MINI PROJECT DESCRIPTION

1. Balanced Course Assignment

a. Problem description

At the beginning of the semester, the head of a computer science department *D* have to assign courses to teachers in a balanced way. The department *D* has m teachers *T*={1, 2, ..., *m*} and *n* courses *C*={1, 2,..., *n*}. Each teacher $t \in T$ has a preference list which is a list of courses he/she can teach depending on his/her specialization. We know a list of pairs of conflicting two courses that cannot be assigned to the same teacher as these courses have been already scheduled in the same slot of the timetable. The load of a teacher is the number of courses assigned to her/him. How to assign *n* courses to *m* teacher such that each course assigned to a teacher is in his/her preference list, no two conflicting courses are assigned to the same teacher, and the difference between maximal load and minimal load is minimal.

InputFile The input consists of following lines

- Line 1: contains two integers *m* and *n* ($1 \leq m \leq 10$, $1 \leq n \leq 30$)
- Line *i*+1: contains an positive integer *k* and *k* positive integers indicating the courses that teacher *i* can teach ($\forall$ *i* = 1,. . . , *m*)
- Line *m*+2: contains an integer *k*
- Line *i*+*m*+2: contains two integer *i* and *j* indicating two conflicting courses ($\forall$ *i*=1,. . . , *k*)

OutputFile

The output contains a unique number which is the maximal load of the teachers in the solution found and the value -1 if not solution found.

Example:

Input	Ouput
4 9 **4 1 2 3 5** **5 1 2 5 6 8** **6 1 2 4 5 7 9** **4 3 4 5 9** **13** **1 3** **1 2** **2 5** **1 8** **1 4** **4 9**	**1**

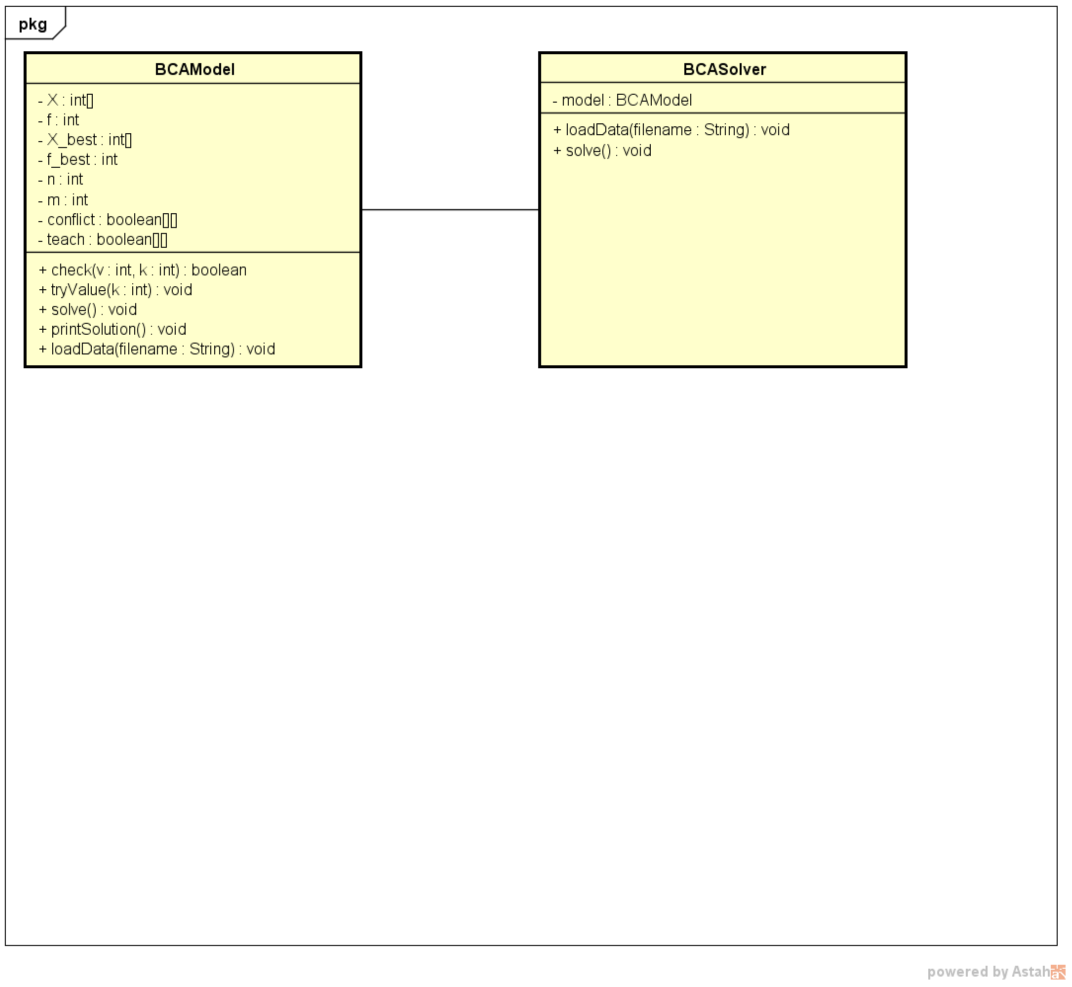
5 7 **2 3** **1 7** **6 9** **4 8** **3 5** **2 7**	

