ITP 20002 Discrete Mathematics, Fall 2020

Homework 2. Set Representations and SAT-based Puzzle Solving

Shin Hong

1. Overview

This homework asks you to implements two versions of integer set data structures, one with array (Section 2.1) and the other with bitvector (Section 2.2), and construct a puzzle solver for the Snake game (Section 3) using set data structure and SAT solver Z3[1].

Set data structures are designed to contain a set of integer values. Array list-based data structure represents a membership of an integer by containing it in an integer array, thus, set operations perform searches, addition, and removal of integer values on the integer array. Bitvector-based data structure marks the membership of its elements on a Boolean vector (calling it bitvector). Set operations for bitvector-based data structure involve bitwise operations on the internal bitvector.

Snake is a grid-based puzzle where a player is asked to connect the top-left corner to the bottom-right corner by marking some of grid cells while satisfying given puzzle constraints. You must design a scheme to transform a given puzzle instance into a corresponding propositional logic formula and construct a puzzle solving program in C using your own set data structure implementation and SAT solver Z3.

The submission deadline is **11:00 PM, 14 October** (Wed). Your submission must include the implementations of the two versions of set data structures, a Snake puzzle solving program and a report on all tasks of Homework 2.

2. Set data structure

You can find the baseline code from the `homework2` branch at `https://github.com/hongshin/DiscreteMath`. Directory `array` contains the code for array list-based data structure, and `bitvect` for bitvector-based data structure.

2.1. Array-based data structure

An integer set can be represented with an array where an integer is contained if and only if the integer is the member of the set. It is possible to determine whether or not an integer is contained in a set by checking if the integer exists in the array. Note that, although members are arranged in a certain order, the ordering does not represent any information of the set.

You can find the structural definition of an array-based set representation `array_set` in `array/set.h`. The `n_size` field is to represent the current number of members. The `elems` field is to point an integer array whose length is `n_size`. The interface of the set operations is declared also at `array/set.h`. The definitions of the operations to construct and deconstruct `array_set` objects are given at `array/set.c`.

2.2. Bitvector-based data structure

For a finite universe with total n distinct values, a set can be represented as a vector of n bits. Each bit is mapped uniquely with a specific value, such that the truth value of a bit indicates whether a corresponding value is contained in a set.

The structure of the bitvector-based set representation is defined as `bitvect_set` in `bitvect/set.h`. The `univ` field is to point an integer array that contains all values of integers, and the n_univ field indicates the size of the univ integer. A bitvector-based set object must be constructed with the given values of `univ` and `n_univ`. You can assume that no duplication exists in `univ`, and the values in `univ` never chance once it is used for constructing set objects. The `bitvector` field points to an array to represent the bitvector where the *n*-th bit is to mark whether or not `univ[n]` is contained in the set. The `n_size` field represents the number of members contained in the set.

The interface of `intset` is declared at `bitvect/set.h`. Note that the types of some operations are difference from the array-based version. The definitions of object construction and deconstruction operations are given at `bitvect/set.c`.

2.3. Set operations

Both versions of the set data structure must provide the following operations:

- $size(S)$: returns the number of distinct elements contained in the given set S (i.e., cardinality).
- $add(S, v)$: update the given set S by adding the given integer v (if v was not in S).
- $remove(S, v)$: update the given set S by removing the given integer v (if v was in S)
- $contains(S, v)$: determine whether the given integer v is contained in the given set S, or not.
- $equals(S_1, S_2)$: determine whether the given two sets S_1 and S_2 are equivalent.
- $union(S_1, S_2)$: create a new set that is the union of the given two sets S_1 and S_2.
- $intersection(S_1, S_2)$: create a new set that is the union of the given two sets S_1 and S_2.
- $difference(S_1, S_2)$: create a new set that is the union of the given two sets S_1 and S_2.
- $powerset(S)$: create an array of all subsets (i.e., power set) of the given set S.

The details of operation requirements are given as comment at `array/set.c` and `bitvet/set.c`. These operations must be implemented to satisfy all given requirements, and as adhere to the existing definitions of the structure and the operations. Also, you can find some samples of operation use cases at the test case code (`test.c`).

