Contents

1 Module Conexion_db : modulo para realizar la conexion a la base de datos

```
exception E of string
val get : 'a option -> 'a
val get_rows :
  string ->
  (Sqlite3.row_not_null -> Sqlite3.headers -> unit) -> Sqlite3.db -> unit
```

funcion para obtener las entradas de una tabla y aplicar una funcion a cada fila que regresa como resultado

```
val open_db_conection : string -> Sqlite3.db
```

abre la conexion a la base de datos

```
val close_db_conection : Sqlite3.db -> unit
```

Cierra la conexion a la base de datos

2 Module Grafica : Modulo para representar graficas de ciudades

```
exception ErrorGrafica of string
```

Exepcion para indicar errores en las operaciones de la grafica

```
type ciudad = {
  id : int ;
  nombre : string ;
  pais : string ;
  poblacion : int ;
  latitud : float ;
```

```
  longitud : float ;
  vecinos : (int, float) Hashtbl.t ;
}
```

record para guradar la informacion de una ciudad

`val tamano : int Pervasives.ref`

`val orden : int Pervasives.ref`

`val initgraf : int -> ciudad array`

funcion que inicializa la grafica. Las graficas se representan como un arreglo de referencias a listas. Cada entrada del arreglo representa un verice y cada lista contiene las adyacencias del nodo

`val agrega : 'a array -> 'a -> unit`

funcion para agregar una ciudad a la grafica

`val conectados : ciudad array -> int -> int -> bool`

Valida si es posible conectar dos nodos o si ya existe una arista que los une en esa direccion

`val conecta : ciudad array -> int -> int -> float -> unit`

conecta dos nodos de una grafica. Al conecta los nodos i y j se agrega a la entrada i del arreglo el par (j,p) donde p es el peso de la arista

`val getPeso : ciudad array -> int -> int -> float`

regresa el peso de la arsita entre dos nodos a partir del indice de ambos

`val get_vecinos : ciudad -> int array`

funcion para obtener los vecinos de un verice

Returns arreglo de indice de vecinos

3 Module `Solucion` : Modulo para operar las soluciones

`val inicializa : int -> unit`

funcion para inicializar el generador de numeros seudo aleatorios con la semilla

`val ant1 : int Pervasives.ref`

variable que guarda la primera posicion del cambio anterior que se genero en la funcion vecino

`val ant2 : int Pervasives.ref`

variable que guarda la segunda posicion del cambio anterior que se genero en la funcion vecino

`val max_dist_path : float Pervasives.ref`

variable que guarda la maxima distancia entre dos ciudades Se debe inicializar con la funcion init_max_avg

`val avg_path : float Pervasives.ref`

variable que guarda la distancia promedio entre dos ciudades Se debe inicializar con la funcion init_max_avg

`val vecino : 'a array -> unit`

funcion que calcula el "vecino" de una solucion La funcion solo realiza un intercambio de valores y guarda los indices de los valores intervambiados para poder volver a la solucion de que se partio

`val regresa : 'a array -> unit`

funcion que regresa el cambio hecho por vecino Solo funciona para una anterior

`val permuta : 'a array -> unit`

funcion para conseguir una permutacion aleatoria de la instancia inicializa y usarla como solucion inicial de la heuristica

`val init_max_avg : Grafica.ciudad array -> int array -> unit`

Funcion que inicializa la distancia promedio y distancia maxima entre dos ciudades en la solucion

`val f : Grafica.ciudad array -> int array -> float`

funcion de costo de la heuristica

`val genera_solucion : Grafica.ciudad array -> int -> int array`

funcion para obtendran una instancia aleatoria que tenga una solucion almenos

`val factible : Grafica.ciudad array -> int array -> int * bool`

funcion que indica si una solucion es factible y el numero de desconexiones presenta

4 Module Svg : Modulo para generar las graficas del la solucion y el comportamiento de las evaluaciones

```
val file : string -> string -> string
val creaLinea :
  Pervasives.out_channel -> float -> float -> float -> float -> string -> unit
val creaLineaf :
  Pervasives.out_channel -> float -> float -> float -> float -> string -> unit
val creaCirculo :
```

```
  Pervasives.out_channel -> float -> float -> int -> string -> unit
val creaCirculof :
  Pervasives.out_channel -> float -> float -> int -> string -> unit
val encabezado : Pervasives.out_channel -> int -> int -> unit
val svg : Pervasives.out_channel -> Grafica.ciudad array -> int array -> unit
val guarda : Grafica.ciudad array -> int array -> int -> unit
val gfuncion : (int, float) Hashtbl.t -> int -> unit
```

5 Module `Umbrales` : Modulo principal que implementa la heristica de aceptacion por umbrales

`val mejor_solucion : int array Pervasives.ref`

Variable que guarda la solucion mejor evaluada de durante todo el proceso

`val mejor_fs : float Pervasives.ref`

Variable que guarda la mejor evaluacion de una solucion

`val evals : (int, float) Hashtbl.t`

diccionario que guarda las evaluaciones de soluciones aceptadas

`val aceptadas : int Pervasives.ref`

variable que guarda el numero de soluciones aceptadas

```
val calcula_lote :
  Grafica.ciudad array -> float -> int array -> float * int array
```

Funcion que va construyendo los lotes y regresando el promedio de la evaluacion de la funcion de costo

```
val aceptacion_por_umbrales :
  Grafica.ciudad array -> float -> int array -> unit
```

funcion principal para la heristica

`val conecta_ciudad : Grafica.ciudad array -> string array -> 'a -> unit`

funcion que se aplicara a cada valor que se regrese de la tabla "connections" y a partir de cada fila conecta dos ciudades

`val agrega_ciudad : Grafica.ciudad array -> string array -> 'a -> unit`

funcion que se aplicara a cada valor que se regrese de la tabla "cities" y a partir de cada fila de esta se crea una ciudad de la grafica

`val umbrales : Grafica.ciudad array -> int array -> int -> unit`

funcion que realiza el preproceso para la heristica y registra los resultados

`val construye_grafica : unit -> Grafica.ciudad array`

Funcion que a partir de la base de datos predefinida construye la grafica de ciudades

Returns la grafica de ciudades para el problema tsp

`val lee_solucion : unit -> int array`

Funcion que lee el archivo que se pasa como argumento y construye solucion inicial con la que se trabajara la heuristica