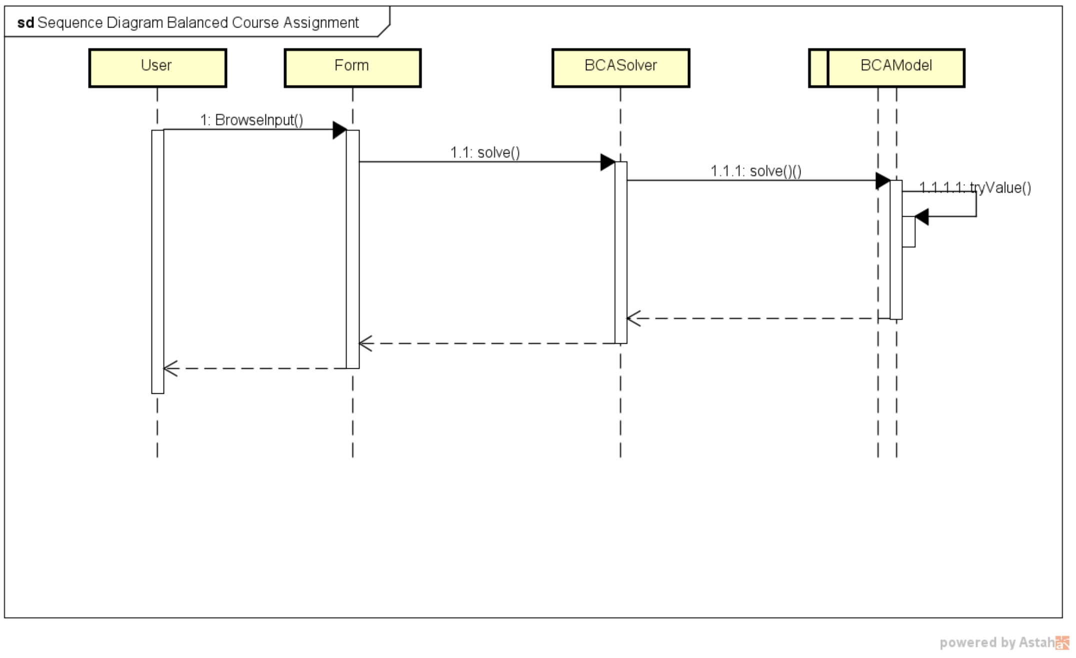
Courses assigned to teacher 1: 1, 5
Courses assigned to teacher 2: 2, 6, 8
Courses assigned to teacher 3: 4, 7
Courses assigned to teacher 1: 3, 9

Maximal load = 3, minimal load = 2

b. Design

- Use backtracking search to explore all possibilities in the search space.
- Decision variables X[] in which X[i] represents the index of teacher assigned to the course i.
- Method tryValue(int k) tries all values for X[k]
- Method check(int v, int k) returns true if value v can be assigned to X[k] without violating constraints of the problem
- Data structures
    - conflict[i][j] = true if courses i and j conflict to each other
    - teach[t][i] = true if teacher t can teach course i





