplest way to improve the code is by improving the deadlock detection (SimpleCornerDeadlock, line 18, Algorithm 1, file `utils.c`). Implement at least 1 deadlock detection in the link above.

Take a look at these 2 links for further optimization ideas & insights.

If you do any optimizations & change of Data Structures used & deadlock detection improvement, please make sure to explain concisely **what** it is that you implemented, **why** you chose that optimization, **how** it affects the performance (show number of expanded nodes before and after the optimization for a set of test maps), and **where** the code of the optimizations is located. This explanation should be included in the file located at the root of the basecode: `Report.md`. Please make sure that your solver still returns the optimal solution.

Deliverable 2 - Submission Certification Form

Once you submit your code, please also fill out the form, no later than 24h after the deadline.

> ⚠️ The form is mandatory, and shouldn't take more than 5 minutes.

Rubric

Assignment marks will be divided into three different components.

1. Solver (12)
2. Code Style (2)
3. Optimizations (1)

Please note that you should be ready to answer any question we might have on the details of your assignment solution by e-mail, or even attending a brief interview with me, in order to clarify any doubts we might have.

Maps & Solution formats

Puzzle File Format

We adapted the sokoban file reader to support the following specification, but we don't support comments specified by `%`.

#	Wall
	Space
.	Goal
@	Sokoban
$	Box
+	Sokoban on Goal
*	Box on Goal
%	Comment Line

Solution File Format

solutions returned by the solver follow this format

l	Move Left
r	Move Right
u	Move Up
d	Move Down
L	Push Left
R	Push Right
U	Push Up
D	Push Down

You can load your map and solution into the JS Visualiser.

JS Visualiser

For the visualiser to work, puzzles must be rectangular. If your map is not rectangular, just filled it in with walls instead of empty spaces.

The Code Base

You are given a base code. You can compile the code and play with the keyboard (arrows). You are going to have to program your solver in the file `ai.c`. Look at the file `main.c` (main function) to know which function is called to call the AI algorithm.

You are given the structure of a node, the state, a max-heap (priority queue) and a hashtable implementation to check for duplicate states efficiently (line 23 in the Algorithm 1). Look into the `utils.*` files to know about the functions you can call to apply an action to update a game state. All relevant files are located in the folder `src/ai/`.

In your final submission, you are free to change any file, but make sure the command line options remain the same.

Input

You can play the game with the keyboard by executing

```
./sokoban <map>
```

where `map` points to the file containing the sokoban problem to solve.

In order to execute your AI solver use the following command:

```
./solver -s <map> play_solution
```

Where `-s` calls your algorithm. `play_solution` is optional, if typed in as an argument, the program will play the solution found by your algorithm once it finishes. All the options can be found if you use option `-h`:

```
$./sokoban -h
USAGE
        ./sokoban <-s> map <play_solution>

DESCRIPTION
 Arguments within <> are optional
    -s                 calls the AI solver
    play_solution      animates the solution found by the AI solver
```


for example:

```
./sokoban -s test_maps/test_map2 play_solution
```

Will run the 2nd map expanding and will play the solution found.

Output

Your solver will print into an `solution.txt` file the following information:

1. Solution
2. Number of expanded nodes.
3. Number of generated nodes.
4. Number of duplicated nodes.
5. Solutions length
6. Number of nodes expanded per second.
7. Total search time, in seconds.

For example, the output of your solver `./sokoban -s test_maps/test_map2` could be:

```
SOLUTION:
rrrrrrdrdLLLLLLLLLullluRRRRRRRururRRRRRRRRRR

STATS:
        Expanded nodes: 978745
        Generated nodes: 3914976
        Duplicated nodes: 2288345
        Solution Length: 43
        Expanded/seconds: 244506
        Time (seconds): 4.002942
```

Expanded/Second is computed by dividing the total number of expanded nodes by the time it took to solve the game. A node is expanded if it was popped out from the priority queue, and a node is generated if it was created using the `applyAction` function. This code is already provided.

Code Submision - Deliverable

This code slide does not have a description.

Submission Certification

In this form you will confirm & certify your submission against the COMP20003 Honor Code and academic integrity of your submission.

COMP20003 Honor Code

We're committed to the integrity of your work on the COMP20003 course. To do so, every learner-student should:

- Submit their own original work
- Do not share code (solution) with other students

Question 1

Did you enjoy the assignment project?

