THE UNIVERSITY of EDINBURGH
informatics

Inf 2D – Coursework 1

Constraint Satisfaction Problems (CSPs)

Jiawei Zheng
jw.zheng@ed.ac.uk
Jan 2022

Aim

- Familiarise yourselves with Constraint Satisfaction Problems (CSPs)
- Implement and evaluate basic algorithms using Haskell.

Tools

- A CSP framework in Haskell.

https://www.learn.ed.ac.uk/bbcswebdav/pid-6107580-dt-content-rid-25602804_1/xid-25602804_1

The framework

- Datatypes and functions for: *(CSPframework.hs)*
  - Variables
  - Assignments
  - Domains
  - Constraints
  - CSPs
- Examples of: *(Examples.hs)*
  - Constraints
  - CSPs

A CSP

- Consists of:
  - **X** : a set of **Variables**
  - **D** : a set of **Domains**
  - **C** : a set of **Constraints**

Datatypes

- `Var` : *String*

  eg. `"X" "Y"`

- `Domain` : *List of Int's*

  eg. `[1,2,3]`

  - Each value can have an implicit meaning

    eg. colour, position, actual value, etc.

- `Domains` : *List associating each variable with a Domain*

  eg. `[("X",[1,2,3]),("Y",[1,2])]`

Assignment

- "State"
- *"Assignment of values to some or all of the variables" - R&N §6.1 / NIE Ch.7 §1*

- Custom datatype - *List of assigned variables*

- Functions:
  - `assign` : *Assign a value to a variable and add it to the list.*
  - `lookupVar` : *Lookup a value of a variable.*
  - `isAssigned` : *Is a variable assigned?*

Constraints

- *"Each constraint Ci consists of a pair: <scope, relation>" - R&N §6.1 / NIE Ch.7 §1*
- Custom datatype – *Pair of a scope and a Relation*
  - Scope as a list of variables:

    eg. `["X","Y"]`
  - Relation as a function:
    - *"Give me a scope and a state and I'll tell you if it's ok!"*

eg. `varsDiff` : *The first two variables in the scope must have different values.*

- `varsDiff ["x","y"] [x=1,y=2]` → `True`
- `varsDiff ["x","y"] [x=1,y=1]` → `False`

Constraint functions

- `checkConstraint` : *Check a constraint on a given state.*
- `checkConstraints` : *Check multiple constraints on a given state.*
- `scope` : *Get the scope of a constraint.*
- `isConstrained` : *Is a variable within the scope?*
- `neighboursOf` : *List of other variables in the same scope.*

A CSP

- Consists of:
  - **X** : a set of **Variables**
  - **D** : a set of **Domains**
  - **C** : a set of **Constraints**

- Custom datatype
  - *Includes a* `Domains` *list and a list of* `Constraints`

- Functions:
  - `cspVars` : Returns **X**
  - `cspDomains` : Returns **D**
  - `cspConstraints` : Returns **C**

CSP functions

- **Domain functions:** `getDomain`, `setDomain`, `addDomainVal`, **etc.**
- **Variable functions:** `firstUnassignedVar`, `constraintsOf`, **etc.**
- **Assignment functions:** `isComplete`, `isConsistent`, **etc.**