2. Sudoku

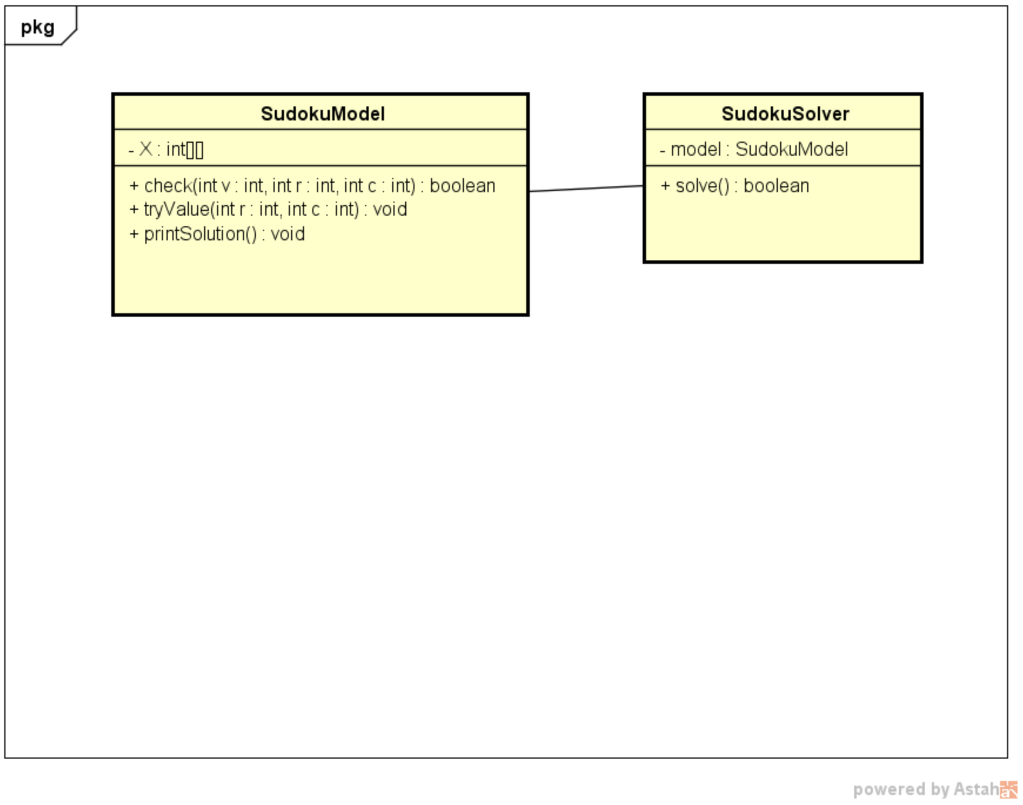
a. Problem description

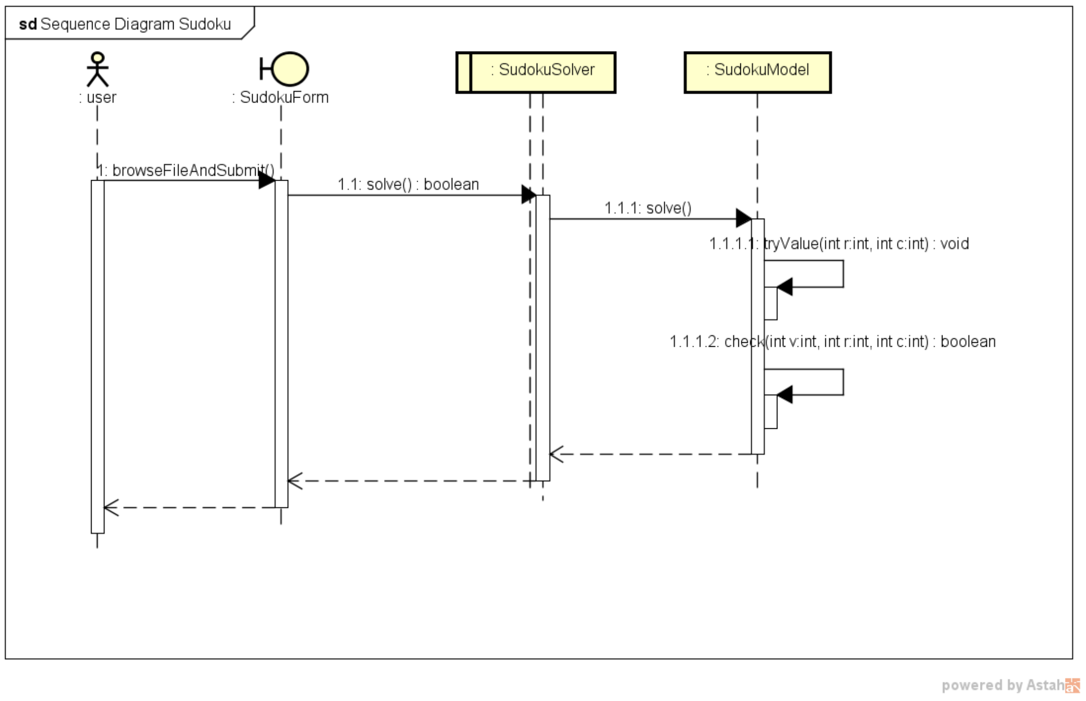
There is an incomplete 9x9 table of numbers. Fill each empty cell a number from 1 to 9 satisfying following constraints