[1] The latest version of Z3 can be found at https://github.com/Z3Prover/z3

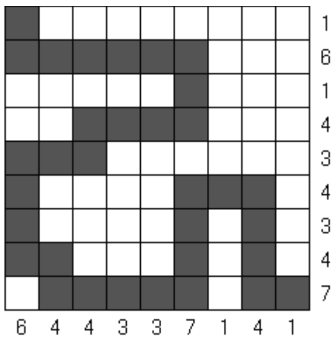

Figure 1. Snake Puzzle Example

3. Snake puzzle

A Snake puzzle[2] consists of a $n \times n$ square grid and annotations given to the rows and the columns of the square grid. Starting from the top-left corner, a player is asked to make a move to one unvisited cell among the four adjacent ones at a time, until the player reaches to the bottom-left corner. In addition, the player's moves must satisfy all constraints of annotations that the number of visited cells in a column/row must be the same as the integer annotated to the corresponding column/row. Figure 1 shows an example of Snake where black cells denote all visited cells.

As similar for Homework 1, you are asked to model a puzzle instance as a propositional logic formula such that an assignment that satisfies the formula represents a solution of the puzzle instance (the formula should be unsatisfiable if a puzzle instance has no solution). Based on this modelling, you must construct a program that uses a SAT solver Z3 to find a solution of a puzzle instance given as input. The description is not repeated, but you are asked to take the same as given in Section 3 of Homework 1.

4. Requirements

4.1. Set data structure implementation

You are asked to complete the missing parts of the two set data structure implementations. You are restricted to add code to where the `/*TO-DO*/` comment exists. Note that You are not allowed to change for all the other parts.

You can build each version by `make` (see `Makefile` in each directory). Your program must be compatible with GCC 5.4.0 or a higher version. As your program will be tested on the peace server[3], it is recommended to check if your program is working well on it.

You can run tests on your program by `make test`. This command prints "`Pass`" when your implementation passes all test cases given in `test.c`, or you will see that an assertion violation happens when your program unexpectedly behaves. Note that your program will be tested with more test cases at evaluation.

4.2. Snake puzzle

An input consists of three lines. The first line contains one positive integer n, the length of one side of the puzzle grid. n does not exceed 15. The second line contains n number of non-negative integers between 0 and n separated by one or multiple white spaces. These numbers represent row-wise annotations: the first number is for the top row, and the last for the bottom row. Similarly, the third line of an input contains n number representing column-wise annotations. The first number is for the left-most column and the last number is for the right-most column. Each number is a non-negative integer between 0 and n, and the n numbers are separated by one or multiple white spaces. The following is the input text for Figure 1:

```
9
1 6 1 4 3 4 3 4 7
6 4 4 3 3 7 1 4 1
```

You can assume that a well-formed input is given to your program.

Your puzzle-solving program must use one of your set data structure implementations. Explain in your report for what purpose a set data structure is used for constructing a Snake puzzle solver.

4.3. Report

You must write one report that covers all aspects of Homework 2 (including two set implementations, and Snake). You must use a given report template and your report must not exceed 2 pages. All contents in reports must be written in English (none of your Korean sentences will be counted in evaluation). Your report must be converted to PDF for submission.

You must detail in your report the following aspects of your solution:

(1) your idea and details on accomplishing each design and implement task
(2) structure of implementations
(3) validation results
(4) open discussion, for instances, lessons learned, observations, further investigation, suggestions of possible extension or improvements, challenges, limitations of your submission, etc.

In addition, as a part of discussion, discuss in which cases an array-based or a bitvector-based data structure has more benefits over the other version.

Evaluation will be primary based on your report (not your implementation), thus, you must try best to deliver all your results with best presentations in the reports.

5. Submission

Submit an archive of all your results (e.g., tar or zip file) by 11:00 PM, 14 October (Wed) via Hisnet. Your submission must include (1) reports in PDF, (2) source code files of two versions of set data structure, and (3) a C source code file of a Snake puzzle solver. The submission deadline is strict, and no late submission will be accepted.

[2] This example is taken from http://cross-plus-a.com

[3] peace.handong.edu