The BACKTRACK algorithm

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
        remove {var = value} and inferences from assignment
    return failure
```

The BACKTRACK algorithm

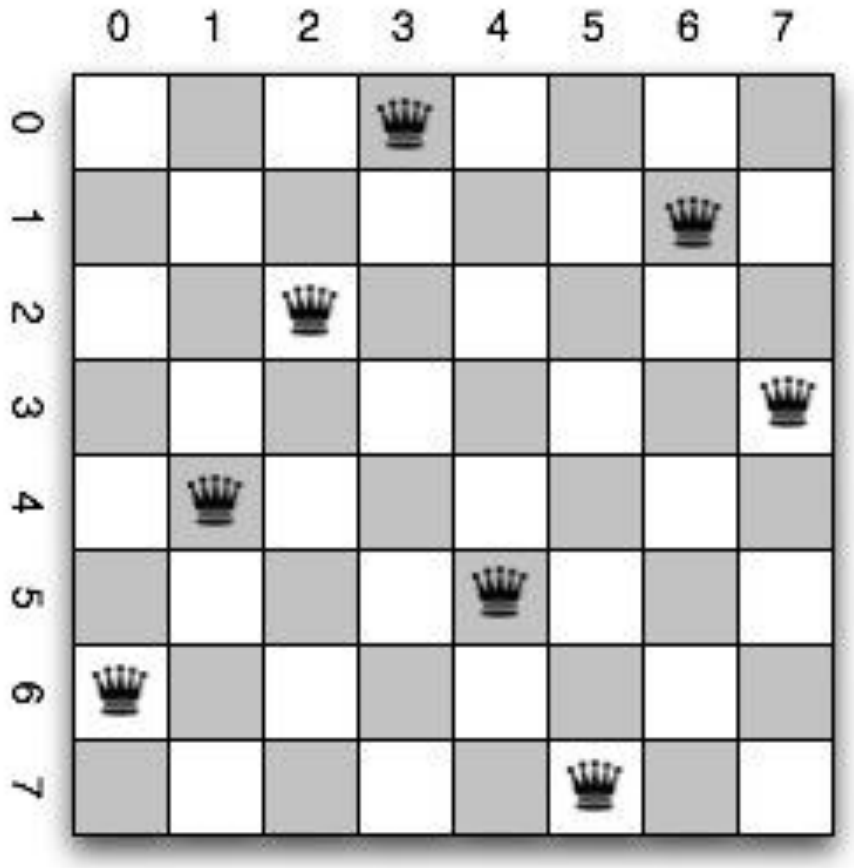
```
bt :: CSP -> Maybe Assignment
bt csp = btRecursion csp []