- Numbers within any of the 9 individual 3x3 boxes are different
- Numbers within any row of the given 9x9 table are different
- Numbers within any column of the given 9x9 table are different

b. **Design**

- Use backtracking search to explore all possibilities in the search space.
- Decision variables X[][] in which X[i][j] represents the value assigned to the cell row i and column j of the table.
- method tryValue(int i, int j) tries all values for cell (i, j) of the table
- method check(int v, int i, int j) returns **true** if v can be assigned to cell (i, j) without violating constraints of the problem.




sd Sequence Diagram Sudoku
: user
: SudokuForm
: SudokuSolver
: SudokuModel
1: browseFileAndSubmit()
1.1: solve() : boolean
1.1.1: solve()
1.1.1.1: tryValue(int r:int, int c:int) : void
1.1.1.2: check(int v:int, int r:int, int c:int) : boolean
powered by Astah

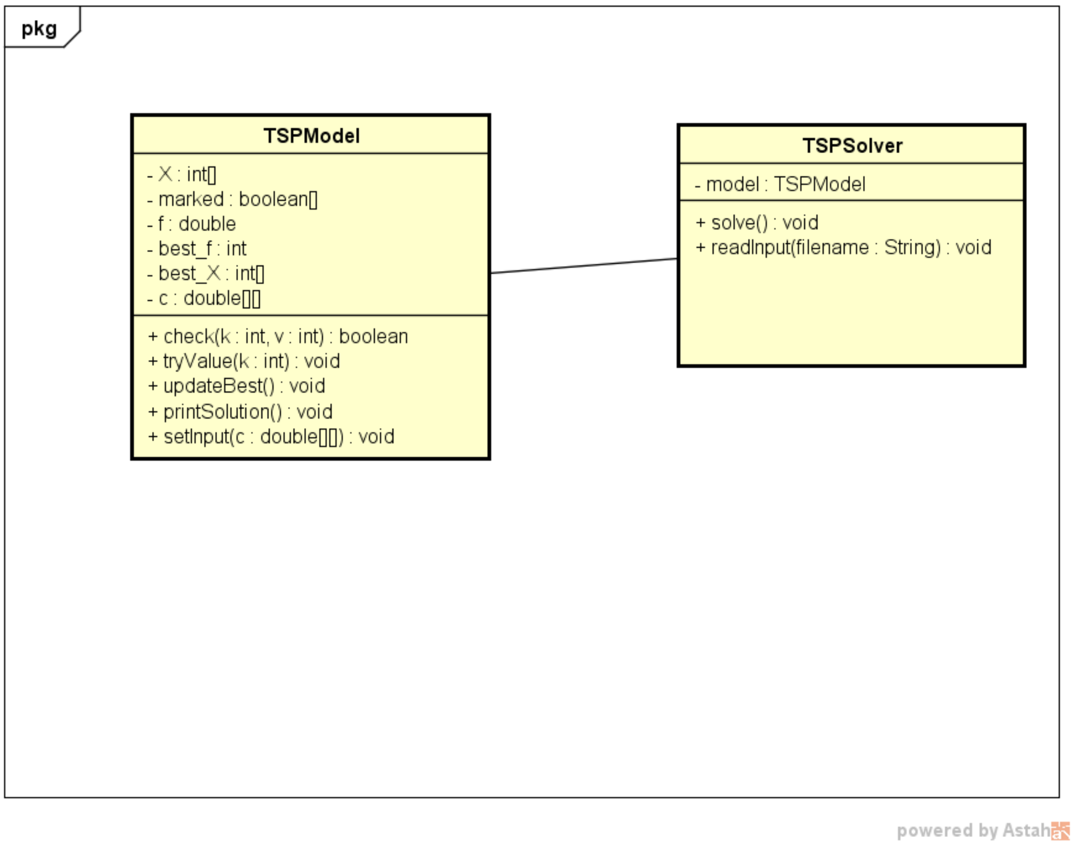
3. Shiping route

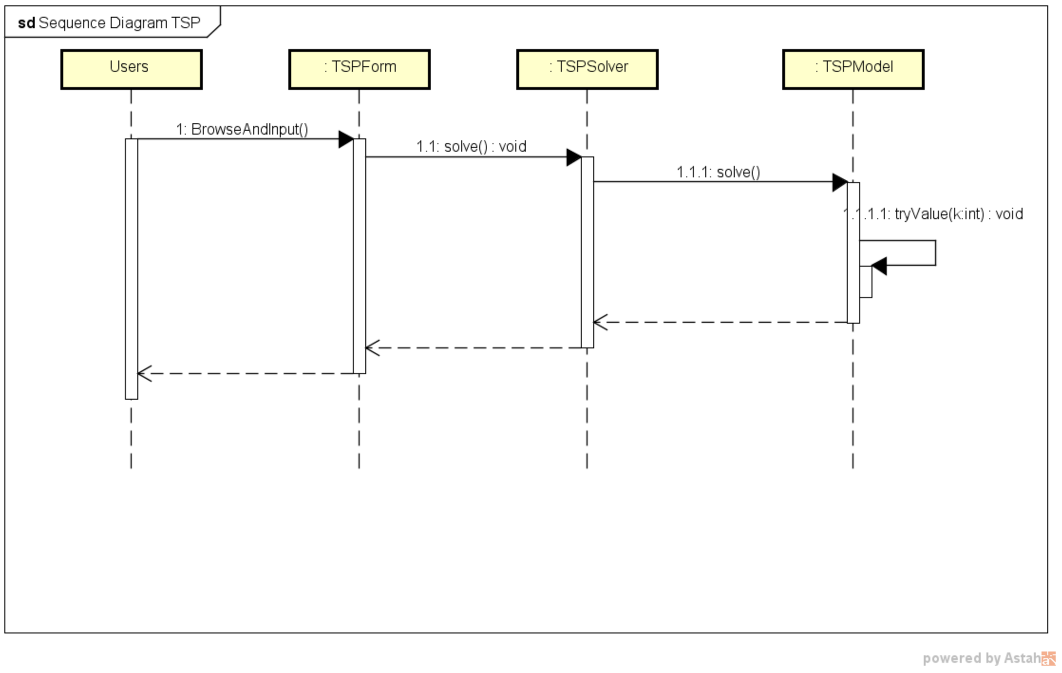
a. Problem description

A shipper must delivery goods to N customers represented by points 1, 2, …, N from the store represented by the point 0. The travel distance from point i to point j is $c_{i,j}$. Find the shipping route from the store, visiting N customers and returning to the store such that to total travel distance is minimal.

b. Design

- Use backtracking search to investigate all candidate solutions in the solution space.
- Decision variables X[] in which X[i] is the i^{th} point of the route, for all i = 0, 1, 2, …, N: the route of the shipper is X[0] → X[1] → X[2] → . . . → X[N] → X[0] (X[0] = 0 is the starting point).
- method tryValue(int k) tries values for X[k]
- method check(int v, int k) returns true if value v can be assigned to X[k] without violating constraints of the problem (each customer point is visited exactly once).





