Assignment

Write a program that uses a generic hill-climbing solver, and can solve either N-queens or the knapsack problem (see homework 1).

Grading

- out of 100: 90 points for correct behavior: 50 for hill-climbing, 20 for each N-queens and knapsack for problem formulation
- 10 points for well written solver code: clear data structures, methods, control flow, no commented code, global variables, etc.

Expect us to run your program against a number of different input graphs.

Libraries: it is acceptable to use an external json reading library (see below), but that is all. It is not acceptable to copy or use code for hill climbing and/or either of the problems you are asked to solve

Running your program

A program run should look like: (may be slightly different depending on language, note this in your README)

hill_climb -N $queens
OR
hill_climb $knapsack-file

Either program run should accept these additional arguments

- -verbose is an optional flag for verbose mode (more later)
- -sideways is followed by an integer >= 0 for the number of allowed sideways motions (0 or unset means none)
- -restarts is followed by an integer >= 0 for the number of allowed random restarts (0 or unset means none)

Your program should exit gracefully and indicate as best as possible the problem in case of bad arguments (e.g. -sideways not an int) or a bad file.

N-Queens

If -N is set, then you should assign a start state of Queen 1 in R=1, C=1, Queen 2 in R=2, C=2 ... Queen N in R=N, C=N.

State should be represented as an array of the row assignments for each column (see outputs), with actions as discussed in class of swapping row assignments.

Knapsack File Input

Input should be a json file with the following contents.

- T - the integer target
- M - the integer max weight
- Start - a string list of items to start in the knapsack
- Items - an object array, where each object is a triple of name, V, W (V is value, W is weight)

Missing T or M is an error, missing Start means start with an empty knapsack.

Algorithms

You are to implement the hill-climbing algorithm discussed in class. This should be coded generically (once) and allow for an interface input that represents either hill-climbing or knapsack. The state interface will likely need:

- Value() returning the int error function where <=0 is the goal
- Next() returning the list of states reachable from the current state
- Restart() returning a random restart state.
- Print() either printing or returning a string representation to be printed for regular and verbose mode.

Hill climbing, options in the algorithm:

- By default, there is no sideways motion, but if -sideways is > 0, allow each sideways that many steps (resets if you go downhill). Also this requires a short-term visited list of states to avoid sideways cycles
- If there is a failure, exit unless -restarts > 0, in which case do a random restart until either a solution found or attempts are exhausted
- Random restart: for knapsack that means a random collection of items in the "bag"; for N-Queens it means a random placement into unique row/column assignments.
- Tie-breakers: for knapsack, as in HW2, higher value followed by lower weight; for N-Queens choose randomly.

Output

By default you print each selected state, as well as goal reached and random restarts

- if -verbose is passed, you also print the 'Next' choices at each state being considered (see also examples)

Examples

- Files (attached) knapsack1.txt : knap.1.out or with -verbose knap1.v.out

- knapsack_side.txt : (fails unless -sideways 2 or higher) knap.side3.out or -verbose knap.side3.v.out

- N-Queens format nq.16.s5.r5.v.out means 16 queens, with -sideways=5 and -restarts=5 and -verbose; where nq.8.out is just -N 8 no other parameters

- Note: our N-Queens runs may not be the same due to tie-breaking formula -- and all of our runs may differ once a random restart is used due to random seeding

Submission

- On Brightspace, a zip file containing: Code for the programs (only, no jars, or other compiled products)
- Makefile/BUILD/pom.xml/etc needed to build your code
- README indicating how to compile and run your code
- In your README include any classmates you consulted with, or online resources you used (other than the textbook)
- As previously indicated, the code should build and run on department linux machines