btRecursion :: CSP -> Assignment -> Maybe Assignment
btRecursion csp assignment =
    if (isComplete csp assignment) then Just assignment
    else findConsistentValue $ getDomain var csp
      where var = firstUnassignedVar assignment csp
            findConsistentValue vals =
              case vals of  -- recursion over the possible values
                            -- instead of for-each loop
                []     -> Nothing
                val:vs ->
                  if (isConsistentValue csp assignment (var,val))
                  then if (isNothing result)
                       then ret
                       else result
                  else ret
                     where result = btRecursion csp $ assign (var,val) assignment
                           ret = findConsistentValue vs
```

Coursework 1

1. The "N-Queens" CSP (10%)
2. CSP Algorithms (45%)
   2.1. Forward Checking (20%)
   2.2. Minimum Remaining Values (MRV) (10%)
   2.3. Least Constraining Value (LCV) (15%)
3. Evaluation (25%)
4. Arc Consistency Algorithm AC-3 (20%)

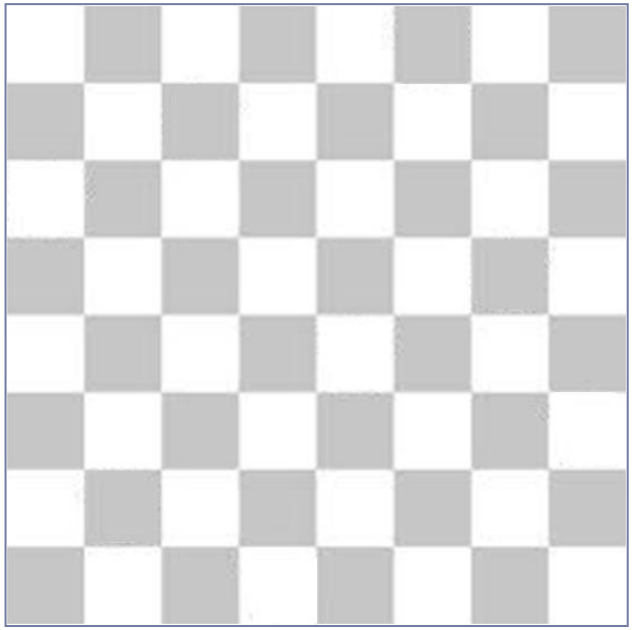
N-Queens as a CSP

N = 8



8-Queens as a CSP

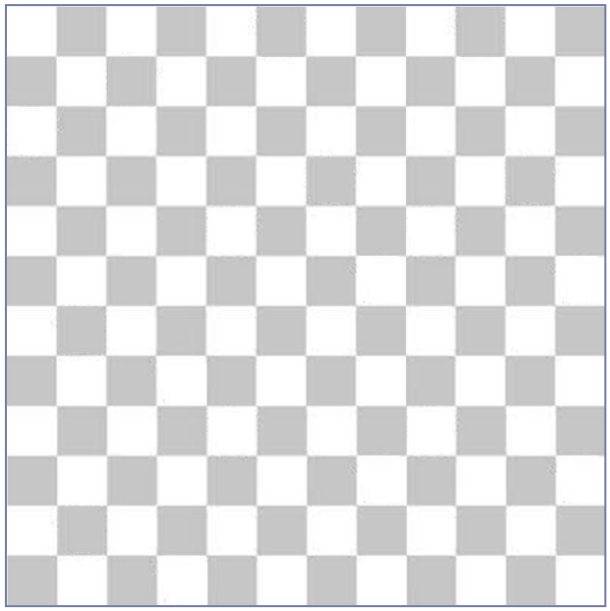
- **8 Queens** 
- **8 x 8 grid**
- **Queens attack:**
  - Horizontally
  - Vertically
  - Diagonally



- ***Fill in the grid so that no queens attack each other.***

12-Queens as a CSP

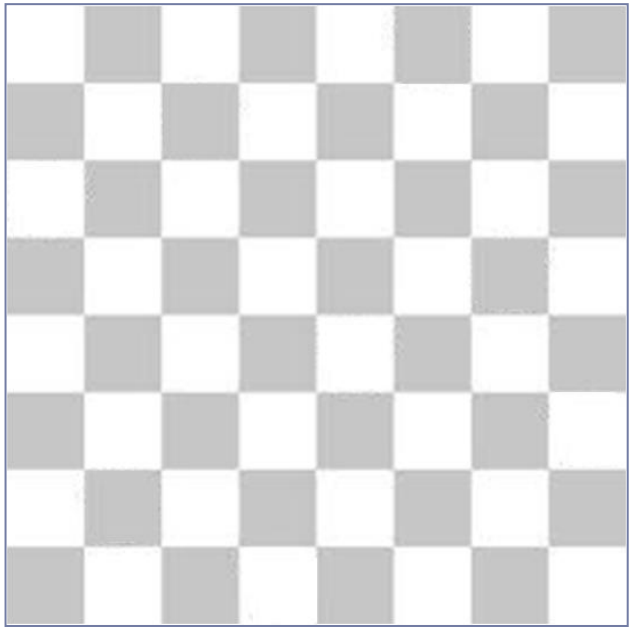
- **12 Queens** 
- **12 x 12 grid**
- **Queens attack:**
  - Horizontally
  - Vertically
  - Diagonally



- ***Fill in the grid so that no queens attack each other.***

N-Queens as a CSP