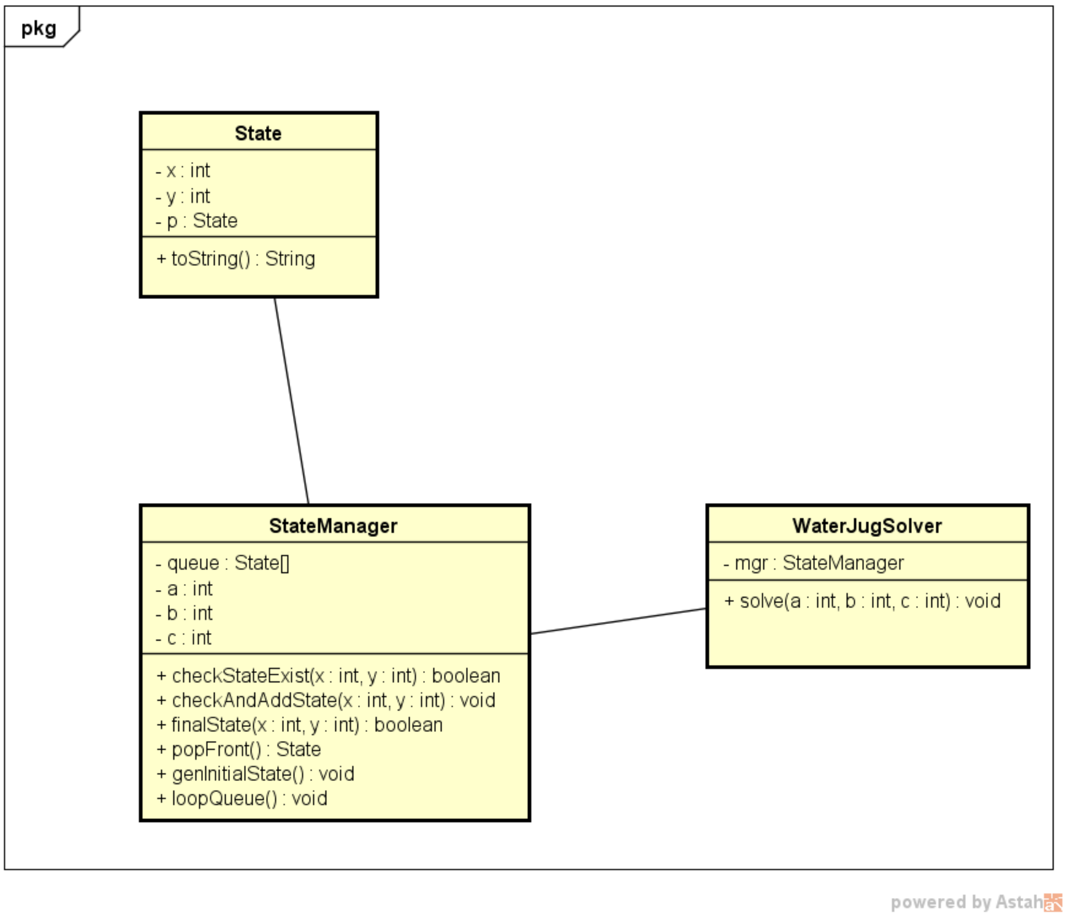
4. Water Jug

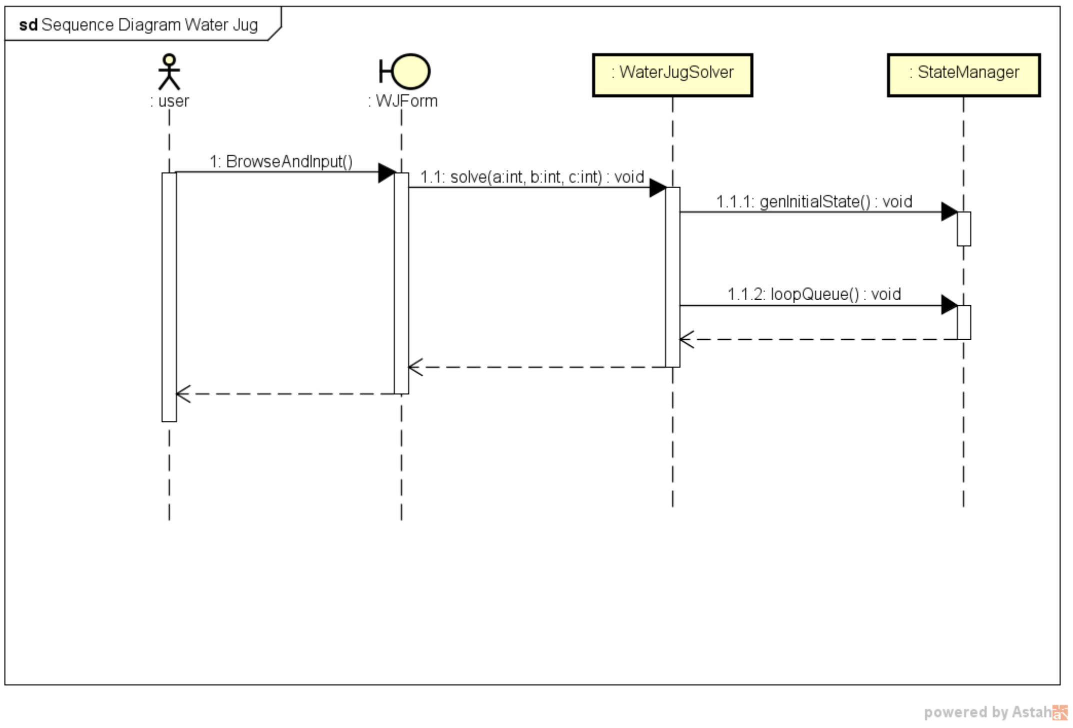
a. Problem description

There are two jugs having capacities *a* and *b* liters. There is a tank with unlimited water. Given a target volumn *c* liters. Find the way to get exactly *c* liters water (*a*, *b*, *c* are positive integers).

b. **Design**

- Use the breadth first search technique to find the optimal move path in a state transition schema. Each state is modelled by a pair (x, y) in which x, và y respectively the amount of water in the jugs 1 and 2. The initial state is (0, 0) and the target state is (x, c) or (c, y) or (x, y) with x+y=c. From the state (x, y), neighboring states can be generated by following operations:
    - Empty the jug 1 → state (0, y)
    - Empty the jug 2 → state (x, 0)
    - Full fill the jug 1 → state (a, y)
    - Full fill the jug 2 → state (x, b)
    - Pour the jug 1 into the jug 2 until the jug 2 is full (if x+y>=b) → state (x+y-b, b)
    - Pour the jug 1 into the jug 2 until the jug 1 is empty (if x+y < b) → state (0, x+y)
    - Pour from the jug 2 into the jug 1 until the jug 1 is full (if x+y >= a) → state (a, x+y-a)
    - Pour from the jug 2 into the jug 1 until the jug 2 is empty (if x+y < a) → state (x+y,0)
- method check(int x, int y) return true if state (x, y) has been generated
- method final(int x, int y) return true if state (x, y) is a final state


