INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ESCOM

ESTRUCTURAS DE DATOS

PROF. FRANCO MARTÍNEZ EDGARDO ADRIÁN

SOLUCIONES RECURSIVAS

PRACTICA 5 PROBLEMA N-REINAS

1CM12

COLABORADORES: (VAPORUB.EXE)

AZPEITIA HERNÁNDEZ VLADIMIR

GÓMEZ RODRÍGUEZ EDUARDO

RAYA TOLENTINO PAOLA

Introducción

Backtracking (o búsqueda atrás) es una técnica de programación para hacer búsqueda sistemática a través de todas las configuraciones posibles dentro de un espacio de búsqueda.

Los algoritmos de tipo backtracking se usan para encontrar una solución, pero otras veces interesa que las revisen todas (por ejemplo, para encontrar la más corta). Esencialmente, la idea es encontrar la mejor combinación posible en un momento determinado, por eso, se dice que este tipo de algoritmo es una búsqueda en profundidad. Durante la búsqueda, si se encuentra una alternativa incorrecta, la búsqueda retrocede hasta el paso anterior y toma la siguiente alternativa. Cuando se han terminado las posibilidades, se vuelve a la elección anterior y se toma la siguiente opción. [1]

Planteamiento del problema

Utilizando el esquema de backtracking solucionar el problema de las N-Reinas, que consiste en colocar n reinas en un tablero de ajedrez de tamaño N*N de forma la reinas no se amenacen según las normas del ajedrez. Se busca encontrar una solución o todas las soluciones posibles.

Las restricciones para este problema consisten en que dos reinas no pueden colocarse en la misma fila, ni en la misma columna ni en la misma diagonal.

Diseño y funcionamiento de la solución.

Tablero de ajedrez.

	0	1	2	3	4	5	6	7
0	Q							
1							Q	
2					Q			
3								Q
4		Q						
5				Q				
6						Q		
7			Q					

Para la solución del problema propusimos guardar las coordenadas de posición (x, y) de las reinas en el tablero en un arreglo de enteros. En donde:

X--> está dada por el índice del arreglo [0 hasta número de reinas]

Y-->está dada por el valor que almacena el arreglo en esa posición

Para comprobar que las reinas no se atacan:

- En caso de las filas, lo damos por hecho, ya que se coloca una reina en cada fila o reglón.
- En caso de las columnas hacemos una operación sencilla, que consiste en comparar el valor del arreglo en esa posición con las posiciones anteriores, si hay un valor que sea igual, entonces esas reinas se están atacando.
- En caso de la diagonal, comprábamos si la resta de la posición anteriores es igual a la resta del arreglo en esa posición menos el valor del arreglo en las posiciones anteriores.

Implementación de la solución

Funcion main

En la función main se hace la declaración de las variables que se van a utilizar dentro de la implementación del lenguaje de programación.

```
int main ()
{
  int *reinas;  // Vector con las posiciones de las reinas de cada fila
  int nreinas;  // Número de reinas
  int i;        // Contador
  int alto=0;

  printf("Ingresa el numero de reinas: ");
  scanf("%d", &nreinas);

  // Colocar las reinas en el tablero

  // Crear vector dinámicamente

      reinas = (int*) malloc ( nreinas*sizeof(int) );

  // Inicializar vector:
  // (inicialmente, ninguna reina está colocada)
      system("cls");
  for (i=0; i<nreinas; i++)
    reinas[i] = -1;
   // Colocar reinas (algoritmo recursivo)
   nReina(0,reinas,nreinas, &alto);
   // Liberar memoria
   free (reinas);
}
```

Función nReina

Colocación de una reina

Parámetros: fila(Fila de la reina que queremos colocar) reinas (Vector con las posiciones de las reinas), n (Número de reinas).

```
void nReina (int fila, int reinas[], int n, int *alto)
{
  int ok = FALSE;

  if (fila<n) {

     // Quedan reinas por colocar

     for (reinas[fila]=0; reinas[fila]<n; reinas[fila]++) {

         // Comprobamos si la posición
         // de la reina actual es válida

         if (prueba(fila,reinas,n)) {

            // Si es así, intentamos colocar
```

```
            // las reinas restantes
            nReina (fila+1, reinas, n, alto);
         }
      }

   } else {

      // No quedan reinas por colocar (solución)
      dibuja(reinas,n,alto);
   }

   return;
}
```

Función prueba

Prueba si una reina está bien colocada

La reina de la fila i está bien colocada si no está en la columna ni en la misma diagonal que cualquiera de las reinas de las filas anteriores.