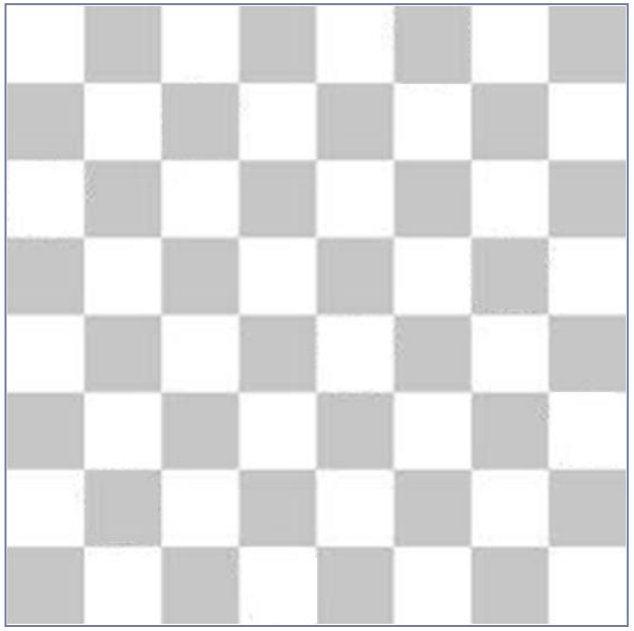
- **N Queens** 
- **N x N grid**
- **Queens attack:**
  - Horizontally
  - Vertically
  - Diagonally



- ***Fill in the grid so that no queens attack each other.***

N-Queens as a CSP **–Hint–**

- **N Columns** 
  - Each queen its own column!
- **N Rows**
- **Queens attack:**
  - Horizontally
  - ~~**Vertically**~~
  - Diagonally



- ~~***Fill in the grid***~~ ***Assign row numbers to queens so that no queens attack each other.***

Task 1 (10%)

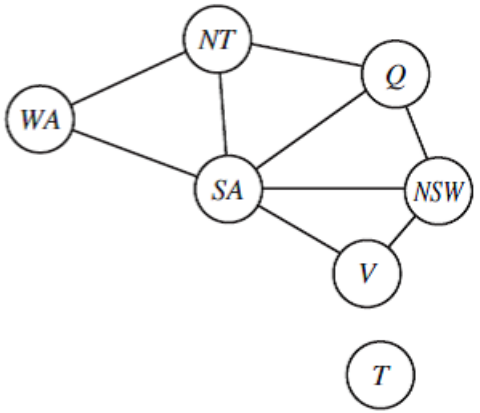
- Define the "N-Queens" CSP for any N within the framework.

- *Tip: Check examples in framework: Map of Australia, Map of Scotland, 3x3 Magic Square, Sudoku*

- Define the variables and their domains.
- Define the diagonal constraint by implementing a Relation:
  - `diagonalRelation` : *Two variables (queens) are **not** in the same diagonal.*
- Define any other necessary constraints (if any) by implementing new Relations or reusing existing ones.
- Test using BT.

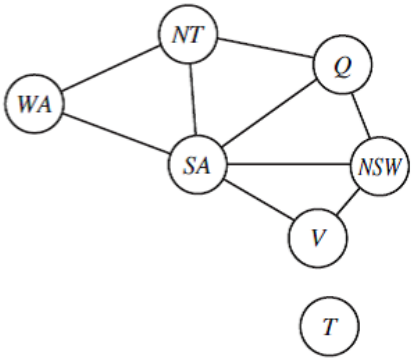
Forwardcheck (20%)

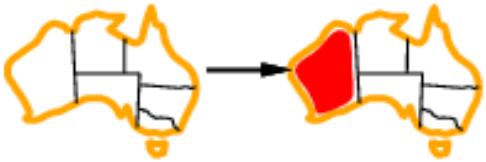
- Whenever a variable *X* is assigned, the forward checking process looks at each unassigned variable *Y* that is a neighbour of *X*.
- Deletes from *Y*'s domain any value that is inconsistent with the value chosen for *X*.



Forwardcheck (20%)

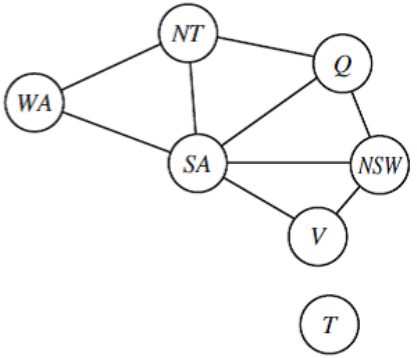
- Whenever a variable *X* is assigned, the forward checking process looks at each unassigned variable *Y* that is a neighbour of *X*.
- Deletes from *Y*'s domain any value that is inconsistent with the value chosen for *X*.