(This question is for us to understand the usefulness of this project assignment for the future)

- ◯ Yes, a lot!

- ◯ Yes

- ◯ More or less

- ◯ Not that much

- ◯ No, I didn't

- ◯ I don't want to participate in this question

Question 2

Do you feel you learn with this assignment project?

(This question is for us to understand the usefulness of this project assignment for the future)

○ Yes, I learnt quite a lot!

○ Yes, I learnt some things

○ I learnt some, but not much

○ I learnt almost nothing

○ I don't want to participate in this question

Question 3

Feel free to share any feedback here

We are interested in any constructive negative or positive feedback, both are helpful to reflect what may need to change and what is working. Good humor and politeness are always welcome! ;-) Thanks!

No response

Question 4

I certify that this was all our original work. If we took any parts from elsewhere, then they were non-essential parts of the project, and they are clearly attributed at the top of the file and in a separate report. I will show I agree to this honor code by typing "Yes":

This declaration is the same as the one in Khan Academy. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us. You are always better off getting a very bad mark (even zero) than risking to go that path, as the consequences are serious for students.

⚠️ The project will not be marked unless this question is answered correctly and exactly with "Yes" as required.

No response

Programming Style

Below is a style guide which assignments are evaluated against. For this subject, the 80 character limit is a guideline rather than a rule - if your code exceeds this limit, you should consider whether your code would be more readable if you instead rearranged it.

```c
 1 /** ************************
 2  * C Programming Style for Engineering Computation
 3  * Created by Aidan Nagorcka-Smith (aidann@student.unimelb.edu.au) 13/03
 4  * Definitions and includes
 5  * Definitions are in UPPER_CASE
 6  * Includes go before definitions
 7  * Space between includes, definitions and the main function.
 8  * Use definitions for any constants in your program, do not just write
 9  * in.
10  *
11  * Tabs may be set to 4-spaces or 8-spaces, depending on your editor. Th
12  * Below is ``gnu'' style. If your editor has ``bsd'' it will follow the
13  * style. Both are very standard.
14 */
```

Some automatic evaluations of your code style may be performed where they are reliable. As determining whether these style-related issues are occurring sometimes involves non-trivial (and sometimes even undecidable) calculations, a simpler and more error-prone (but highly successful) solution is used. You may need to add a comment to identify these cases, so check any failing test outputs for instructions on how to resolve incorrectly flagged issues.

Plagiarism

This is an individual assignment. The work must be your own.

While you may discuss your program development, coding problems and experimentation with your classmates, you must not share files, as this is considered plagiarism.

If you refer to published work in the discussion of your experiments, be sure to include a citation to the publication or the web link.

"Borrowing" of someone else's code without acknowledgment is plagiarism. Plagiarism is considered a serious offense at the University of Melbourne. You should read the University code on Academic integrity and details on plagiarism. Make sure you are not plagiarizing, intentionally or unintentionally.

You are also advised that there will be a C programming component (on paper, not on a computer) in the final examination. Students who do not program their own assignments will be at a disadvantage for this part of the examination.

Late Policy

The late penalty is 10% of the available marks for that project for each day (or part thereof) overdue. Requests for extensions on medical grounds will need to be supported by a medical certificate. Any request received less than 48 hours before the assessment date (or after the date!) will generally not be accepted except in the most extreme circumstances. In general, extensions will not be granted if the interruption covers less than 10% of the project duration. Remember that departmental servers are often heavily loaded near project deadlines, and unexpected outages can occur; these will not be considered as grounds for an extension.

Students who experience difficulties due to personal circumstances are encouraged to make use of the appropriate University student support services, and to contact the lecturer, at the earliest opportunity.

Finally, we are here to help! There is information about getting help in this subject on the LMS. Frequently asked questions about the project will be answered on Ed.

Additional Support

Your tutors will be available to help with your assignment during the scheduled workshop times. Questions related to the assignment may be posted on the Ed discussion forum, using the folder tag Assignments for new posts. You should feel free to answer other students' questions if you are confident of your skills.

A tutor will check the discussion forum regularly, and answer some questions, but be aware that for some questions you will just need to use your judgment and document your thinking. For example, a question like, "How much data should I use for the experiments?", will not be answered; you must try out different data and see what makes sense.

If you have questions about your code specifically which you feel would reveal too much of the assignment, feel free to post a private question on the discussion forum.

Have fun!