Parámetros: fila (Fila de la reina cuya posición queremos validar), reinas (Vector con las posiciones de las reinas), n (Número de reinas).

```
int prueba (int fila, int reinas[], int n)
{
   int i;

   for (i=0; i<fila; i++)
      if (  ( reinas[i]==reinas[fila] )                    // Misma
columna
         || ( abs(fila-i) == abs(reinas[fila]-reinas[i]) ) ) // Misma
diagonal
         return FALSE;

   return TRUE;
}
```

Función dibuja

Mostrar el tablero con las reinas

Parámetros: reinas (Vector con las posiciones de las distintas reinas), n (Número de reinas)

```
void dibuja (int reinas[], int n, int *alto)
{
   int i,j,x=0,y=*alto, aux=0;

   for (i=0; i<n; i++) {
         x=0;
      for (j=0; j<n; j++) {

          if (reinas[i]==j){
            MoverCursor(x,y);
            printf("##");
            MoverCursor(x,y+1);
```

```
            printf("##");
          }else{
            if((j+aux)%2 == 0){
                  MoverCursor(x,y);
                  printf(",,");
                  MoverCursor(x,y+1);
                  printf(",,");
            }else{
                  MoverCursor(x,y);
                  printf("..");
                  MoverCursor(x,y+1);
                  printf("..");
            }
          }
          x+=3;
      }
      y+=2;
      aux=1-aux;
      //printf(" %d %d\n",i,reinas[i]);
      *alto = y+3;
  }
}
```

Funcionamiento.

Primero compilamos y ejecutamos el archivo con los siguientes comandos:

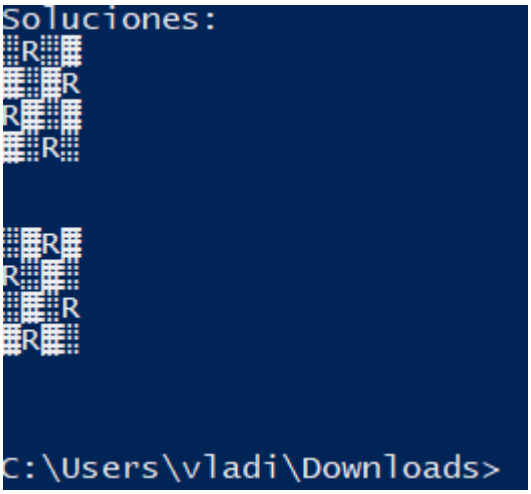
```
C:\Users\vladi\Downloads>gcc -o reinas2 reinas2.c

C:\Users\vladi\Downloads>reinas2
```

Se iniciará el programa. Nos pedirá que ingresemos el número de reinas.

```
C:\Users\vladi\Downloads>reinas2
Ingresa el numero de PrincesaXd: 4
```

Para este ejemplo, lo haremos para 4 reinas, aunque el programa funciona hasta para 10 reinas.

Finalmente, el programa nos mostrara las configuraciones del tablero en donde ninguna reina se ataca.



Errores detectados

Tuvimos un único detalle que no imprimíamos conforme a la especificación de la práctica, pero fue solucionado inmediatamente.

Posibles mejoras

Mejora la forma de impresión para que se vea de una manera más didactica.

Conclusiones

Gómez

Una de las mejores prácticas de recursividad que eh realizado en equipo, ya que conlleva verdadero ingenio, y es necesario hacer uso del backtracking para poder comprender el funcionamiento del programa y del cómo se llega realmente a las soluciones, por eso es que es realmente importante hacer uso de soluciones recursivas bien implementadas y lograr llegar a poder hacer un buen backtrack de un programa para verificar que el funcionamiento que tiene es el ideado desde el principio.