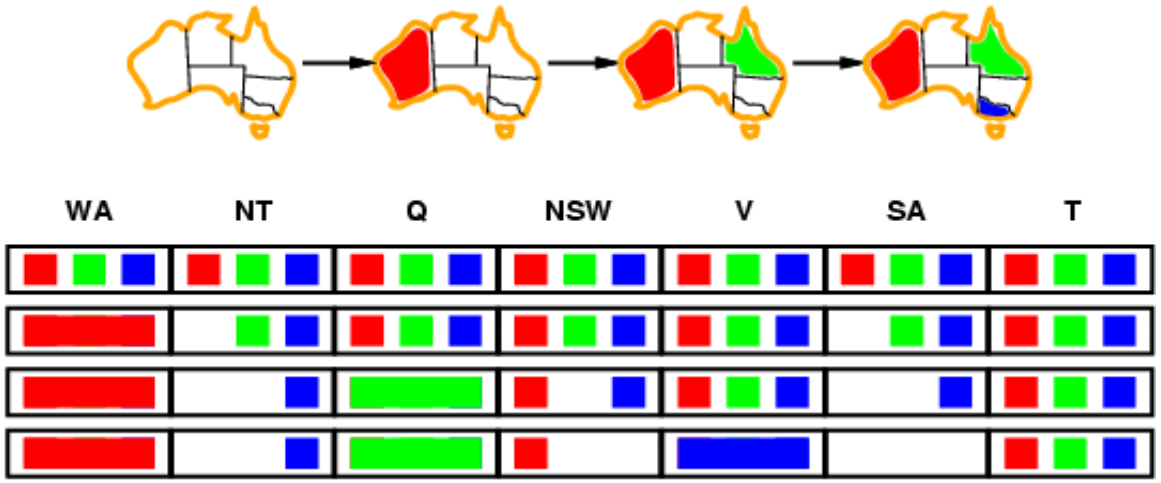
Forwardcheck (20%)

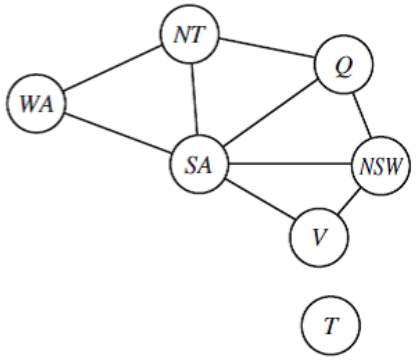
- Whenever a variable *X* is assigned, the forward checking process looks at each unassigned variable *Y* that is a neighbour of *X*.
- Deletes from *Y*'s domain any value that is inconsistent with the value chosen for *X*.



Forwardcheck (20%)

- Whenever a variable *X* is assigned, the forward checking process looks at each unassigned variable *Y* that is a neighbour of *X*.
- Deletes from *Y*'s domain any value that is inconsistent with the value chosen for *X*.





Forwardcheck (20%)

- Whenever a variable *X* is assigned, the forward checking process looks at each unassigned variable *Y* that is a neighbour of *X*.
- Deletes from *Y*'s domain any value that is inconsistent with the value chosen for *X*.
- Implement:
  1. `forwardcheck` : *Given X and the current state, apply forwardchecking.*
  2. `fcRecursion` : *Similar to* `btRecursion` *but uses forwardchecking.*

Minimum Remaining Values (MRV) (10%)

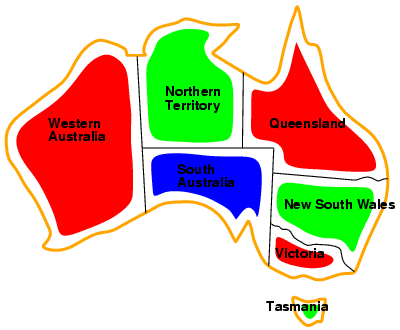
- **Variable ordering** heuristic.
- Selects the *variable* that has the *least available values*.
- Implement:
  1. `getMRVVariable` : *Returns the unassigned variable that has the smallest domain.*
     - *Tip:* Use `sortBy`!
  2. `fcMRVRecursion` : *Same as* `fcRecursion` *but with MRV!*

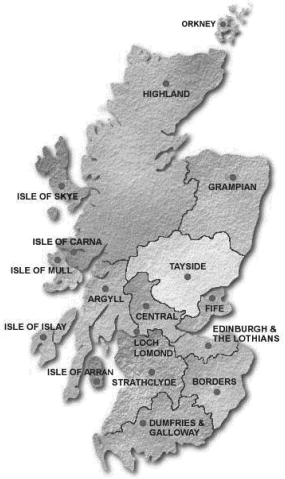
Least Constraining Value (LCV) (15%)

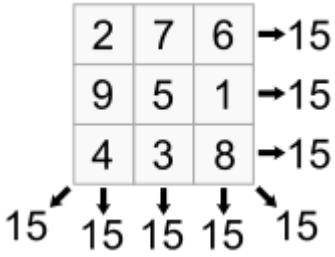
- **Value ordering** heuristic.
- Selects the *value* that has the value *that allows the most choices for the neighbours*.
- Implement:
