Problema de las 8 Reinas
Programación IV
Laboratorio: 1

Héctor F. Jiménez Saldarriaga
hfjimenez@utp.edu.co
PGP KEY ID: 0xB05AD7B8

Fecha de Entrega: **14** Febrero,2016
Profesor: Ing. Ángel Augusto Agudelo Zapata

1 Restricciones de Objetivo

Analizar, diseñar e implementar el problema de las 8 reinas en el lenguaje deseado.

- El tamaño del tablero debe ser variable.
- Debe tomar tiempos para saber lo que se demora el algoritmo seleccionado.
- Debe mostrar el resultado en pantalla
- Debe generar una animación

2 Análisis de Problema

Para entender el problema a realizar se hizo la lectura del articulo provisto por el docente a cargo del autor Niklaus Wirth[1] en el articulo se plantea en como se debería de llevar acabo un diseño cuidadoso y modular cuando se intenta resolver un problema, el problema en contexto es el análisis del problema de las 8 reinas utilizando *step refinement*, la idea es descomponer la solución planteada tanto como sea posible, estableciendo adecuadamente las p posibles condiciones que acortan la solución. Para el problema de las 8 reinas ***no existen*** soluciones analíticas y se deben explorar todas las posibles soluciones esto por obvias razones una solución por fuerza bruta seria costosa temporal y especialmente dado

1. https://www.inf.ethz.ch/personal/wirth/Articles/StepwiseRefinement.pdf

que tomaría $\binom{64}{8} = \frac{64!}{8!(64-8)!} = 4,426,165,368$ combinaciones, por tal motivo es necesario reducir la búsqueda de soluciones a una que cumpla los requisitos y condiciones establecidas por nosotros; utilizando *shortcuts* como lo menciona Niklaus tenemos en cuenta el movimiento ofensivo de la reina que puede atacar a i elementos en su fila y columna correspondiente, así que solo debe de haber una y solo una reina por fila,columna x, y del tablero, con esto podemos deducir que debemos permutar la n cantidad de reinas que deseemos poner en el tablero, reduciendo esto a un número inferior de posibles 40,320 posibles ubicaciones, esto corresponde a 8! = 40320. Nuestra próxima condición será verificar los ataque en diagonales, que para para cada permutación hallada, que representa un conjunto de posibles soluciones que satisfacen que ninguna reina se ataque a nivel horizontal y vertical; si hay almenos una solución dejaremos de buscar más sobre el conjunto de permutaciones.

3 Solución del Problema

Teniendo en cuenta los hechos mencionados anteriormente asumiré que mi tablero máximo será un tablero de $n = 10$ ya que debo acotar la solución de mi problema, además encontrar la cantidad de permutaciones en c++ no parece una tarea tan fácil con un de n mayor a 10 ya que toma algo de tiempo, como lo sería con el paquete **itertools** de **Python3** como se realiza la prueba[2] . Para resolver el problema Representamos las n reinas mediante un vector$[1-n]$, teniendo en cuenta que cada índice del vector representa una fila y el valor de una columna. Así cada reina estaría en la posición $(i, v[i])$ para i = 1-8, seguidamente se realizaría una permutaciÓn usando el método *next_permutation*, c++ stl[3] del vector anterior para obtener todas las posibles combinaciones de reinas en el tablero de ajedrez, esto nos asegurara que no existan mas de una reina por fila y columna, dejandono como ultimo paso la validación de nuestra condición p para verificar si las reinas ofensivamente tiene alguna posibilidad entre si en una diagonal ascendente o descendente, tome como ejemplo un par de reinas $Q1 : (i, j)$,$Q2 : (k, l)$ estas estarán en la misma diagonal *si y solo si* se comprueban las siguientes condiciones :

$$i - j == k - l || i + j == k + l || j - l == i - k || j - l == k - i$$

Si nuestro vento no retorna verdad en las condiciones anteriores hemos encontrado una solución, Por lo tanto procederemos a ir graficando las no soluciones en nuestro board hasta llegar al vector solución.El cumplimiento de esta función se puede observar en el fragmento de código 1,2 implementado en c. Se decidio usar **c++11** ya que era lo que conocía, para medir el tiempo utilizare la librería chrono[4] y se puede contrastar con el tiempo que me tire la utilidad **time** de Unix y presente en sistemas Gnu Linux, para realizar SFML[5], y me he basado

2. `https://github.com/h3ct0rjs/ProgrammingIVassignments/blob/master/Lab1/nreinas.py`
3. `http://www.cplusplus.com/reference/algorithm/next_permutation/`
4. `http://www.cplusplus.com/reference/chrono/`
5. `https://www.sfml-dev.org/tutorials/2.0/start-linux.php`

en el tablero realizado por el ruso.

```
bool check(vector <int>vect){
for(auto const&r1: vect){
 for(int j=r1+1;j<vect.size();j++){
 if( (r1-vect[r1] == j-vect[j]) || (r1+vect[r1] == j+vect[j]) || \
 (vect[r1]-vect[j] == r1-j) || (vect[r1]-vect[j] == j-r1) ) {
                                  return 1;                //si hay un elemento en la diagonal
                          }
                 }
         }
         return 0;                                         //Si termino, el vector recibido es
}
```

.

Listing 1: Shortcut p, verificacion de la diagonal ascendente, descendente

```
void nqueens(int n){        //this will create the permutations needed
         int n2=n;
         vector<int> v(n2);
         iota(begin(v),end(v),0);          //fill the vector fast, linear time
         int sol=0;                         //count the number of solutions founded
         do {
                  if(check(v)){
                 Graphsml(v);         //show interactions
         }
                 else {
                         cout<<"Combinación Ganadora!"<<endl;
                         sol++;
                         for(auto const &x: v){
                             cout <<x+1 <<" ";
                     }
                     cout <<endl;
                 }cout<<endl;
                 if(sol>=1) break;
           }while (next_permutation(v.begin(),v.end()) );
      //3!:6,4!:24,5!:120,6!:720,7!:5040,8!:40320,10!:3628800
      }
```

.

Listing 2: Generacion de permutaciones

Después de hacer pruebas con el código, se observo que cuando se le tiraba un n **mayor a** 12 el programa tardaba entre 4 y 5 minutos, lo cual me da una sen-

sación de que realizar las validaciones de la diagonal usando cada permutacion gastaba demasiado tiempo, pues lo que se realizaba era generar la permutacion y verificar si era una posible solución.

```
initializar variable;
while no exista al menos una solucion do
    generapermutacion();
    if nohay ofensiva entre reinas then
        Terminar y mostrar final;
    else
        Graficar en tablero las permutaciones erroneas;
    end
end
```

Algorithm 1: Pseudocodigo

Ahora que pasaría si nosotros almacenáramos primero todas las permutaciones en una lista de vectores, y recorriéramos la lista de manera aleatoria con una buena distribución probabilisticamente podríamos obtener mejores resultados, pero eso sera para una próxima.
Con la idea también de jugar un poco con esto de argc, y argv he añadido las siguientes opciones :

1. -v or - -version * : Current software version
2. -h or - -help * : Show the information usage
3. -t or - -theory : Quick Explanation of how this computer programm works.
4. -nc or - -not-color : Disable ncurses color,this black and white.

4 Anexos

Los archivos correspondientes al proyecto de este laboratorio, junto con su readme y opciones soportadas pueden ser descargados del repositorio [6]

6. `https://github.com/h3ct0rjs/ProgrammingIVassignments/tree/master/Lab1`