pkg
State
- x : int
- y : int
- p : State
+ toString() : String
StateManager
- queue : State[]
- a : int
- b : int
- c : int
+ checkStateExist(x : int, y : int) : boolean
+ checkAndAddState(x : int, y : int) : void
+ finalState(x : int, y : int) : boolean
+ popFront() : State
+ genInitialState() : void
+ loopQueue() : void
WaterJugSolver
- mgr : StateManager
+ solve(a : int, b : int, c : int) : void
powered by Astah


sd Sequence Diagram Water Jug
: user
: WJForm
: WaterJugSolver
: StateManager
1: BrowseAndInput()
1.1: solve(a:int, b:int, c:int) : void
1.1.1: genInitialState() : void
1.1.2: loopQueue() : void
powered by Astah

5. **DataBase**

   a. **Problem description**

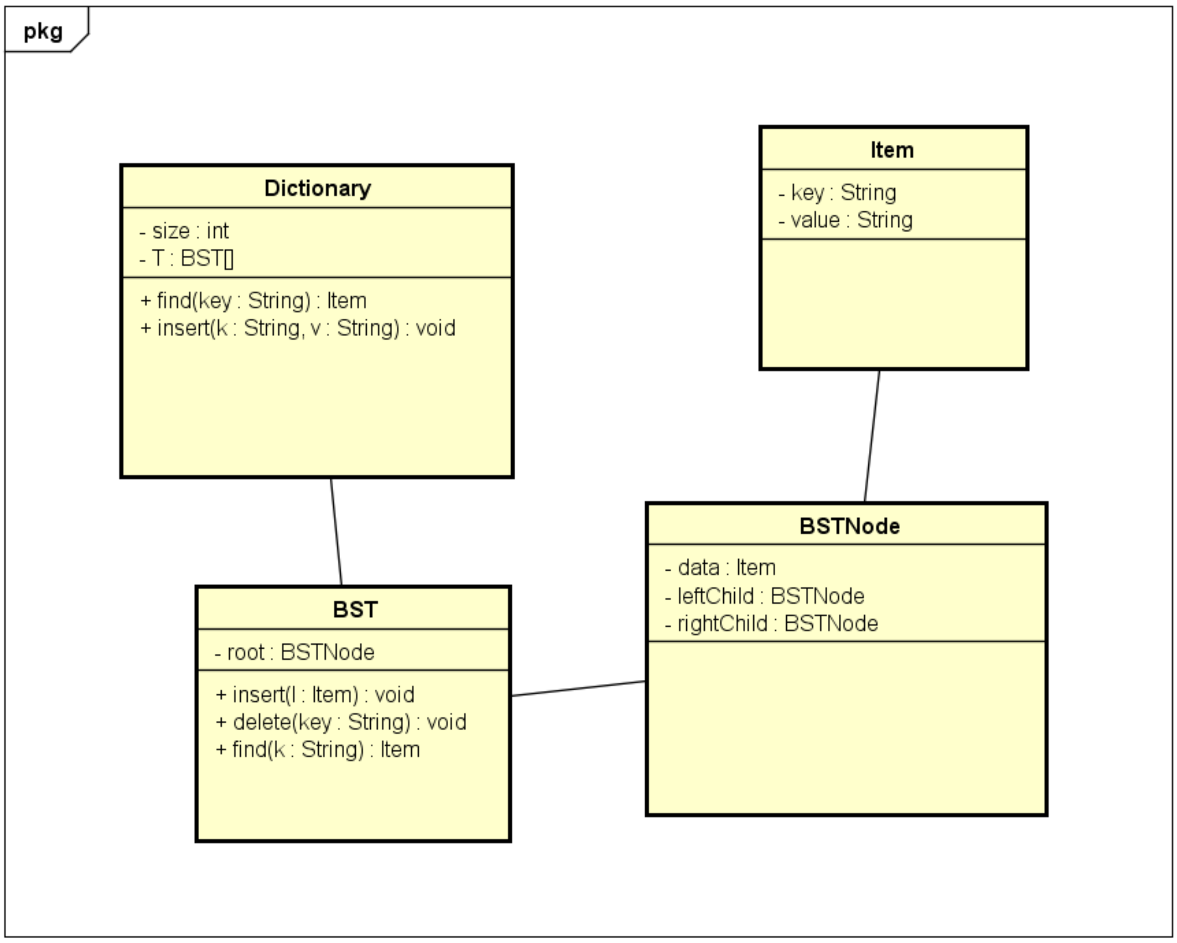
      Build a data structure (database) storing profiles of staffs in an organization. Each staff has a unique ID which is a string and other personal information. The data structure allows users to perfom following operations