  1. `lcvSort` : *Returns domain of a variable sorted with LCV.*
  2. `fcLCVRecursion`
  3. `fcMRV_LCVRecursion`

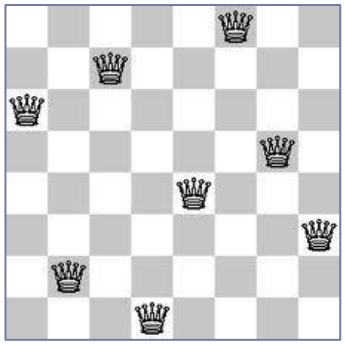
3. Evaluation & Discussion (25%)

- ***5*** *algorithms*:
  - **BT FC FC+MRV FC+LCV FC+MRV+LCV**
- ***5*** *problems:*

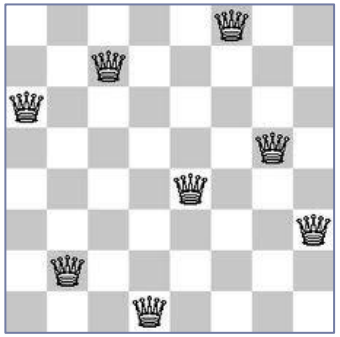








N = 8



N = 12

- Evaluation: 5x5 Table of *visited nodes*.
  - A search node is considered *visited* when a value has been assigned to a variable.

Evaluation & Discussion (25%)

- Report (write in the code file):
  1. Explain *briefly* the values on the table.
  2. Compare the following pairs of algorithms:
     - BT *vs.* FC
     - FC *vs.* FC+MRV
     - FC *vs.* FC+LCV
     - FC+MRV *vs.* FC+LCV
     - FC+MRV *vs.* FC+MRV+LCV
  3. Compare values, and explain the differences.
     - Expected or not?
  4. Is there a **simple**, **effective** optimisation for FC?
     - Consider large scale CSPs
     - Give suggestions and arguments
- **No** longer than **one A4** page of plain text.

Arc Consistency Algorithm AC-3 (20%)

- Constraint propagation algorithm.
- An arc between two variables *X* and *Y* is *consistent* if
  - for every value *x* of variable *X*
  - there is a *possible* value of *Y* that is consistent with *x*.
- If there is a value *x'* such that if it is assigned to *X* and there is **no consistent value** for *Y* then we *remove x'* from *X*'s possible values (domain).

- eg. "X" ∈ [1,2] – "Y" ∈ [1] – Constraint: X≠Y
  - To make X→Y consistent, AC-3 will *revise* the domain of *X* by *removing* 1.

Arc Consistency Algorithm AC-3 (20%)

- R&N - Figure 6.3