Azpeitia

La práctica fue muy sencilla gracias a los conocimientos que adquirimos con las practicas anteriores. Fue interesante y formativo encontrar un algoritmo que resolviera el problema de colocar las reinas sin que ninguna se ataque, el tema de backtracking quedo muy claro.

Raya

Esta práctica me ayudo a entender mucho mejor el tema de backtracking, ya que en el problema de las n-reinas existen muchas de posibles soluciones y usando el backtracking se pueden ir checando de una forma un poco más rápido y entendible las posibles soluciones.

Anexo

```
#include <stdio.h>
#include <stdlib.h>
#include "presentacion.h"

// Constantes simbólicas

#define TRUE  1
#define FALSE 0


// prueba si una reina está bien colocada
// ---------------------------------------
// La reina de la fila i está bien colocada si no está
// en la columna ni en la misma diagonal que cualquiera
// de las reinas de las filas anteriores
//
// Parámetros
//   fila   - Fila de la reina cuya posición queremos validar
//   reinas - Vector con las posiciones de las reinas
//   n      - Número de reinas


int prueba (int fila, int reinas[], int n)
{
  int i;

  for (i=0; i<fila; i++)
      if (  ( reinas[i]==reinas[fila] )                      // Misma
columna
         || ( abs(fila-i) == abs(reinas[fila]-reinas[i]) ) ) // Misma
diagonal
         return FALSE;

  return TRUE;
}


// Mostrar el tablero con las reinas
// ---------------------------------
// Parámetros:
//   reinas - Vector con las posiciones de las distintas reinas
//   n      - Número de reinas

void dibuja (int reinas[], int n, int *alto)
{
  int i,j,x=0,y=*alto, aux=0;

  for (i=0; i<n; i++) {
        x=0;
      for (j=0; j<n; j++) {

          if (reinas[i]==j){
            MoverCursor(x,y);
            printf("##");
            MoverCursor(x,y+1);
```

```
            printf("##");
          }else{
            if((j+aux)%2 == 0){
                  MoverCursor(x,y);
                  printf(",,");
                  MoverCursor(x,y+1);
                  printf(",,");
            }else{
                  MoverCursor(x,y);
                  printf("..");
                  MoverCursor(x,y+1);
                  printf("..");
            }
          }
          x+=3;
      }
      y+=2;
      aux=1-aux;
      //printf(" %d %d\n",i,reinas[i]);
      *alto = y+3;
  }
}


// Colocación de una reina
// -----------------------
// Parámetros
//   fila   - Fila de la reina que queremos colocar
//   reinas - Vector con las posiciones de las reinas
//   n      - Número de reinas

void nReina (int fila, int reinas[], int n, int *alto)
{
  int ok = FALSE;

  if (fila<n) {

     // Quedan reinas por colocar

     for (reinas[fila]=0; reinas[fila]<n; reinas[fila]++) {

         // Comprobamos si la posición
         // de la reina actual es válida

         if (prueba(fila,reinas,n)) {

            // Si es así, intentamos colocar
            // las reinas restantes
            nReina (fila+1, reinas, n, alto);
         }
     }

  } else {

     // No quedan reinas por colocar (solución)
     dibuja(reinas,n,alto);
  }
```

```
  return;
}

// Programa principal
// ------------------

int main ()
{
  int *reinas;  // Vector con las posiciones de las reinas de cada fila
  int nreinas;  // Número de reinas
  int i;        // Contador
  int alto=0;

  printf("Ingresa el numero de reinas: ");
  scanf("%d", &nreinas);

  // Colocar las reinas en el tablero

  // Crear vector dinámicamente

      reinas = (int*) malloc ( nreinas*sizeof(int) );

  // Inicializar vector:
  // (inicialmente, ninguna reina está colocada)
      system("cls");
  for (i=0; i<nreinas; i++)
    reinas[i] = -1;
   // Colocar reinas (algoritmo recursivo)
   nReina(0,reinas,nreinas, &alto);
   // Liberar memoria
   free (reinas);
}
```

Bibliografía

[1] J. B. Aranda. [En línea]. Available: http://jabaier.sitios.ing.uc.cl/iic2552/backtracking.pdf. [Último acceso: 11 11 2017].