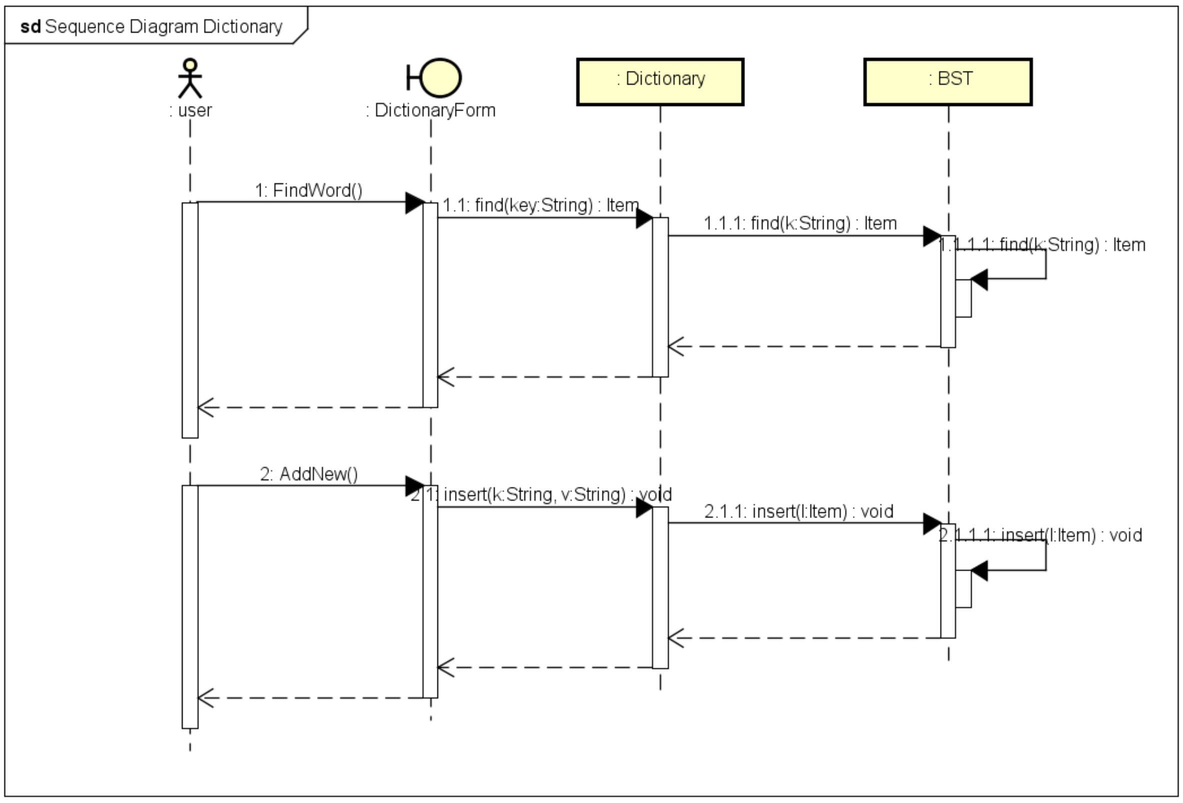
      - add(String ID, Info I): add a profile of a staff to the database
      - find(String ID): find and return the profile of the staff given his/her ID
      - delete(String ID): remove the profile of the staff given his/her ID

      The data structure employ hashing and binary search tree techniques.

   b. **Design**

      - Class Item represents the profiles of staffs
      - Class BST represents binary search trees
      - Lớp BSTNode represents nodes of the BST. It contains an Item and references to the left and right children.
      - Class Dictionary represents the data structure using the hashing technique and binary search trees for separate chaining.


pkg
Dictionary
- size : int
- T : BST[]
+ find(key : String) : Item
+ insert(k : String, v : String) : void
Item
- key : String
- value : String
BST
- root : BSTNode
+ insert(I : Item) : void
+ delete(key : String) : void
+ find(k : String) : Item
BSTNode
- data : Item
- leftChild : BSTNode
- rightChild : BSTNode



sd Sequence Diagram Dictionary
: user
: DictionaryForm
: Dictionary
: BST
1: FindWord()
1.1: find(key:String) : Item
1.1.1: find(k:String) : Item
1.1.1.1: find(k:String) : Item
2: AddNew()
2.1: insert(k:String, v:String) : void
2.1.1: insert(I:Item) : void
2.1.1.1: insert(I:Item) : void

powered by Astah