```
function AC-3( csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

```
function REVISE( csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Arc Consistency Algorithm AC-3 (20%)

- Implement:
  1. `revise` : *The REVISE function of Fig 6.3. Returns a possibly revised CSP and True if the domain of Xi was revised (False if not).*
  2. `ac3Check` : *The AC-3 function of Fig 6.3. Returns an updated CSP and True if no inconsistencies were found (False otherwise).*
  3. `ac3Recursion`
  4. `ac3MRV_LCVRecursion`

Arc Consistency Algorithm AC-3 (20%)

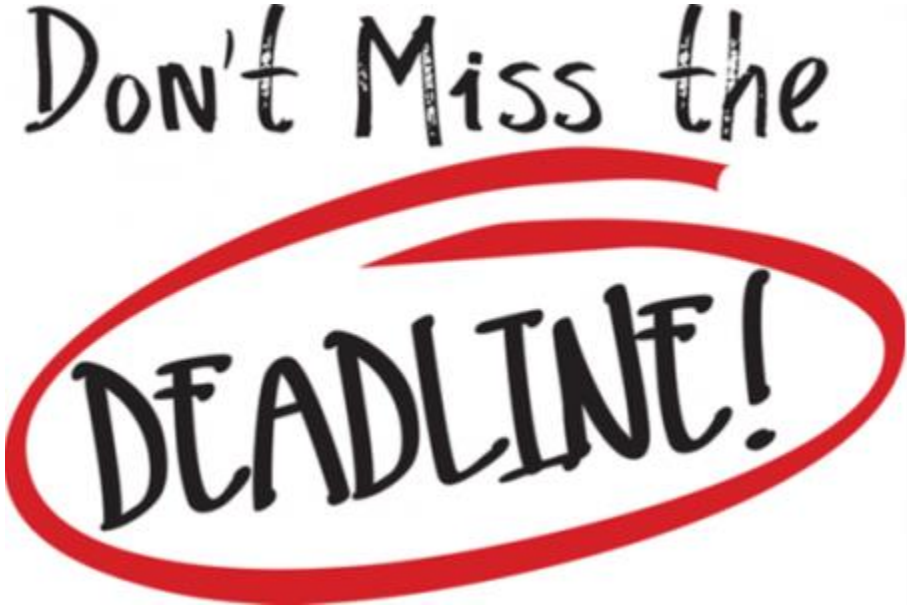
- Report:
  - Add visited nodes to the table.
  - Compare:
    - FC *vs.* AC3
    - FC+MRV+LCV *vs.* AC3+MRV+LCV

- **No** longer than an additional **half A4** page of plain text.

Help!

- Theory: *Week 2 CSP lecture*
- *R&N §6.1-6.3 / NIE Ch.7 §1-3*
- Framework: CSP documentation *and Example code*
- Drop-in lab
- Questions on Piazza
- Haskell: *www.haskell.org*
- Haskell refresher lecture: *Week 3 Tuesday, 01/02/2022*

- **Start early!**
- Don't change types or functions!
- Don't re-implement! Do re-use!
- **Any** custom functions you want **BUT** comment and explain!
- Avoid constructors (AV, CT, CSP).
- Play around with values – create **unit tests**.
- **Max runtime**: 3 minutes
  - (NOT for Sudoku)
- Comment!
- Compile!
- No plagiarism!!
- **Start early!!**



8th March 2022 – 3pm

Extra: Sudoku Competition!

Please do not let this be a priority and/or sidetrack you from the actual assignment.

***No* extra marks will be awarded through this competition.**

- Fastest AC-3 based sudoku solver!
  - Will be tested on various random sudoku problems based on the implementation found in Examples.hs.

- Eligibility:
  1. Correct implementation of AC-3 + heuristics + optimisations
  2. 80% in the assignment.
  3. Code included in Inf2d.hs

Extra: Sudoku Competition!

Please do not let this be a priority and/or sidetrack you from the actual assignment.

***No* extra marks will be awarded through this competition.**

- Prizes:
  - 1st 15£ Amazon voucher
  - 2nd 10£ Amazon voucher
  - 3rd 5£ Amazon voucher

- More details and helpful commands will be sent through the mailing list.


GOOD
LUCK!


THANK YOU!
thank you!