Lista de chequeo - projectlogic

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Respuesta** | **Check** | **FECHA** | **Porcentaje** |
| Lista de participantes  (Restricciones: debe haber a lo sumo un estudiante MACC 1.0) | Camilo Andrés Martínez Mejía y Víctor Samuel Pérez Díaz | OK | 17 de agosto  (1%) | 1 |
| Repositorio github | https://github.com/BogoCoder/logicproject | OK | 24 de agosto  (1%) | 1 |
| Situación a representar  (Resumen) | Completar un sudoku 4x4 según las reglas del juego. | OK | 7 de septiembre  (2%) | 2 |
| Situación a representar  (archivo en repositorio) | problema\_sudoku.pdf, ejemplo\_sudoku.pdf | OK |
| Representación  (Resumen) | A través de fórmulas lógicas buscamos representar las reglas de un sudoku 4x4 y ejemplificar la resolución de este. | OK | 14 de septiembre  (1%) | 1 |
| Representación  (archivo en repositorio) | claves\_rep.pdf | OK |
| Implementación gráfica de la representación  (uso en línea de comando) | $ python visualizacion.py tableros.csv  “tableros.csv” es un archivo donde cada línea contiene la representación de un tablero de sudoku resuelto, en forma de una lista de literales.  Consideraremos tres posibles soluciones a un sudoku 4x4, las cuales están presentes en el archivo. | OK | 7 de  octubre  (1%) | 1 |
| Implementación gráfica de la representación  (archivo en repositorio) | visualizacion.py  Visualización de tableros de sudoku 4x4 a partir de una lista de literales. Cada literal representa un estado de una casilla; el literal es positivo sii el número que representa está en la casilla correspondiente.  Es decir: 'p', 'q', 'r' y 's' representan respectivamente los números '1', '2', '3' y '4'.  El índice ‘i’ del literal representa la casilla a la que corresponde, es decir:  'pi': si 'i' = 1, 'p1' representaría que el número '1' está en la casilla 1.  'si': si 'i' = 9, '~s9' representaría que el número '4' NO está en la casilla 9. Y de la misma forma con los demás…  Formato de la entrada:  - Las letras proposicionales serán: ‘pi’'pi','qi','ri','si', i perteneciendo a {1, ... , 16};  - Solo se aceptan literales (ej. p1, ~q2, r3, ~r12, etc.)  Requiere también un número natural, para servir de índice del sudoku toda vez que se dibuje uno nuevo.  Salida: archivo sudoku\_%j.png, donde %j es un número natural. | OK |
| Aplicación mediante tableaux  (uso en línea de comando) | $ python resolver\_tableau.py  “resolver\_tableau.py” se encargará de darnos todas las posibles soluciones dadas nuestras reglas A, B, C, D, y si se quiere, unas condiciones parciales.  El código está adaptado para dar todas las posibles soluciones a las dos primeras casillas de un sudoku parcialmente resuelto.  Los .png presentados en la carpeta /Soluciones corresponden al resultado de este proceso. | OK | (3%) | 3 |
| Aplicación mediante tableaux  (archivo en repositorio) | resolver\_tableau.py  Código para crear la fórmula para la resolución de sudokus, y así mismo invocar la solución por medio de tableaux y la posterior visualización.  Este código está adaptado para solucionar las dos primeras casillas de un sudoku parcialmente lleno.  ATENCIÓN: Este procedimiento dura aprox. 2 horas en un computador de hogar.  Si se quiere simplificar la ejecución para ver un resultado, puede eliminar una regla de la variable ‘lista\_hojas’, recomendamos ‘B’ en este caso.  Si así se quiere, **se puede adaptar para la solución de todo un sudoku 4x4**. Para hacer esto modifique la variable 'val', haga val = 16  PELIGRO: Esto podría tomar la edad del universo en un computador de hogar.  También se puede adaptar para la resolución de diferentes números de casillas haciendo la variable ‘val’ igual a un número n en el intervalo [2,16].  Cada fórmula de las reglas está representada por R1, R2, R3, R4 respectivamente.  Cada una tiene su sección de creación, para posteriormente ser transformadas a objetos .Tree en las variables A,B,C,D respectivamente.  Las condiciones para el sudoku parcialmente lleno, sudoku a solucionar, están dadas en la variable 'sudoq' que luego es convertida a objeto .Tree en la variable 'Z'.  La variable 'sudoq' puede ser modificada si se quiere resolver un sudoku particular, claro está siguiendo las reglas de representación. | OK |
| Aplicación algoritmo DLL  (uso en línea de comando) | $ python3 resolver\_dpll.py  “resolver\_dpll.py” se encargará de darnos una solucion dadas nuestras reglas A, B, C, D, y si se quiere, alguna condición inicial.  El código está adaptado para dar una solución completa a un sudoku parcialmente lleno.  El .png presentado en la carpeta /SolucionesDPLL corresponden al resultado de este proceso.  OJO: Aségurese de tener las librerías necesarias para python3. La que suele generar conflicto es matplotlib.  $ pip3 install matplotlib | OK | 9 de noviembre  (3%) | 3 |
| Aplicación algoritmo DLL  (archivo en repositorio) | resolver\_dpll.py  Código para crear la fórmula para la resolución de sudokus, y así mismo convertir a forma clausal, para invocar el algoritmo DPLL, que en este caso, nos retornará una solución.  Este código está adaptado para solucionar un sudoku parcialmente lleno.  Duración apróximada: 5 segundos.  Si así se quiere, se puede adaptar para la solución de todo un sudoku.  Para hacer esto asegurese de que la variable val esté definida en 16. Posteriormente, vaya abajo y lea la documentación respecto a la variable ‘conjunciones’.  También se puede adaptar la resolución de diferentes números de casillas, haciendo la variable val igual a un número n en el intervalor [2,16].  Cada fórmula de las reglas está representada por R1, R2, R3, R4 respectivamente.  Cada una tiene su sección de creación, para posteriormente ser unidas a través de conjunciones y luego ser transformadas a un conjunto de cláusulas.  Las condiciones para el sudoku parcialmente lleno, sudoku a solucionar, están dadas en la variable ‘sudoq’, la cual puede ser modificada si se quiere resolver un sudoku particular, claro está, siguiendo las reglas de representación. Luego ‘sudoq’ se concatena con las demás reglas y se une a través de una conjunción.  DPLL.py  Código principal del algoritmo DPLL, con sus respectivas subrutinas.  cnf.py  Código para pasar de polaca inversa a una lista de cláusulas.  Todo esto está en programado en python3. | OK |
| Sustentación | ¡Proyecto sustentado!  Thanks! | OK | 21 de noviembre | 3 |